NEC

# Preliminary User's Manual

# μPD98501

## Communication Controller

Document No. S14767EJ1V0UM00 (1st edition)
Date Published April 2000 N CP(K)

© NEC Corporation 2000
Printed in Japan

**[MEMO]**

## SUMMARY OF CONTENTS


**CHAPTER 1 INTRODUCTION ..... 25**

**CHAPTER 2 VR4120A CPU ..... 55**

**CHAPTER 3 SYSTEM CONTROLLER ..... 221**

**CHAPTER 4 ATM CELL PROCESSOR ..... 265**

**CHAPTER 5 ETHERNET CONTROLLER ..... 339**

**CHAPTER 6 USB CONTROLLER ..... 373**

**CHAPTER 7 UART ..... 445**

**CHAPTER 8 TIMER ..... 457**

**CHAPTER 9 MICRO WIRE ..... 461**

**APPENDIX A MIPS III INSTRUCTION SET DETAILS ..... 465**

**APPENDIX B MIPS16 INSTRUCTION SET FORMAT ..... 625**

**APPENDIX C VR4120A COPROCESSOR 0 HAZARDS ..... 665**

**[MEMO]**

## NOTES FOR CMOS DEVICES

### ① PRECAUTION AGAINST ESD FOR SEMICONDUCTORS

Note:

Strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it once, when it has occurred. Environmental control must be adequate. When it is dry, humidifier should be used. It is recommended to avoid using insulators that easily build static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work bench and floor should be grounded. The operator should be grounded using wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with semiconductor devices on it.

### ② HANDLING OF UNUSED INPUT PINS FOR CMOS

Note:

No connection for CMOS device inputs can be cause of malfunction. If no connection is provided to the input pins, it is possible that an internal input level may be generated due to noise, etc., hence causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using a pull-up or pull-down circuitry. Each unused pin should be connected to $V_{DD}$ or GND with a resistor, if it is considered to have a possibility of being an output pin. All handling related to the unused pins must be judged device by device and related specifications governing the devices.

### ③ STATUS BEFORE INITIALIZATION OF MOS DEVICES

Note:

Power-on does not necessarily define initial status of MOS device. Production process of MOS does not define the initial operation status of the device. Immediately after the power source is turned ON, the devices with reset function have not yet been initialized. Hence, power-on does not guarantee out-pin levels, I/O settings or contents of registers. Device is not initialized until the reset signal is received. Reset operation must be executed immediately after power-on for devices having reset function.

**VR4100, VR4102, VR4111, VR4120A, VR4305, VR4310, VR5000, VR10000, VR Series, VR4000 Series, VR4100 Series, and EEPROM are trademarks of NEC Corporation.**
**Micro Wire is a trademark of National Semiconductor Corp.**
**iAPX is a trademark of Intel Corp.**
**DEC VAX is a trademark of Digital Equipment Corp.**
**UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd.**
**Ethernet is a trademark of Xerox Corp.**
**MIPS is a trademark of MIPS Technologies, Inc.**

- **The information contained in this document is being issued in advance of the production cycle for the device. The parameters for the device may change before final production or NEC Corporation, at its own discretion, may withdraw the device prior to its production.**
- **Not all devices/types available in every country. Please check with local NEC representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Corporation. NEC Corporation assumes no responsibility for any errors which may appear in this document.
- NEC Corporation does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from use of a device described herein or any other liability arising from use of such device. No license, either express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Corporation or others.
- Descriptions of circuits, software, and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software, and information in the design of the customer's equipment shall be done under the full responsibility of the customer. NEC Corporation assumes no responsibility for any losses incurred by the customer or third parties arising from the use of these circuits, software, and information.
- While NEC Corporation has been making continuous effort to enhance the reliability of its semiconductor devices, the possibility of defects cannot be eliminated entirely. To minimize risks of damage or injury to persons or property arising from a defect in an NEC semiconductor device, customers must incorporate sufficient safety measures in its design, such as redundancy, fire-containment, and anti-failure features.
- NEC devices are classified into the following three quality grades:
  "Standard", "Special", and "Specific". The Specific quality grade applies only to devices developed based on a customer designated "quality assurance program" for a specific application. The recommended applications of a device depend on its quality grade, as indicated below. Customers must check the quality grade of each device before using it in a particular application.
  - Standard: Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots
  - Special: Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support)
  - Specific: Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems or medical equipment for life support, etc.

  The quality grade of NEC devices is "Standard" unless otherwise specified in NEC's Data Sheets or Data Books. If customers intend to use NEC devices for applications other than those specified for Standard quality grade, they should contact an NEC sales representative in advance.

M5D 98. 12

# PREFACE

**Readers**

This manual is intended for engineers who need to be familiar with the capability of the μPD98501 in order to develop application systems based on it.

**Purpose**

The purpose of this manual is to help users understand the hardware capabilities (listed below) of the μPD98501.

**Configuration**

This manual consists of the following chapters:

- Introduction
- VR4120A™ CPU
- System controller
- ATM cell processor
- Ethernet™ controller
- USB controller
- UART
- Timer
- Micro Wire™

**Guidance**

Readers of this manual should already have a general knowledge of electronics, logic circuits, and microcomputers.
To gain an overall understanding of the function of the μPD98501:
→ Read through all the chapters, in sequence.

To check the electrical characteristics of the μPD98501:
→ Refer to the separate data sheet.

**Notation**

This manual uses the following conventions:

| | |
|---|---|
| Data bit significance: | High-order bits on the left side;<br>low-order bits on the right side |
| Active low: | XXXX_B (Pin and signal names are suffixed with _B.) |
| **Note**: | Explanation of an indicated part of text |
| **Caution**: | Information requiring the user's special attention |
| **Remark**: | Supplementary information |
| Numerical value: | Binary ... xxxx or xxxxB<br>Decimal ... xxxx<br>Hexadecimal ... xxxxH |

**Related Documents**

Use this manual in combination with the following documents.
The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

- μPD98501 Data Sheet: To be prepared

**[MEMO]**

**CONTENTS**


**CHAPTER 1 INTRODUCTION ..... 25**
**1.1 Features ..... 25**
**1.2 Ordering Information ..... 25**
**1.3 System Configuration ..... 26**
**1.4 Block Diagram (Summary) ..... 27**
**1.5 Block Diagram (Detail) ..... 28**
1.5.1 VR4120A RISC processor core ..... 28
1.5.2 IBUS ..... 29
1.5.3 System controller ..... 30
1.5.4 ATM cell processor ..... 31
1.5.5 Ethernet controller ..... 32
1.5.6 USB controller ..... 33
**1.6 Pin Configuration (Bottom View) ..... 34**
**1.7 Pin Function ..... 37**
1.7.1 System PLL power supply ..... 37
1.7.2 USB PLL power supply ..... 37
1.7.3 System control interface ..... 38
1.7.4 Memory interface ..... 39
1.7.5 ATM interface ..... 41
1.7.6 Ethernet interface ..... 43
1.7.7 USB interface ..... 44
1.7.8 UART/Micro Wire interface ..... 45
1.7.9 Parallel port interface ..... 45
1.7.10 Boundary scan interface ..... 45
1.7.11 N-Wire interface ..... 46
**1.8 I/O Register Map ..... 47**
**1.9 Memory Map ..... 51**
**1.10 Reset Configuration ..... 52**
**1.11 Interrupts ..... 53**
**1.12 Clock Control Unit ..... 54**

**CHAPTER 2 VR4120A ..... 55**
**2.1 Overview for VR4120A ..... 55**
2.1.1 Internal block configuration ..... 56
2.1.2 VR4120A registers ..... 57
2.1.3 VR4120A instruction set overview ..... 58
2.1.4 Data formats and addressing ..... 60
2.1.5 Coprocessors (CP0) ..... 62
2.1.6 Floating-point unit (FPU) ..... 63
2.1.7 CPU core memory management system (MMU) ..... 64
2.1.8 Translation lookaside buffer (TLB) ..... 64
2.1.9 Operating modes ..... 65
2.1.10 Cache ..... 65
2.1.11 Instruction Pipeline ..... 65

**2.2 MIPS III Instruction Set Summary ..... 66**
2.2.1 MIPS III ISA instruction formats ..... 66
2.2.2 Instruction classes ..... 67
**2.3 MIPS16 Instruction Set ..... 84**
2.3.1 Features ..... 84
2.3.2 Register set ..... 84
2.3.3 ISA mode ..... 86
2.3.4 Types of Instructions ..... 87
2.3.5 Instruction format ..... 89
2.3.6 MIPS16 operation code bit encoding ..... 93
2.3.7 Outline of instructions ..... 95
**2.4 Pipeline ..... 112**
2.4.1 Pipeline stages ..... 112
2.4.2 Branch delay ..... 117
2.4.3 Load delay ..... 118
2.4.4 Pipeline operation ..... 119
2.4.5 Interlock and exception handling ..... 125
2.4.6 Program compatibility ..... 131
**2.5 Memory Management System ..... 132**
2.5.1 Translation lookaside buffer (TLB) ..... 132
2.5.2 Virtual address space ..... 133
2.5.3 Physical address space ..... 145
2.5.4 System control coprocessor ..... 148
2.5.5 CP0 registers ..... 150
**2.6 Exception Processing ..... 162**
2.6.1 Exception processing operation ..... 162
2.6.2 Precision of exceptions ..... 163
2.6.3 Exception processing registers ..... 163
2.6.4 Details of exceptions ..... 176
2.6.5 Exception processing and servicing flowcharts ..... 193
**2.7 Initialization Interface ..... 200**
2.7.1 Cold reset ..... 200
2.7.2 Soft reset ..... 200
2.7.3 VR4120A processor modes ..... 200
**2.8 Cache Memory ..... 203**
2.8.1 Memory organization ..... 203
2.8.2 Cache organization ..... 204
2.8.3 Cache operations ..... 206
2.8.4 Cache states ..... 207
2.8.5 Cache state transition diagrams ..... 208
2.8.6 Cache data integrity ..... 209
2.8.7 Manipulation of the caches by an external agent ..... 216
**2.9 CPU Core Interrupts ..... 217**
2.9.1 Non-maskable interrupt (NMI) ..... 217
2.9.2 Ordinary interrupts ..... 217
2.9.3 Software interrupts generated in CPU core ..... 217
2.9.4 Timer interrupt ..... 217
2.9.5 Asserting interrupts ..... 218

**CHAPTER 3 SYSTEM CONTROLLER ..... 221**
**3.1 Overview ..... 221**
3.1.1 CPU interface ..... 221
3.1.2 Memory interface ..... 221
3.1.3 IBUS interface ..... 221
3.1.4 UART ..... 222
3.1.5 Timer ..... 222
3.1.6 Interrupt controller ..... 222
3.1.7 DSU (Deadman's SW UNIT) ..... 222
3.1.8 System block diagram ..... 223
3.1.9 Data flow diagram ..... 224
**3.2 Registers ..... 225**
3.2.1 Register summary ..... 225
3.2.2 General registers ..... 227
**3.3 CPU Interface ..... 237**
3.3.1 Overview ..... 237
3.3.2 Data rate control ..... 237
3.3.3 Address decoding ..... 237
3.3.4 Endian conversion ..... 237
3.3.5 I/O performance ..... 239
**3.4 Memory Interface ..... 240**
3.4.1 Overview ..... 240
3.4.2 Memory regions and devices ..... 240
3.4.3 Memory signal connections ..... 241
3.4.4 Memory performance ..... 242
3.4.5 Memory control registers ..... 243
3.4.6 Boot ROM ..... 250
3.4.7 SDRAM ..... 253
3.4.8 SDRAM refresh ..... 256
3.4.9 Memory-to-CPU prefetch FIFO ..... 256
3.4.10 CPU-to-memory write FIFO ..... 256
3.4.11 SDRAM memory initialization ..... 257
**3.5 IBUS Interface Register ..... 258**
3.5.1 ITCNTR (IBUS timeout timer control register) ..... 258
3.5.2 ITSETR (IBUS timeout timer set register) ..... 258
**3.6 DSU (Deadman's SW Unit) ..... 259**
3.6.1 Overview ..... 259
3.6.2 Registers ..... 259
3.6.3 DSU register setting flow ..... 260
**3.7 Endian Mode Software Issues ..... 261**
3.7.1 Overview ..... 261
3.7.2 Endian modes ..... 261

**CHAPTER 4 ATM CELL PROCESSOR ..... 265**
**4.1 Overview ..... 265**
4.1.1 Function features ..... 265
4.1.2 Block diagram of ATM cell processor ..... 266
4.1.3 ATM cell processing operation overview ..... 269

**4.2 Memory Space ..... 275**
4.2.1 Work RAM and register space ..... 276
4.2.2 Shard memory ..... 276
**4.3 Interruption ..... 277**
**4.4 Registers for ATM Cell Processing ..... 277**
4.4.1 Register map ..... 277
4.4.2 A_GMR (General Mode Register) ..... 279
4.4.3 A_GSR (General Status Register) ..... 280
4.4.4 A_IMR (Interrupt Mask Register) ..... 282
4.4.5 A_RQU (Receiving Queue Underrun Register) ..... 283
4.4.6 A_RQA (Receiving Queue Alert Register) ..... 283
4.4.7 A_VER (Version Register) ..... 283
4.4.8 A_CMR (Command Register) ..... 283
4.4.9 A_CER (Command Extension Register) ..... 284
4.4.10 A_MSA0-A_MSA3 (Mailbox Start Address Register) ..... 284
4.4.11 A_MBA0-A_MBA3 (Mailbox Bottom Address Register) ..... 284
4.4.12 A_MTA0-A_MTA3 (Mailbox Tail Address Register) ..... 285
4.4.13 A_MWA0-A_MWA3 (Mailbox Write Address Register) ..... 285
4.4.14 A_RCC (Valid Receive Cell Counter Register) ..... 285
4.4.15 A_TCC (Valid Transmit Cell Counter Register) ..... 285
4.4.16 A_RUEC (Receive Unprovisioned VPI/VCI Error Cell Counter Register) ..... 286
4.4.17 A_RIDC (Receive Internal Dropped Cell Counter Register) ..... 286
4.4.18 A_APR (ABR Parameter Register) ..... 286
4.4.19 A_T1R (T1 Time Register) ..... 286
4.4.20 A_TSR (Time Stamp Register) ..... 286
4.4.21 A_IBBAR (IBUS Base Address Register) ..... 287
4.4.22 A_INBAR (Instruction Base Address register) ..... 287
4.4.23 A_UMCMD (UTOPIA Management I/F Command Register) ..... 287
**4.5 Data Structure ..... 288**
4.5.1 Tx packet ..... 288
4.5.2 Rx pool structure ..... 291
**4.6 Initialize ..... 297**
4.6.1 Before starting RISC core ..... 297
4.6.2 After RISC core's F/W is starting ..... 298
4.6.3 Work RAM initialization ..... 299
**4.7 Commands ..... 300**
4.7.1 Set_Link_Rate command ..... 301
4.7.2 Open_Channel command ..... 302
4.7.3 Close_Channel command ..... 303
4.7.4 Open_IP_Channel command ..... 304
4.7.5 Close_IP_Channel command ..... 305
4.7.6 Tx_Ready command ..... 306
4.7.7 IP_Flow_Ready command ..... 307
4.7.8 Add_Buffer command ..... 308
4.7.9 Indirect_Access command ..... 309
4.7.10 Set_Rx_Congestion command ..... 309
4.7.11 BRM_Tx command ..... 310

**4.8 Operations ... 311**
4.8.1 Work RAM usage ... 311
4.8.2 Transmission function ... 313
4.8.3 Receiving function ... 325
4.8.4 Mailbox ... 331
4.8.5 ABR function ... 332

**CHAPTER 5 ETHERNET CONTROLLER ... 339**
**5.1 Overview ... 339**
5.1.1 Features ... 339
5.1.2 Block diagram of Ethernet controller block ... 339
**5.2 Register ... 341**
5.2.1 Statistics counter registers ... 343
5.2.2 DMA and FIFO management registers ... 345
5.2.3 Interrupt and configuration registers ... 345
5.2.4 Detail of MAC control registers ... 346
5.2.5 Detail of DMA and FIFO management registers ... 357
5.2.6 Detail of interrupt and configuration registers ... 363
**5.3 Operation ... 365**
5.3.1 Initialization ... 365
5.3.2 Buffer structure for Ethernet block ... 365
5.3.3 Buffer descriptor format ... 366
5.3.4 Frame transmission ... 367
5.3.5 Frame reception ... 370

**CHAPTER 6 USB CONTROLLER ... 373**
**6.1 Overview ... 373**
6.1.1 Features ... 373
6.1.2 Internal block diagram ... 374
**6.2 Register Set ... 375**
6.2.1 Register map ... 375
6.2.2 Explanation of registers ... 377
**6.3 USB Attachment Sequence ... 409**
**6.4 Initialization ... 410**
6.4.1 Receive pool setting ... 411
6.4.2 Send/receive mailbox setting ... 411
**6.5 Data Send Function ... 413**
6.5.1 Overview of send processing ... 413
6.5.2 Send buffer configuration ... 413
6.5.3 Data send modes ... 416
6.5.4 VR4120A RISC processor processing at data sending ... 417
6.5.5 USB controller processing at data sending ... 420
6.5.6 Tx indication ... 422
**6.6 Data Receive Function ... 423**
6.6.1 Overview of receive processing ... 423
6.6.2 Receive buffer configuration ... 424
6.6.3 Receive pool settings ... 426
6.6.4 Data receive mode ... 427

6.6.5 VR4120A RISC processor receive processing....430
6.6.6 USB controller receive processing....431
6.6.7 Detection of errors on USB....434
6.6.8 Rx data corruption on isochronous EndPoint....436
6.6.9 Rx FIFO overrun....437
6.6.10 Rx indication....438
**6.7 Power Management....440**
6.7.1 Suspend....440
6.7.2 Resume....441
6.7.3 Remote wake up....442
**6.8 Loopback Mode....443**
**6.9 Example of Connection....444**

**CHAPTER 7 UART....445**
**7.1 Overview....445**
**7.2 UART Block Diagram....445**
**7.3 UART Registers....446**
7.3.1 UART Receiver data Buffer Register (UARTRBR) (80H, DLAB = 0, R)....446
7.3.2 UART Interrupt Enable Register (UARTIER) (84H, DLAB = 1, R/W)....446
7.3.3 UART Divisor Latch LSB Register (UARTDLL) (80H, DLAB = 1, R/W)....447
7.3.4 UART Divisor Latch MSB Register (UARTDLM) (84H, DLAB = 1, R/W)....447
7.3.5 UART Interrupt ID Register (UARTIIR) (88H, R)....449
7.3.6 UART FIFO Control Register (UARTFCR) (88H, W)....450
7.3.7 UART Line Control Register (UARTLCR) (8CH, R/W)....451
7.3.8 UART Modem Control Register (UARTMCR) (90H, R/W)....452
7.3.9 UART Line Status Register (UARTLSR) (94H, R/W)....453
7.3.10 UART Modem Status Register (UARTMSR) (98H, R/W)....454
7.3.11 UART Scratch Register (UARTSCR) (9CH, R/W)....455

**CHAPTER 8 TIMER....457**
**8.1 Overview....457**
**8.2 Block Diagram....457**
**8.3 Timer Registers....458**
8.3.1 Timer Mode Register (TMMR) (B0H, R/W)....458
8.3.2 Timer CH0 Count Set Register (TM0CSR) (B4H, R/W)....458
8.3.3 Timer CH1 Count Set Register (TM1CSR) (B8H, R/W)....458
8.3.4 Timer CH0 Current Count Register (TM0CCR) (BCH, R)....458
8.3.5 Timer CH1 Current Count Register (TM1CCR) (C0H, R)....459

**CHAPTER 9 MICRO WIRE....461**
**9.1 Overview....461**
**9.2 Micro Wire Connection Diagram....461**
**9.3 Operations....462**
9.3.1 Data read at the power up load....462
9.3.2 Accessing to EEPROM....462
**9.4 Registers....463**
9.4.1 ECCR (EEPROM Command Control Register) Address 1000_00D0h [Write Only]....463
9.4.2 ERDR (EEPROM Read Data Register) Address 1000_00D4h [Read Only]....463

9.4.3 MACAR1 (MAC Address Register 1) Address 1000_00D8h [Read Only] ..... 463
9.4.4 MACAR2 (MAC Address Register 2) Address 1000_00DCh [Read Only] ..... 463
9.4.5 MACAR3 (MAC Address Register 3) Address 1000_00E0h [Read Only] ..... 463

**APPENDIX A MIPS III INSTRUCTION SET DETAILS ..... 465**
**A.1 Instruction Notation Conventions ..... 465**
**A.2 Load and Store Instructions ..... 467**
**A.3 Jump and Branch Instructions ..... 468**
**A.4 System Control Coprocessor (CP0) Instructions ..... 469**
**A.5 CPU Instructions ..... 469**
**A.6 CPU Instruction Opcode Bit Encoding ..... 622**

**APPENDIX B MIPS16 INSTRUCTION SET FORMAT ..... 625**

**APPENDIX C VR4120A COPROCESSOR 0 HAZARDS ..... 665**

## LIST OF FIGURES (1/6)


| Figure No. | Title | Page |
|---|---|---|
| 1-1 | Examples of the μPD98501 System Configuration | 26 |
| 1-2 | Block Diagram of the μPD98501 | 27 |
| 1-3 | Block Diagram of VR4120A RISC Processor | 28 |
| 1-4 | Block Diagram of IBUS | 29 |
| 1-5 | Block Diagram of System Controller | 30 |
| 1-6 | Block Diagram of ATM Cell Processor | 31 |
| 1-7 | Block Diagram of Ethernet Controller | 32 |
| 1-8 | Block Diagram of USB Controller | 33 |
| 1-9 | Memory Map | 51 |
| 1-10 | Reset Configuration | 52 |
| 1-11 | Interrupt Signal Connection | 53 |
| 1-12 | Block Diagram of Clock Control Unit | 54 |
| 2-1 | VR4120A Core Internal Block Diagram | 55 |
| 2-2 | VR4120A Registers | 57 |
| 2-3 | CPU Instruction Formats (32-bit Length Instruction) | 58 |
| 2-4 | CPU Instruction Formats (16-bit Length Instruction) | 59 |
| 2-5 | Little-Endian Byte Ordering in Word Data | 60 |
| 2-6 | Little-Endian Byte Ordering in Double Word Data | 60 |
| 2-7 | Misaligned Word Accessing (Little-Endian) | 61 |
| 2-8 | CP0 Registers | 62 |
| 2-9 | MIPS III ISA CPU Instruction Formats | 66 |
| 2-10 | Pipeline Stages (MIPS III Instruction Mode) | 112 |
| 2-11 | Instruction Execution in the Pipeline | 113 |
| 2-12 | Pipeline Stages (MIPS16 Instruction Mode) | 113 |
| 2-13 | Instruction Execution in Pipeline (MIPS16 Instruction Mode) | 114 |
| 2-14 | Pipeline Activities (MIPS III) | 115 |
| 2-15 | Pipeline Activities (MIPS16) | 116 |
| 2-16 | Branch Delay (In MIPS III Instruction Mode) | 117 |
| 2-17 | Branch Delay (In MIPS16 Instruction Mode) | 117 |
| 2-18 | ADD Instruction Pipeline Activities (In MIPS III Instruction Mode) | 119 |
| 2-19 | ADD Instruction Pipeline Activities (In MIPS16 Instruction Mode) | 119 |
| 2-20 | JALR Instruction Pipeline Activities (In MIPS III Instruction Mode) | 120 |
| 2-21 | JALR Instruction Pipeline Activities (In MIPS16 Instruction Mode) | 120 |
| 2-22 | BEQ Instruction Pipeline Activities (In MIPS III Instruction Mode) | 121 |
| 2-23 | BEQ Instruction Pipeline Activities (In MIPS16 Instruction Mode) | 121 |
| 2-24 | TLT Instruction Pipeline Activities | 122 |
| 2-25 | LW Instruction Pipeline Activities (In MIPS III Instruction Mode) | 123 |
| 2-26 | LW Instruction Pipeline Activities (In MIPS16 Instruction Mode) | 123 |
| 2-27 | SW Instruction Pipeline Activities (In MIPS III Instruction Mode) | 124 |
| 2-28 | SW Instruction Pipeline Activities (In MIPS16 Instruction Mode) | 124 |
| 2-29 | Relationship among Interlocks, Exceptions, and Faults | 125 |

## LIST OF FIGURES (2/6)


| Figure No. | Title | Page |
|---|---|---|
| 2-30 | Exception Detection | 127 |
| 2-31 | Data Cache Miss Stall | 128 |
| 2-32 | CACHE Instruction Stall | 128 |
| 2-33 | Load Data Interlock | 129 |
| 2-34 | MD Busy Interlock | 129 |
| 2-35 | Virtual-to-Physical Address Translation | 133 |
| 2-36 | 32-bit Mode Virtual Address Translation | 134 |
| 2-37 | 64-bit Mode Virtual Address Translation | 135 |
| 2-38 | User Mode Address Space | 136 |
| 2-39 | Supervisor Mode Address Space | 138 |
| 2-40 | Kernel Mode Address Space | 140 |
| 2-41 | $\mu$PD98501 Physical Address Space | 145 |
| 2-42 | CP0 Registers and TLB | 148 |
| 2-43 | Format of a TLB Entry | 149 |
| 2-44 | Index Register | 150 |
| 2-45 | Random Register | 150 |
| 2-46 | EntryLo0 and EntryLo1 Registers | 151 |
| 2-47 | Page Mask Register | 152 |
| 2-48 | Positions Indicated by Wired Register | 153 |
| 2-49 | Wired Register | 153 |
| 2-50 | EntryHi Register | 154 |
| 2-51 | PRId Register | 155 |
| 2-52 | Config Register Format | 156 |
| 2-53 | LLAddr Register | 157 |
| 2-54 | TagLo Register | 158 |
| 2-55 | TagHi Register | 158 |
| 2-56 | TLB Address Translation | 160 |
| 2-57 | Context Register Format | 164 |
| 2-58 | BadVAddr Register Format | 165 |
| 2-59 | Count Register Format | 165 |
| 2-60 | Compare Register Format | 166 |
| 2-61 | Status Register Format | 167 |
| 2-62 | Status Register Diagnostic Status Field | 168 |
| 2-63 | Cause Register Format | 169 |
| 2-64 | EPC Register Format (When MIPS16 ISA Is Disabled) | 171 |
| 2-65 | EPC Register Format (When MIPS16 ISA Is Enabled) | 172 |
| 2-66 | WatchLo Register Format | 172 |
| 2-67 | WatchHi Register Format | 172 |
| 2-68 | XContext Register Format | 173 |
| 2-69 | Parity Error Register Format | 173 |
| 2-70 | Cache Error Register Format | 174 |
| 2-71 | ErrorEPC Register Format (When MIPS16 ISA Is Disabled) | 175 |

## LIST OF FIGURES (3/6)


| Figure No. | Title | Page |
|---|---|---|
| 2-72 | ErrorEPC Register Format (When MIPS16 ISA Is Enabled) | 176 |
| 2-73 | Common Exception Handling | 195 |
| 2-74 | TLB/XTLB Refill Exception Handling | 197 |
| 2-75 | Cold Reset Exception Handling | 199 |
| 2-76 | Soft Reset and NMI Exception Handling | 200 |
| 2-77 | Logical Hierarchy of Memory | 204 |
| 2-78 | Cache Support | 205 |
| 2-79 | Instruction Cache Line Format | 206 |
| 2-80 | Data Cache Line Format | 206 |
| 2-81 | Cache Data and Tag Organization | 207 |
| 2-82 | Data Cache State Diagram | 209 |
| 2-83 | Instruction Cache State Diagram | 209 |
| 2-84 | Data Check Flow on Instruction Fetch | 210 |
| 2-85 | Data Check Flow on Load Operations | 210 |
| 2-86 | Data Check Flow on Store Operations | 211 |
| 2-87 | Data Check Flow on Index_Invalidate Operations | 211 |
| 2-88 | Data Check Flow on Index_Writeback_Invalidate Operations | 212 |
| 2-89 | Data Check Flow on Index_Load_Tag Operations | 212 |
| 2-90 | Data Check Flow on Index_Store_Tag Operations | 212 |
| 2-91 | Data Check Flow on Create_Dirty Operations | 213 |
| 2-92 | Data Check Flow on Hit_Invalidate Operations | 213 |
| 2-93 | Data Check Flow on Hit_Writeback_Invalidate Operations | 214 |
| 2-94 | Data Check Flow on Fill Operations | 214 |
| 2-95 | Data Check Flow on Hit_Writeback Operations | 215 |
| 2-96 | Writeback Flow | 215 |
| 2-97 | Refill Flow | 216 |
| 2-98 | Writeback & Refill Flow | 216 |
| 2-99 | Non-maskable Interrupt Signal | 218 |
| 2-100 | Hardware Interrupt Signals | 219 |
| 2-101 | Masking of Interrupt Request Signals | 220 |
| 3-1 | Bit and Byte Order of Endian Modes | 265 |
| 3-2 | Halfword Data-Array Example | 265 |
| 3-3 | Word Data-Array Example | 266 |
| 4-1 | Block Diagram of ATM Cell Processor | 268 |
| 4-2 | AAL-5 Sublayer and ATM Layer | 270 |
| 4-3 | AAL-5 Sublayer and ATM Layer | 271 |
| 4-4 | ATM Cell | 272 |
| 4-5 | LLC Encapsulation | 274 |
| 4-6 | Memory Space from VR4120A and RISC Core | 275 |
| 4-7 | Work RAM and Register Space | 276 |

## LIST OF FIGURES (4/6)


| Figure No. | Title | Page |
|---|---|---|
| 4-8 | Tx Packet | 288 |
| 4-9 | Tx Buffer Elements | 289 |
| 4-10 | Tx Packet Descriptor | 290 |
| 4-11 | Tx Buffer Descriptor / Link Pointer | 291 |
| 4-12 | Rx Pool | 292 |
| 4-13 | Rx Pool Structure | 293 |
| 4-14 | Rx Pool Elements | 294 |
| 4-15 | Rx Pool Descriptor | 295 |
| 4-16 | Rx Buffer Descriptor/ Link Pointer | 296 |
| 4-17 | Transfer of F/W | 297 |
| 4-18 | Instruction RAM and Instruction Cache | 298 |
| 4-19 | Work RAM after Initialization | 299 |
| 4-20 | Set_Link_Rate Command | 301 |
| 4-21 | Open_Channel Command and Open_Channel Indication | 302 |
| 4-22 | Close_Channel Command and Close_Channel Indication | 302 |
| 4-23 | Open_IP_Channel Command and Open_IP_Channel Indication | 304 |
| 4-24 | Close_Channel Command and Close_Channel Indication | 305 |
| 4-25 | Tx_Ready Command and Tx_Ready Command Indication | 306 |
| 4-26 | IP_Flow_Ready Command | 307 |
| 4-27 | Add_Buffers Command | 308 |
| 4-28 | Indirect_Access Command | 309 |
| 4-29 | Set_Rx_Congestion Command | 309 |
| 4-30 | BRM_Tx Command | 310 |
| 4-31 | Work RAM Usage | 312 |
| 4-32 | Structure of the Transmit Queue | 316 |
| 4-33 | Packet Info Structure | 316 |
| 4-34 | Transmit Queue Packet Descriptor | 317 |
| 4-35 | Tx VC Table | 318 |
| 4-36 | Table Format | 320 |
| 4-37 | Raw Cell with CRC-10 | 321 |
| 4-38 | Send Indication Format | 321 |
| 4-39 | IP Header | 322 |
| 4-40 | TCP Header | 323 |
| 4-41 | LLC Encapsulation Format | 324 |
| 4-42 | Receive VC Table | 326 |
| 4-43 | Raw Cell Data Format | 328 |
| 4-44 | Receive Indication Format | 329 |
| 4-45 | Mailbox Structure | 331 |
| 4-46 | RM Cell Format | 334 |
| 4-47 | Rate Field ABR Parameter | 337 |

## LIST OF FIGURES (5/6)


| Figure No. | Title | Page |
|---|---|---|
| 5-1 | Block Diagram of Ethernet Controller | 340 |
| 5-2 | Tx FIFO Control Mechanism | 358 |
| 5-3 | Rx FIFO Control Mechanism | 361 |
| 5-4 | Buffer Structure for Ethernet Block | 365 |
| 5-5 | Transmit Descriptor Format | 366 |
| 5-6 | Receive Descriptor Format | 366 |
| 5-7 | Transmit Procedure | 369 |
| 5-8 | Receive Procedure | 371 |
| 6-1 | USB Controller Internal Configuration | 374 |
| 6-2 | USB Attachment Sequence | 409 |
| 6-3 | Mailbox Configuration | 412 |
| 6-4 | Division of Data into USB Packets | 413 |
| 6-5 | Send Buffer Configuration | 414 |
| 6-6 | Configuration of Send Packet Descriptors | 415 |
| 6-7 | VR4120A RISC Processor Processing at Data Sending | 417 |
| 6-8 | Send Command Issue | 418 |
| 6-9 | Send Status Register | 419 |
| 6-10 | USB Controller Send Operation Flow Chart | 420 |
| 6-11 | Send Indication Format | 422 |
| 6-12 | Division of Data into USB Packets | 423 |
| 6-13 | Receive Pool Configuration | 424 |
| 6-14 | Receive Descriptor Configuration | 425 |
| 6-15 | Buffer Directory Addition Command | 427 |
| 6-16 | Data Receiving in EndPoint0, EndPoint6 | 428 |
| 6-17 | EndPoint2, EndPoint4 Receive Normal Mode | 428 |
| 6-18 | EndPoint2, EndPoint4 Receive Assemble Mode | 429 |
| 6-19 | EndPoint2, EndPoint4 Receive Separate Mode | 429 |
| 6-20 | VR4120A RISC Processor Receive Processing | 430 |
| 6-21 | USB Controller Receive Operations (Assemble Mode) | 431 |
| 6-22 | USB Controller Receive Operation Sequence (Separate Mode) | 433 |
| 6-23 | USB Timing Errors | 434 |
| 6-24 | Example of Buffers Including Corrupted Data | 437 |
| 6-25 | Receive Indication Format | 438 |
| 6-26 | Suspend Sequence | 440 |
| 6-27 | Resume Sequence | 441 |
| 6-28 | Remote Wake Up Sequence | 442 |
| 6-29 | Data Flow in Loopback Mode | 443 |
| 6-30 | Example of Connection | 444 |

## LIST OF FIGURES (6/6)


| Figure No. | Title | Page |
|---|---|---|
| 9-1 | Micro Wire Connection | 461 |
| A-1 | VR4120A Opcode Bit Encoding | 622 |

## LIST OF TABLES (1/3)


| Table No. | Title | Page |
|---|---|---|
| 2-1 | System Control Coprocessor (CP0) Register Definitions | 63 |
| 2-2 | Number of Delay Slot Cycles Necessary for Load and Store Instructions | 67 |
| 2-3 | Byte Specification Related to Load and Store Instructions | 68 |
| 2-4 | Load/Store Instruction | 69 |
| 2-5 | Load/Store Instruction (Extended ISA) | 70 |
| 2-6 | ALU Immediate Instruction | 71 |
| 2-7 | ALU Immediate Instruction (Extended ISA) | 72 |
| 2-8 | Three-Operand Type Instruction | 72 |
| 2-9 | Three-Operand Type Instruction (Extended ISA) | 73 |
| 2-10 | Shift Instruction | 73 |
| 2-11 | Shift Instruction (Extended ISA) | 74 |
| 2-12 | Multiply/Divide Instructions | 75 |
| 2-13 | Multiply/Divide Instructions (Extended ISA) | 76 |
| 2-14 | Number of Stall Cycles in Multiply and Divide Instructions | 77 |
| 2-15 | Number of Delay Slot Cycles in Jump and Branch Instructions | 77 |
| 2-16 | Jump Instruction | 78 |
| 2-17 | Branch Instructions | 79 |
| 2-18 | Branch Instructions (Extended ISA) | 80 |
| 2-19 | Special Instructions | 81 |
| 2-20 | Special Instructions (Expanded ISA) | 81 |
| 2-21 | System Control Coprocessor (CP0) Instructions | 82 |
| 2-22 | General Registers | 85 |
| 2-23 | Special Registers | 85 |
| 2-24 | MIPS16 Instruction Set Outline | 87 |
| 2-25 | Field Definition | 89 |
| 2-26 | Bit Encoding of Major Operation Code (op) | 93 |
| 2-27 | RR Minor Operation Code (RR-Type Instruction) | 93 |
| 2-28 | RRR Minor Operation Code (RRR-Type Instruction) | 94 |
| 2-29 | RRRI-A Minor Operation Code (RRI-Type ADD Instruction) | 94 |
| 2-30 | SHIFT Minor Operation Code (SHIFT-Type Instruction) | 94 |
| 2-31 | I8 Minor Operation Code (I8-Type Instruction) | 94 |
| 2-32 | I64 Minor Operation Code (64-bit Only, I64-Type Instruction) | 95 |
| 2-33 | Base PC Address Setting | 96 |
| 2-34 | Extendable MIPS16 Instructions | 97 |
| 2-35 | Load and Store Instructions | 99 |
| 2-36 | ALU Immediate Instructions | 102 |
| 2-37 | Two-/Three-Operand Register Type | 104 |
| 2-38 | Shift Instructions | 105 |
| 2-39 | Multiply/Divide Instructions | 107 |
| 2-40 | Jump and Branch Instructions | 109 |
| 2-41 | Special Instructions | 111 |
| 2-42 | Operation in Each Stage of Pipeline (MIPS III) | 115 |

## LIST OF TABLES (2/3)


| Table No. | Title | Page |
|---|---|---|
| 2-43 | Operation in Each Stage of Pipeline (MIPS16) | 116 |
| 2-44 | Correspondence of Pipeline Stage to Interlock and Exception Conditions | 125 |
| 2-45 | Pipeline Interlock | 126 |
| 2-46 | Description of Pipeline Exception | 126 |
| 2-47 | VR Series Supported Instructions | 131 |
| 2-48 | Comparison of useg and xuseg | 137 |
| 2-49 | 32-bit and 64-bit Supervisor Mode Segments | 138 |
| 2-50 | 32-bit Kernel Mode Segments | 141 |
| 2-51 | 64-bit Kernel Mode Segments | 142 |
| 2-52 | Cacheability and xkphys Address Space | 143 |
| 2-53 | ROM Addresses (When Using 16-bit Data Bus) | 146 |
| 2-54 | Cache Algorithm | 152 |
| 2-55 | Mask Values and Page Sizes | 152 |
| 2-56 | CP0 Exception Processing Registers | 163 |
| 2-57 | Cause Register Exception Code Field | 170 |
| 2-58 | 64-Bit Mode Exception Vector Base Addresses | 176 |
| 2-59 | 32-Bit Mode Exception Vector Base Addresses | 177 |
| 2-60 | Exception Priority Order | 178 |
| 3-1 | Endian Configuration Table | 238 |
| 3-2 | Endian Translation Table in Endian Converter | 238 |
| 3-3 | External Pin Mapping | 241 |
| 3-4 | Examples of Memory Performance (4word-burst access from CPU) | 242 |
| 3-5 | Examples of Memory Performance (4word-burst access from IBUS Master) | 242 |
| 3-6 | Boot-ROM Size Configuration at Reset | 250 |
| 3-7 | Command Sequence | 251 |
| 3-8 | SDRAM Size Configuration at Reset | 253 |
| 3-9 | SDRAM Configurations Supported | 253 |
| 3-10 | SDRAM Bank Select Signals Mapping | 253 |
| 3-11 | SDRAM Word Order for Instruction-Cache Line-Fill | 254 |
| 4-1 | List of Tx Packet attribute | 290 |
| 4-2 | List of Rx Pool Attributes | 295 |
| 4-3 | Commands | 300 |
| 4-4 | Reception Errors That Can Occur During Packet Reception | 330 |
| 4-5 | Error Reporting Priorities | 331 |
| 4-6 | RM Cell Initial Value | 335 |
| 4-7 | ABR Parameter | 336 |
| 5-1 | Ethernet Controller's Register Map | 341 |
| 5-2 | MAC Control Register Map | 342 |
| 5-3 | Statistics Counter Register Map | 343 |

## LIST OF TABLES (3/3)


| Table No. | Title | Page |
|---|---|---|
| 5-4 | DMA and FIFO Management Registers Map | 345 |
| 5-5 | Interrupt and Configuration Registers Map | 345 |
| 5-6 | Attribute for Transmit Descriptor | 366 |
| 5-7 | Attribute for Receive Descriptor | 367 |
| 7-1 | Correspondence between Baud Rates and Divisors | 448 |
| 9-1 | EEPROM Initial Data | 462 |
| 9-2 | EEPROM Command List | 462 |
| A-1 | CPU Instruction Operation Notations | 466 |
| A-2 | Load and Store Common Functions | 467 |
| A-3 | Access Type Specifications for Loads/Stores | 468 |
| C-1 | VR4120A CPU Coprocessor 0 Hazards | 666 |
| C-2 | Calculation Example of CP0 Hazard and Number of Instructions Inserted | 669 |

# CHAPTER 1 INTRODUCTION

The μPD98501 is a high performance controller which can perform the protocol conversion between IP Packets and ATM Cells, which is especially suitable for ADSL modem and it includes high performance MIPS™ based 64bit RISC processor VR4120A CPU core, ATM Cell Processor, Ethernet Controller, USB controller Block, UTOPIA2 interface and SDRAM interface.

## 1.1 Features

- Includes high performance MIPS based 64-bit RISC processor VR4120A
- Can perform RTOS and network middleware (M/W) on the chip
- Includes interface for PROM and flash ROM used for storing boot program
- Includes 32bit RISC controller, as ATM Cell processor
- Software SAR processing by RISC controller affords flexibility for specification update
- Supports CBR/VBR/ABR/UBR/GFR(TBD) service classes
- Includes 2-channel 10/100Mbps Ethernet controller compliant to IEEE802.3,IEEE 802.3u and IEEE802.3x
- Can directly connect external Ethernet PHY device through 3.3V MII interface
- Includes USB full speed function controller compliant to USB specification 1.1
- Supports operation conforming to the USB Communication Device Class Specification
- Can directly connect 16Mbit and 64Mbit SDRAM as external memory
- Includes 8bit 33MHz UTOPIA level 2 interface compliant to ATM Forum af-phy-0039
- Includes boundary scan function(JTAG) compliant to IEEE 1149.1
- Includes serial N-Wire interface for field debug
- Includes Micro Wire interface
- Includes 2ch general purpose timers
- Using advanced CMOS technology
- Supply Voltage 2.5V (required 3.3V supply for 3.3V interface)
- Package 352-pin T-BGA

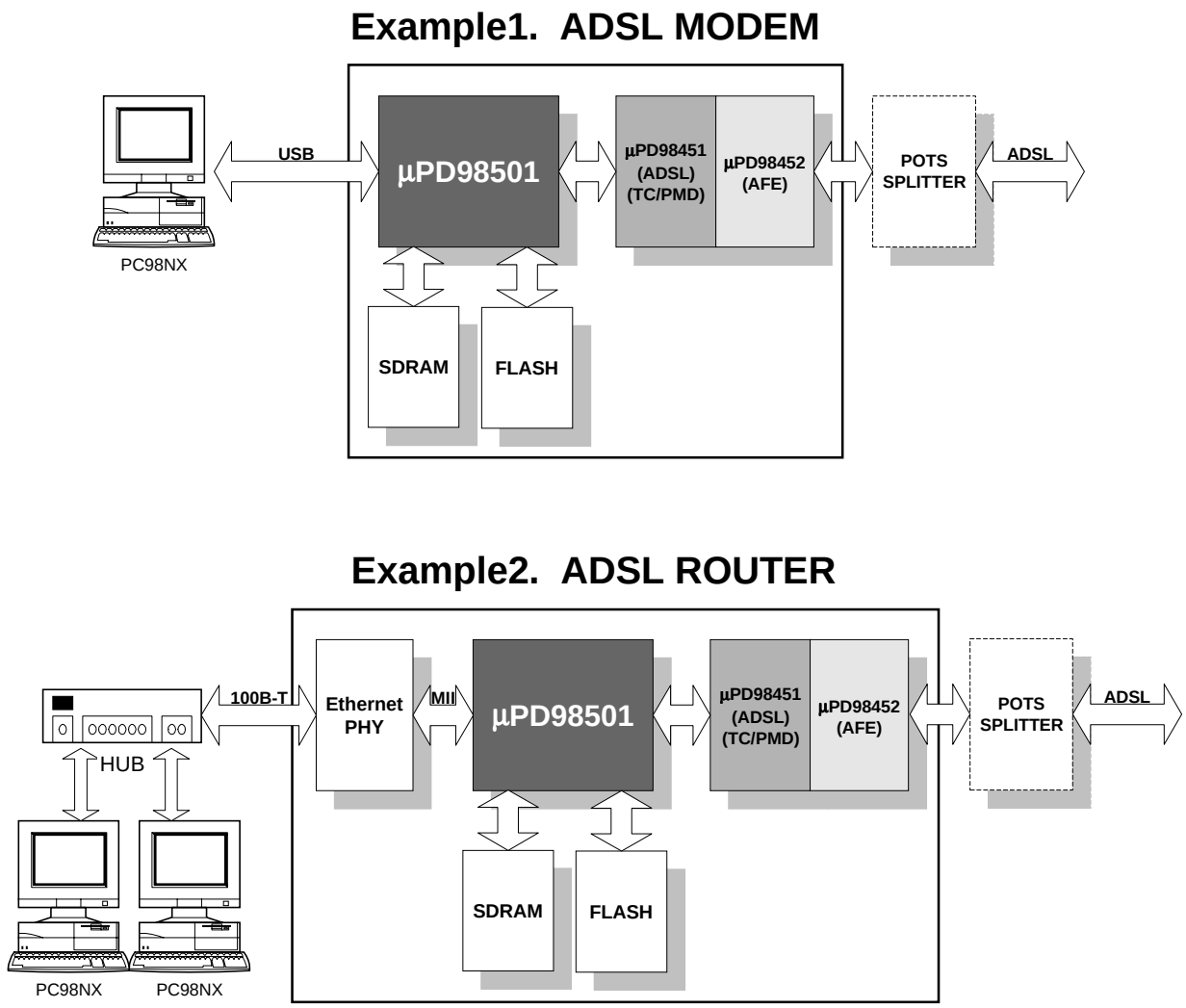
## 1.2 Ordering Information

| Part Number | Package |
|---|---|
| μPD98501N7-F6 | 352-pin Tape BGA (35 × 35) |

## 1.3 System Configuration

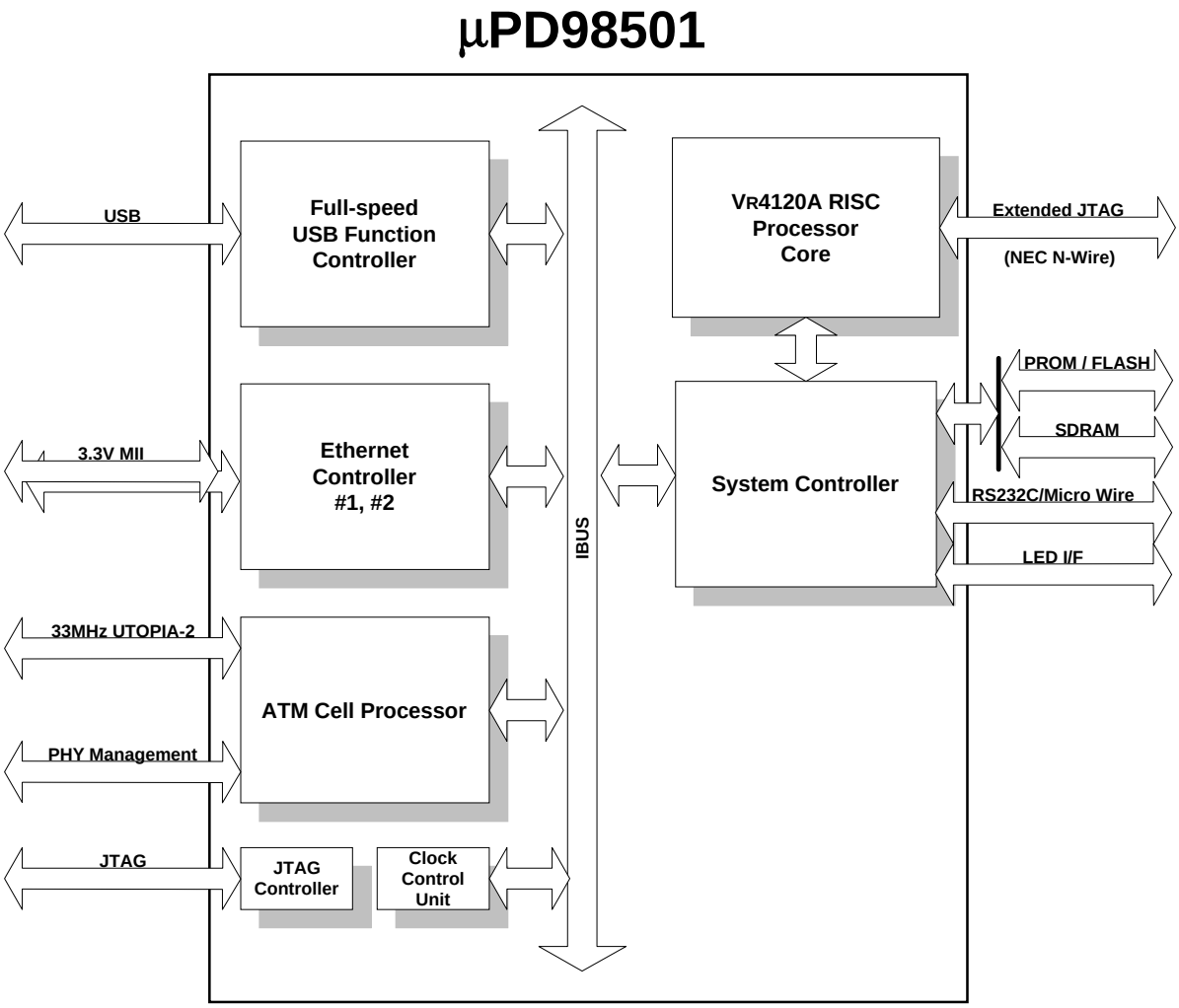
The μPD98501 The can perform bridging and routing function between ADSL/ATM interface and USB/Ethernet interface and provides this function in a single chip. By selecting user interface, examples of system configuration will be realized as shown below. USB and Ethernet functions will exclusively operate each other.

**Figure 1-1. Examples of the μPD98501 System Configuration**



## 1.4 Block Diagram (Summary)

**Figure 1-2. Block Diagram of the μPD98501**



## 1.5 Block Diagram (Detail)

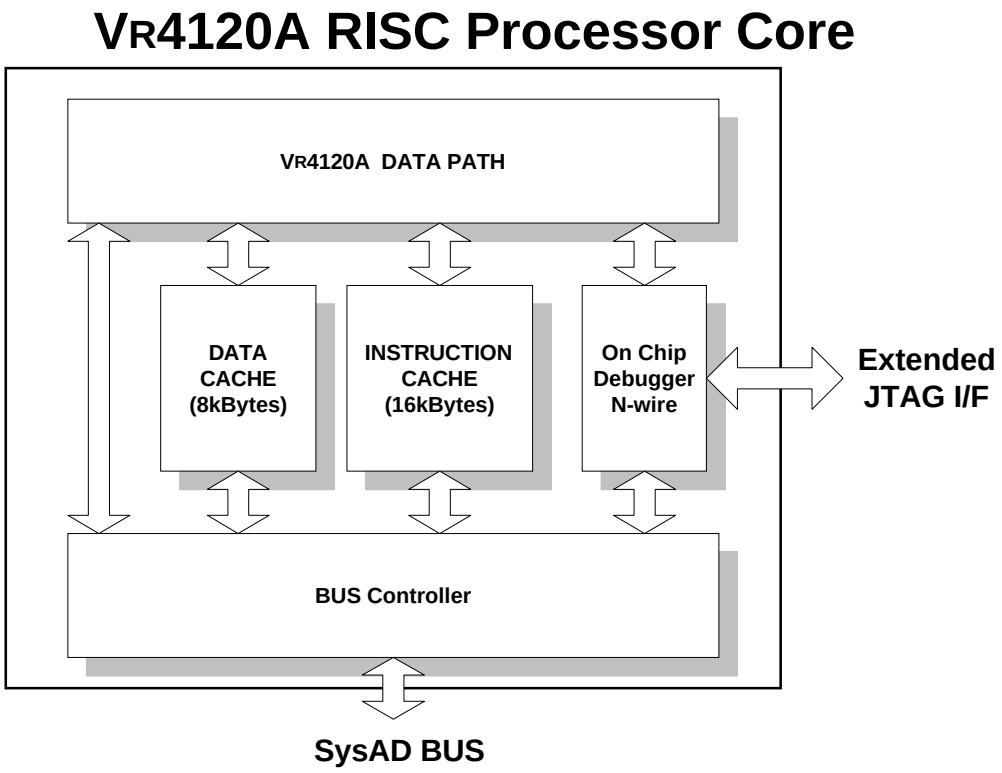
### 1.5.1 VR4120A RISC processor core

We will support real-time OS running on high performance RISC processor VR4120A core and can perform network protocols (TCP/IP, PPP, SNMP, HTTP etc) to realize ADSL router and modem. Middlewares including RTOS will be loaded to SDRAM from external PROM and Flash ROM and by setting write protected area for such a area, high speed processing will be realized together with large size instruction cache.

Features of VR4120A RISC Processor Core are as follows;

- MIPS/I/II/III and MIPS16 instruction set will be supported (FPU, LL, LLD, SC, SCD instruction will be excluded)
- Realize high speed processing of application by supporting high speed multiply and accumulate function
- Includes Large size cache memory (Instruction:16kbytes, Data:8kbytes)
- Supports up to 1T byte virtual address space by using full associative TLB
- Implements switching function between Big-Endian and Little-Endian
- Supports Real-Time trace and Debug using Extended JTAG I/F(N-Wire)

**Figure 1-3. Block Diagram of VR4120A RISC Processor**



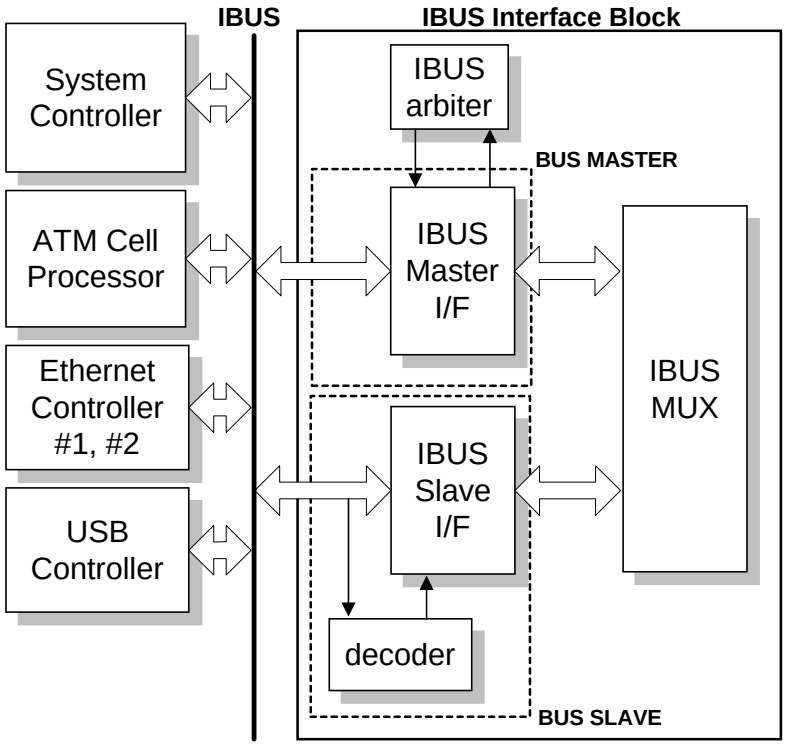
### 1.5.2 IBUS

The IBUS is a 32-bit, 66-MHz high-speed on-chip bus which enables interconnection between itself and IBUS(64-bit bus).

The IBUS supports the following bus protocols;

- Single read/write transfer
- Burst read/write transfer
- Slave lock
- Retry and disconnect
- Bus parking

**Figure 1-4. Block Diagram of IBUS**




**(Using MUX Bus Arrangement)**

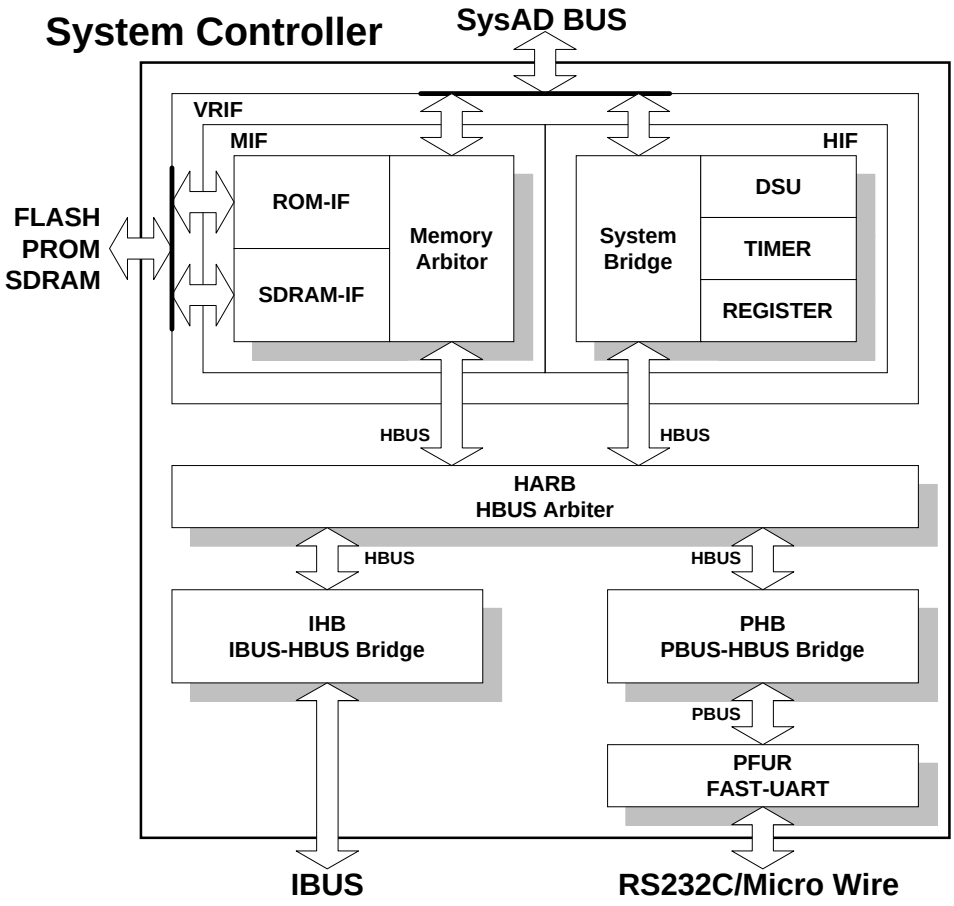
### 1.5.3 System controller

System controller is μPD98501's internal system controller. System controller provides bridging function among the VR4120A System Bus "SysAD", NEC original high speed on-chip bus "IBUS" and memory bus for SDRAM/PROM/FLASH.

Features of System controller are as follows;

- Implements 4word prefetch FIFO buffer between SysAD and Memory
- Implements 32bit×64word FIFO buffer for each TX and RX to IBUS
- Implements 32bit× 4word FIFO buffer for each TX and RX to HBUS
- Provides bus bridging function among SysAD bus and IBUS(internal bus) and MEMORY
- Supports Endian Converting function on SysAD bus
- Can directly connect 16Mbit/64Mbit SDRAM(MAX.32MBytes) and PROM/FLASH(MAX.8MBytes) memory
- Supports all VR4120A bus cycles at 66MHz or 100MHz
- PROM/FLASH data signals multiplexed on SDRAM data signals
- Supports 266 MB/sec (32bit @66MHz) bursts on IBUS
- Generates NMI and INT
- Supports NS16550 compatible Universal Asynchronous Receiver/Transmitter(UART)
- Supports separated 2ch Timer
- Supports Deadmans Switch Unit(Watch Dog Timer)
- Supports Micro Wire interface

**Figure 1-5. Block Diagram of System Controller**



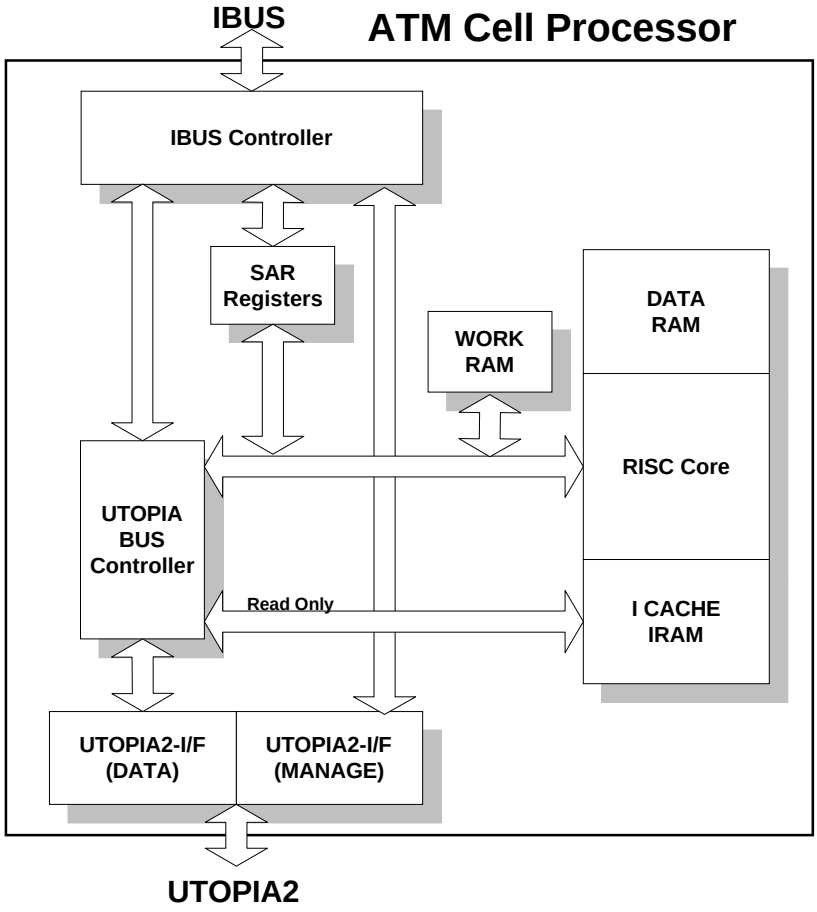
### 1.5.4 ATM cell processor

By using NEC proprietary 32bit controller, we will realize ATM Cell processor Unit. ATM Cell processing by firmware realizes more flexibility than before.

Features of ATM Cell Processor are as follows;

- Realize software SAR function by using 32bit RISC controller (76MIPS @66MHz)
- Firmware is downloaded from external memory to Instruction Cache
- Supports 64VCs
- Supports UTOPIA level 2 (including management interface) as PHY layer interface
- Supports processing AAL5, Raw cell(AAL0) and F4,F5 OAM cells
- Supports 4 service classes (CBR,VBR,ABR,UBR)
- Supports dynamic mapping between IP-flow and ATM VC
- Supports up to 50Mbps Cell speed together with upstream and downstream
- Supports fine grain ATM cell shaping in 1cell/sec granularity on per VC basis

**Figure 1-6. Block Diagram of ATM Cell Processor**



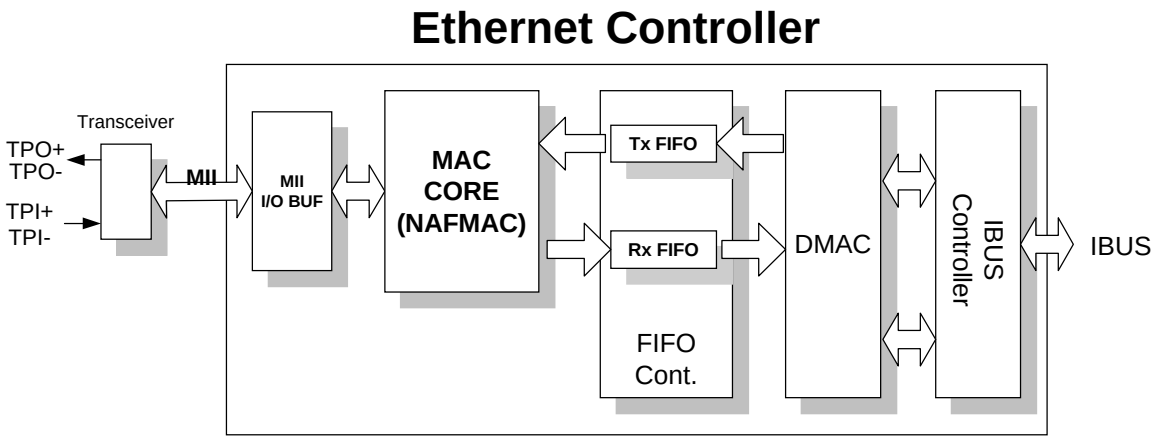
### 1.5.5 Ethernet controller

Ethernet Controller supports 2-channel 10Mbps/100Mbps Ethernet MAC (Media Access Control) function and MII (Media Independent Interface) function

Features of Ethernet Controller are as follows;

- Supports 10M/100M Ethernet MAC function compliant to IEEE802.3 and IEEE802.3u
- Supports 3.3V MII compliant to IEEE802.3u
- Supports full duplex operation for both 100Mbps and 10Mbps
- Supports flow control function compliant to IEEE802.3x/D3.2
- Implements 256Byte FIFO buffer for each TX and RX
- Implements address filtering functions for unicast/multicast/broadcast
- Implements MIB counters for network management (MIB II, Ether-like MIB,IEEE802.3LME are supported)
- Supports loopback function within MII interface
- Implements local DMA controller with individual DMA channels for each TX and RX

**Figure 1-7. Block Diagram of Ethernet Controller**

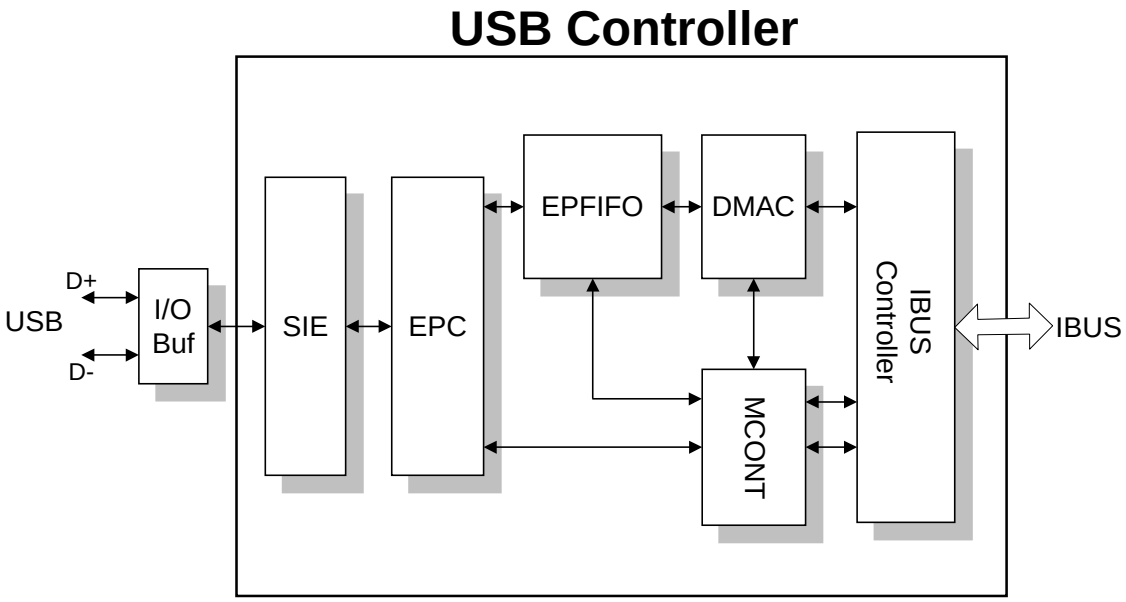


### 1.5.6 USB controller

USB Controller provides Full Speed Function device function defined in Universal Serial Bus.
Features of USB Controller are as follows;

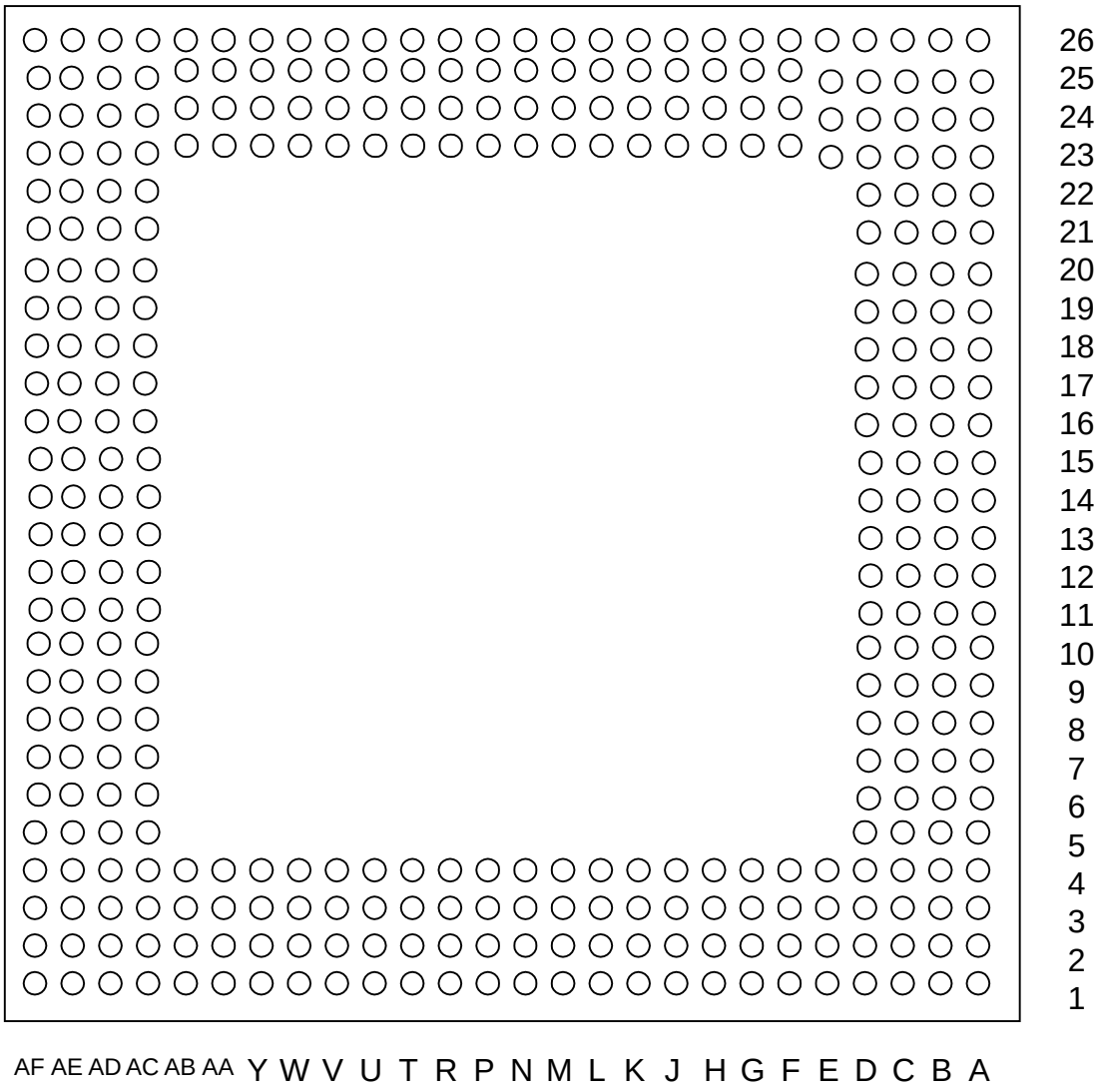
- Compliant to Universal Serial Bus Specification Rev1.1
- Supports xDSL Sub Class function compliant to Communication Class Definition
- Supports Device class function by software running on VR4120A core
- Performs 12Mbps Full Speed USB function device (Hub function will be not supported)
- Can handle Suspend, Resume and Wake-up management signaling
- Supports Remote Wake-up.
- Implements 7 kinds of endpoints (Control, Interrupt IN/OUT, Isochronous IN/OUT, Bulk IN/OUT)
- Implements 64Bytes FIFO buffer used for Control transfer for TX
- Implements 128Bytes FIFO buffer used for Isochronous transfer for TX
- Implements 128Bytes FIFO buffer used for Bulk transfer for TX
- Implements 64Bytes FIFO buffer used for Interrupt transfer for TX
- Implements 128Bytes shared FIFO buffer used for Control/Isochronous/Bulk/Interrupt transfer for RX
- Implements local DMAC(DMA controller) block
- Can directly connect USB connector through USB dedicated I/O buffer

**Figure 1-8. Block Diagram of USB Controller**



## 1.6 Pin Configuration (Bottom View)

- 352-pin Tape BGA (35 × 35)
  $\mu$PD98501N7-F6



## Pin Name

| Pin No. | Pin Name | Pin No. | Pin Name | Pin No. | Pin Name | Pin No. | Pin Name | Pin No. | Pin Name |
|---|---|---|---|---|---|---|---|---|---|
| A01 | Open | C01 | SCLK | E01 | EVDD | L23 | UMD0 | V01 | SMA6 |
| A02 | IVDD | C02 | CLKSL | E02 | PSDVD | L24 | IC-PUp | V02 | SMA5 |
| A03 | GND | C03 | IC-PDn | E03 | PSAGND | L25 | GND* | V03 | IVDD |
| A04 | PUAGND | C04 | PUMD | E04 | GND | L26 | IVDD | V04 | GND* |
| A05 | GND | C05 | PUAVD | E23 | IVDD | M01 | SMD16 | V23 | MI2RD1 |
| A06 | EVDD* | C06 | IC-PDn | E24 | UMAD4 | M02 | SMD17 | V24 | MI2RD0 |
| A07 | EVDD | C07 | IC-PDn | E25 | UMAD2 | M03 | IVDD | V25 | MI2MD |
| A08 | IC-PUpR | C08 | USBDP | E26 | EVDD | M04 | GND* | V26 | MI2RDV |
| A09 | Open | C09 | IC-PDnR | F01 | SRMCS_B | M23 | IC-PUpR | W01 | SMA4 |
| A10 | IVDD | C10 | IVDD* | F02 | SRMOE_B | M24 | IC-PUpR | W02 | SMA3 |
| A11 | IVDD* | C11 | UDRCLV | F03 | PSTBY | M25 | IC-PUpR | W03 | GND |
| A12 | EVDD | C12 | UDRD6 | F04 | PSMD | M26 | IC-PUpR | W04 | SMA2 |
| A13 | UDRD1 | C13 | UDRD3 | F23 | UMAD1 | N01 | SMA19 | W23 | IVDD* |
| A14 | IVDD | C14 | UDRD0 | F24 | UMAD0 | N02 | SMA20 | W24 | MI2MCLK |
| A15 | UDRAD3 | C15 | UDRCLK | F25 | IC-PUpR | N03 | GND* | W25 | MI2RD3 |
| A16 | UDRAD0 | C16 | GND | F26 | IC-PUpR | N04 | EVDD | W26 | MI2RD2 |
| A17 | UDTE_B | C17 | UDTAD4 | G01 | SMD30 | N23 | IC-PUpR | Y01 | SMA1 |
| A18 | UDTAD3 | C18 | UDTAD1 | G02 | SMD31 | N24 | IC-PUpR | Y02 | EVDD |
| A19 | GND* | C19 | UDTD7 | G03 | IVDD* | N25 | IC-PUpR | Y03 | SMA0 |
| A20 | UDTD5 | C20 | UDTD4 | G04 | GND | N26 | IC-PUpR | Y04 | SDCKE1 |
| A21 | UDTCLK | C21 | IVDD* | G23 | IC-PUpR | P01 | SMA18 | Y23 | GND |
| A22 | UMRST_B | C22 | UMRDY_B | G24 | IC-PUpR | P02 | SMA17 | Y24 | IC-PUpR |
| A23 | UDTD0 | C23 | UMRD_B | G25 | IC-PUpR | P03 | SMA16 | Y25 | IC-PUpR |
| A24 | UMINT_B | C24 | EVDD | G26 | IC-PUpR | P04 | SMA15 | Y26 | GND |
| A25 | UMAD11 | C25 | UMAD7 | H01 | SMD27 | P23 | MI2TD1 | AA01 | IVDD |
| A26 | UMMD | C26 | GND | H02 | GND | P24 | MI2TD0 | AA02 | GND* |
| B01 | IC-PDn | D01 | PSAVD | H03 | SMD28 | P25 | IVDD | AA03 | SDCLK1 |
| B02 | Open | D02 | PSDGND | H04 | SMD29 | P26 | GND | AA04 | SDCS_B |
| B03 | Open | D03 | GND* | H23 | IC-PUpR | R01 | SMA14 | AA23 | EVDD |
| B04 | PUDVD | D04 | IC-PDn | H24 | IC-PUpR | R02 | EVDD | AA24 | MITD1 |
| B05 | PUDGND | D05 | USBCLK | H25 | UMD7 | R03 | SMA13 | AA25 | MITD0 |
| B06 | PUSTBY | D06 | IC-PDn | H26 | GND | R04 | SMA12 | AA26 | IVDD |
| B07 | GND* | D07 | Open | J01 | IVDD | R23 | MI2TCLK | AB01 | SDRAS_B |
| B08 | EVDD | D08 | USBDM | J02 | GND* | R24 | MI2COL | AB02 | SDCAS_B |
| B09 | Open | D09 | IC-PDnR | J03 | SMD25 | R25 | MI2TD3 | AB03 | EVDD |
| B10 | GND* | D10 | GND | J04 | SMD26 | R26 | MI2TD2 | AB04 | SDCLK0 |
| B11 | UDRSC | D11 | UDRE_B | J23 | IVDD | T01 | SMA11 | AB23 | GND |
| B12 | UDRD5 | D12 | UDRD7 | J24 | UMD6 | T02 | IVDD* | AB24 | MICRS |
| B13 | UDRD2 | D13 | UDRD4 | J25 | UMD5 | T03 | GND | AB25 | MITD3 |
| B14 | GND* | D14 | UDRAD4 | J26 | UMD4 | T04 | SMA10 | AB26 | MITD2 |
| B15 | UDRAD2 | D15 | UDRAD1 | K01 | SMD22 | T23 | MI2TER | AC01 | SDWE_B |
| B16 | IVDD | D16 | UDTCLV | K02 | SMD23 | T24 | MI2CRS | AC02 | SDCKE0 |
| B17 | UDTSC | D17 | EVDD* | K03 | EVDD* | T25 | IVDD* | AC03 | SMD15 |
| B18 | UDTAD2 | D18 | IVDD | K04 | SMD24 | T26 | GND | AC04 | SMD10 |
| B19 | UDTAD0 | D19 | UDTD6 | K23 | UMD3 | U01 | SMA9 | AC05 | SMD6 |
| B20 | EVDD | D20 | UDTD3 | K24 | EVDD | U02 | SMA8 | AC06 | EVDD |
| B21 | UDTD2 | D21 | GND | K25 | UMD2 | U03 | SMA7 | AC07 | SMD1 |
| B22 | UDTD1 | D22 | GND | K26 | UMD1 | U04 | EVDD | AC08 | EXNM_BI |
| B23 | UMSL_B | D23 | UMAD10 | L01 | SMD18 | U23 | MI2TE | AC09 | POM5 |
| B24 | UMWR_B | D24 | UMAD6 | L02 | SMD19 | U24 | MI2RCLK | AC10 | POM2 |
| B25 | UMAD9 | D25 | UMAD5 | L03 | SMD20 | U25 | EVDD | AC11 | POM0 |
| B26 | UMAD8 | D26 | UMAD3 | L04 | SMD21 | U26 | MI2RER | AC12 | URSDI |

| Pin No. | Pin Name | Pin No. | Pin Name |
|---|---|---|---|
| AC13 | EVDD | AE13 | URDTR_B |
| AC14 | NTRCD3 | AE14 | GND |
| AC15 | EVDD | AE15 | NTRCD1 |
| AC16 | GND | AE16 | NBKTIO |
| AC17 | NRMJTDI | AE17 | NJTMS |
| AC18 | JRSTB_B | AE18 | BIG |
| AC19 | JDO | AE19 | IVDD* |
| AC20 | IC-PDn | AE20 | JCK |
| AC21 | ROMSEL0 | AE21 | IC-PDn |
| AC22 | MIRD2 | AE22 | IC-PDn |
| AC23 | GND | AE23 | MIRD3 |
| AC24 | MITER | AE24 | IVDD |
| AC25 | MITCLK | AE25 | MIRCLK |
| AC26 | MICOL | AE26 | MITE |
| AD01 | GND | AF01 | GND |
| AD02 | SMD11 | AF02 | SMD13 |
| AD03 | SMD14 | AF03 | SMD12 |
| AD04 | SMD8 | AF04 | SMD5 |
| AD05 | GND* | AF05 | SMD7 |
| AD06 | SMD4 | AF06 | SMD0 |
| AD07 | IVDD* | AF07 | RST_B |
| AD08 | EXINT_B | AF08 | EVDD |
| AD09 | POM7 | AF09 | POM4 |
| AD10 | IVDD* | AF10 | POM1 |
| AD11 | URCLK | AF11 | IVDD |
| AD12 | URDSR_B/MWDO | AF12 | URCTS_B/MWSK |
| AD13 | URRTS_B/MWDI | AF13 | URSDO |
| AD14 | IVDD | AF14 | IC-PDn |
| AD15 | NTRCD0 | AF15 | NTRCD2 |
| AD16 | NJTRST_B | AF16 | NJTDO |
| AD17 | NTCED | AF17 | IVDD* |
| AD18 | NTRCLK | AF18 | NJTCK |
| AD19 | JMS | AF19 | GND |
| AD20 | EVDD | AF20 | JDI |
| AD21 | ROMSEL1 | AF21 | SSEL |
| AD22 | MIMCLK | AF22 | IC-PDn |
| AD23 | MIRD0 | AF23 | GND |
| AD24 | MIMD | AF24 | MIRD1 |
| AD25 | MIRER | AF25 | GND* |
| AD26 | IVDD* | AF26 | MIRDV |
| AE01 | IVDD* | | |
| AE02 | GND | | |
| AE03 | EVDD | | |
| AE04 | SMD9 | | |
| AE05 | SMD3 | | |
| AE06 | SMD2 | | |
| AE07 | GND | | |
| AE08 | ENDCEN | | |
| AE09 | POM6 | | |
| AE10 | POM3 | | |
| AE11 | GND* | | |
| AE12 | URDCD_B/MWCS | | |

**Special pin name description:**

IC-PDn: Pull Down

IC-PDnR: Pull Down with Resistor

IC-PUp: Pull Up

IC-PUpR: Pull Up with Resistor

## 1.7 Pin Function

Symbol of I/O column indicates following status in this section.

I :Input
O :Output
I/O :Bidirection
I/OZ :Bidirection (Include Hi-Z state)
I/OD :Bidirection (Open drain output)
OZ :Output (Include Hi-Z state)
OD :Output (Open drain)

### 1.7.1 System PLL power supply

| Pin Name | Pin No. | I/O | Active Level | Function |
|---|---|---|---|---|
| PSAGND | E03 | | | Analog ground |
| PSAVD | D01 | | | Analog power supply |
| PSDGND | D02 | | | Digital ground |
| PSDVD | E02 | | | Digital power supply |

### 1.7.2 USB PLL power supply

| Pin Name | Pin No. | I/O | Active Level | Function |
|---|---|---|---|---|
| PUAGND | A04 | | | Analog ground |
| PUAVD | C05 | | | Analog power supply |
| PUDGND | B05 | | | Digital ground |
| PUDVD | B04 | | | Digital power supply |

### 1.7.3 System control interface

| Pin Name | Pin No. | I/O | Active Level | Function |
|---|---|---|---|---|
| SCLK | C01 | I | | System clock (33MHz) |
| CLKSL | C02 | I | | Clock select (100MHz/66MHz) |
| PSMD | F04 | I | | System PLL mode control input |
| PSTBY | F03 | I | | System PLL standby mode control input |
| PUMD | C04 | I | | USB PLL Mode control |
| PUSTBY | B06 | I | | USB PLL Standby mode control |
| BIG | AE18 | I | H | VR4120A Bigendian mode |
| ENDCEN | AE08 | I | | Big Endian mode enable |
| EXINT_B | AD08 | I | L | External interrupt |
| EXNMI_B | AC08 | I | L | External non-maskable interrupt |
| RSTB_B | AF07 | I | L | System reset |
| ROMSEL0 | AC21 | I | | ROM access bus width |
| ROMSEL1 | AD21 | I | | (ROMSEL1/0=L/L;32bit, L/H;16bit, H/L;8bit) |
| SSEL | AF21 | I | | UART/Micro Wire Select (L; UART, H; Micro Wire) |

### 1.7.4 Memory interface

(1/2)

| Pin Name | Pin No. | I/O | Active Level | Function |
|---|---|---|---|---|
| SDCLK0 | AB04 | O | | SDRAM Clock |
| SDCLK1 | AA03 | O | | SDRAM Clock |
| SDCKE0 | AC02 | O | H | SDRAM Clock Enable |
| SDCKE1 | Y04 | O | H | SDRAM Clock Enable |
| SDCS_B | AA04 | O | L | SDRAM Chip select |
| SDRAS_B | AB01 | O | L | SDRAM Row address strobe |
| SDCAS_B | AB02 | O | L | SDRAM Column address strobe |
| SDWE_B | AC01 | O | L | SDRAM/PROM/FLASH write enable |
| SRMCS_B | F01 | O | L | PROM/FLASH chip select |
| SRMOE_B | F02 | O | L | PROM/FLASH output enable |
| SMA0 | Y03 | O | | Memory Address |
| SMA1 | Y01 | O | | Memory Address |
| SMA2 | W04 | O | | Memory Address |
| SMA3 | W02 | O | | Memory Address |
| SMA4 | W01 | O | | Memory Address |
| SMA5 | V02 | O | | Memory Address |
| SMA6 | V01 | O | | Memory Address |
| SMA7 | U03 | O | | Memory Address |
| SMA8 | U02 | O | | Memory Address |
| SMA9 | U01 | O | | Memory Address |
| SMA10 | T04 | O | | Memory Address |
| SMA11 | T01 | O | | Memory Address |
| SMA12 | R04 | O | | Memory Address |
| SMA13 | R03 | O | | Memory Address |
| SMA14 | R01 | O | | Memory Address |
| SMA15 | P04 | O | | Memory Address |
| SMA16 | P03 | O | | Memory Address |
| SMA17 | P02 | O | | Memory Address |
| SMA18 | P01 | O | | Memory Address |
| SMA19 | N01 | O | | Memory Address |
| SMA20 | N02 | O | | Memory Address |
| SMD0 | AF06 | I/O | | Memory data |
| SMD1 | AC07 | I/O | | Memory data |
| SMD2 | AE06 | I/O | | Memory data |
| SMD3 | AE05 | I/O | | Memory data |
| SMD4 | AD06 | I/O | | Memory data |
| SMD5 | AF04 | I/O | | Memory data |

(2/2)

| Pin Name | Pin No. | I/O | Active Level | Function |
|---|---|---|---|---|
| SMD6 | AC05 | I/O | | Memory data |
| SMD7 | AF05 | I/O | | Memory data |
| SMD8 | AD04 | I/O | | Memory data |
| SMD9 | AE04 | I/O | | Memory data |
| SMD10 | AC04 | I/O | | Memory data |
| SMD11 | AD02 | I/O | | Memory data |
| SMD12 | AF03 | I/O | | Memory data |
| SMD13 | AF02 | I/O | | Memory data |
| SMD14 | AD03 | I/O | | Memory data |
| SMD15 | AC03 | I/O | | Memory data |
| SMD16 | M01 | I/O | | Memory data |
| SMD17 | M02 | I/O | | Memory data |
| SMD18 | L01 | I/O | | Memory data |
| SMD19 | L02 | I/O | | Memory data |
| SMD20 | L03 | I/O | | Memory data |
| SMD21 | L04 | I/O | | Memory data |
| SMD22 | K01 | I/O | | Memory data |
| SMD23 | K02 | I/O | | Memory data |
| SMD24 | K04 | I/O | | Memory data |
| SMD25 | J03 | I/O | | Memory data |
| SMD26 | J04 | I/O | | Memory data |
| SMD27 | H01 | I/O | | Memory data |
| SMD28 | H03 | I/O | | Memory data |
| SMD29 | H04 | I/O | | Memory data |
| SMD30 | G01 | I/O | | Memory data |
| SMD31 | G02 | I/O | | Memory data |

### 1.7.5 ATM interface

#### 1.7.5.1 UTOPIA management interface

| Pin Name | Pin No. | I/O | Active Level | Function |
|---|---|---|---|---|
| UMMD | A26 | O | | Management mode select |
| UMINT_B | A24 | I | L | Interrupt from PHY |
| UMRD_B | C23 | O | L | Management read enable |
| UMRDY_B | C22 | I | L | Management data ready |
| UMRST_B | A22 | O | L | PHY reset |
| UMSL_B | B23 | O | L | PHY select |
| UMWR_B | B24 | O | L | Management write enable |
| UMAD0 | F24 | O | | PHY address |
| UMAD1 | F23 | O | | PHY address |
| UMAD2 | E25 | O | | PHY address |
| UMAD3 | D26 | O | | PHY address |
| UMAD4 | E24 | O | | PHY address |
| UMAD5 | D25 | O | | PHY address |
| UMAD6 | D24 | O | | PHY address |
| UMAD7 | C25 | O | | PHY address |
| UMAD8 | B26 | O | | PHY address |
| UMAD9 | B25 | O | | PHY address |
| UMAD10 | D23 | O | | PHY address |
| UMAD11 | A25 | O | | PHY address |
| UMD0 | L23 | I/O | | Management data |
| UMD1 | K26 | I/O | | Management data |
| UMD2 | K25 | I/O | | Management data |
| UMD3 | K23 | I/O | | Management data |
| UMD4 | J26 | I/O | | Management data |
| UMD5 | J25 | I/O | | Management data |
| UMD6 | J24 | I/O | | Management data |
| UMD7 | H25 | I/O | | Management data |

### 1.7.5.2 UTOPIA data interface

| Pin Name | Pin No. | I/O | Active Level | Function |
|---|---|---|---|---|
| UDRCLK | C15 | O | | Receive clock |
| UDRCLV | C11 | I | H | Receive cell available |
| UDRE_B | D11 | O | L | Receive enable |
| UDRSC | B11 | I | H | Receive cell start |
| UDRAD0 | A16 | O | | Receive PHY Address |
| UDRAD1 | D15 | O | | Receive PHY Address |
| UDRAD2 | B15 | O | | Receive PHY Address |
| UDRAD3 | A15 | O | | Receive PHY Address |
| UDRAD4 | D14 | O | | Receive PHY Address |
| UDRD0 | C14 | I | | Receive data |
| UDRD1 | A13 | I | | Receive data |
| UDRD2 | B13 | I | | Receive data |
| UDRD3 | C13 | I | | Receive data |
| UDRD4 | D13 | I | | Receive data |
| UDRD5 | B12 | I | | Receive data |
| UDRD6 | C12 | I | | Receive data |
| UDRD7 | D12 | I | | Receive data |
| UDTCLK | A21 | O | | Transmit clock |
| UDTCLV | D16 | I | H | Transmit Cell Available |
| UDTE_B | A17 | O | L | Transmit enable |
| UDTSC | B17 | O | H | Transmit Cell start position |
| UDTAD0 | B19 | O | | Transmit PHY Address |
| UDTAD1 | C18 | O | | Transmit PHY Address |
| UDTAD2 | B18 | O | | Transmit PHY Address |
| UDTAD3 | A18 | O | | Transmit PHY Address |
| UDTAD4 | C17 | O | | Transmit PHY Address |
| UDTD0 | A23 | O | | Transmit data |
| UDTD1 | B22 | O | | Transmit data |
| UDTD2 | B21 | O | | Transmit data |
| UDTD3 | D20 | O | | Transmit data |
| UDTD4 | C20 | O | | Transmit data |
| UDTD5 | A20 | O | | Transmit data |
| UDTD6 | D19 | O | | Transmit data |
| UDTD7 | C19 | O | | Transmit data |

### 1.7.6 Ethernet interface

#### 1.7.6.1 Ethernet interface (Channel 1)

| Pin Name | Pin No. | I/O | Active Level | Function |
|---|---|---|---|---|
| MIRCLK | AE25 | I | | MII - Receive clock (25MHz) |
| MIMCLK | AD22 | O | | MII - management clock |
| MIMD | AD24 | I/O | | MII - management |
| MICOL | AC26 | I | | MII - Collision |
| MICRS | AB24 | I | | MII - carrier Sense |
| MIRDV | AF26 | I | | MII - Receive data valid |
| MIRER | AD25 | I | | MII - Receive error |
| MIRD0 | AD23 | I | | MII - Receive data |
| MIRD1 | AF24 | I | | MII - Receive data |
| MIRD2 | AC22 | I | | MII - Receive data |
| MIRD3 | AE23 | I | | MII - Receive data |
| MITCLK | AC25 | I | | MII - Transmit clock (25MHz) |
| MITE | AE26 | O | | MII - Transmit enable |
| MITER | AC24 | O | | MII - Transmit error |
| MITD0 | AA25 | O | | MII - Transmit data |
| MITD1 | AA24 | O | | MII - Transmit data |
| MITD2 | AB26 | O | | MII - Transmit data |
| MITD3 | AB25 | O | | MII - Transmit data |

### 1.7.6.2 Ethernet interface (Channel 2)

| Pin Name | Pin No. | I/O | Active Level | Function |
|---|---|---|---|---|
| MI2RCLK | U24 | I | | MII - Receive clock (25MHz) |
| MI2MCLK | W24 | O | | MII - management clock |
| MI2MD | V25 | I/O | | MII - management |
| MI2COL | R24 | I | | MII - Collision |
| MI2CRS | T24 | I | | MII - carrier Sense |
| MI2RDV | V26 | I | | MII - Receive data valid |
| MI2RER | U26 | I | | MII - Receive error |
| MI2RD0 | V24 | I | | MII - Receive data |
| MI2RD1 | V23 | I | | MII - Receive data |
| MI2RD2 | W26 | I | | MII - Receive data |
| MI2RD3 | W25 | I | | MII - Receive data |
| MI2TCLK | R23 | I | | MII - Transmit clock (25MHz) |
| MI2TE | U23 | O | | MII - Transmit enable |
| MI2TER | T23 | O | | MII - Transmit error |
| MI2TD0 | P24 | O | | MII - Transmit data |
| MI2TD1 | P23 | O | | MII - Transmit data |
| MI2TD2 | R26 | O | | MII - Transmit data |
| MI2TD3 | R25 | O | | MII - Transmit data |

### 1.7.7 USB interface

| Pin Name | Pin No. | I/O | Active Level | Function |
|---|---|---|---|---|
| USBCLK | D05 | I | | External USB Clock |
| USBDM | D08 | I/O | | USB Data (-) |
| USBDP | C08 | I/O | | USB Data (+) |

### 1.7.8 UART/Micro Wire interface

| Pin Name | Pin No. | I/O | Active Level | Function |
|---|---|---|---|---|
| URCLK | AD11 | I | | UART external Clock |
| URSDO | AF13 | O | | UART serial data output |
| URSDI | AC12 | I | | UART serial data input |
| URDTR_B | AE13 | O | L | UART data terminal ready |
| URRTS_B | AD13 | O | L | UART data request to send |
| /MWDI | | I | | Micro Wire data in |
| URCTS_B | AF12 | I | L | UART clear to send |
| /MWSK | | O | | Micro Wire SK |
| URDCD_B | AE12 | I | L | UART data carrier detect |
| /MWCS | | O | | Micro Wire chip select |
| URDSR_B | AD12 | I | L | UART data set ready |
| /MWDO | | O | | Micro Wire data out |

**Remark** For the function multiplexed pins (AD13, AF12, AE12, AD12), function is determined as follows.

SSEL = L: UART operation mode

SSEL = H: Micro Wire operation mode

### 1.7.9 Parallel port interface

| Pin Name | Pin No. | I/O | Active Level | Function |
|---|---|---|---|---|
| POM0 | AC11 | O | | Parallel Port Signal Output |
| POM1 | AF10 | O | | Parallel Port Signal Output |
| POM2 | AC10 | O | | Parallel Port Signal Output |
| POM3 | AE10 | O | | Parallel Port Signal Output |
| POM4 | AF09 | O | | Parallel Port Signal Output |
| POM5 | AC09 | O | | Parallel Port Signal Output |
| POM6 | AE09 | O | | Parallel Port Signal Output |
| POM7 | AD09 | O | | Parallel Port Signal Output |

### 1.7.10 Boundary scan interface

| Pin Name | Pin No. | I/O | Active Level | Function |
|---|---|---|---|---|
| JCK | AE20 | I | | B-SCAN clock |
| JDI | AF20 | I | | B-SCAN input-data |
| JDO | AC19 | OZ | | B-SCAN output-data |
| JMS | AD19 | I | | B-SCAN mode select |
| JRSTB_B | AC18 | I | L | B-SCAN reset |

### 1.7.11 N-Wire interface

| Pin Name | Pin No. | I/O | Active Level | Function |
|---|---|---|---|---|
| NJTMS | AE17 | I | | N-Wire Mode select |
| NTRCLK | AD18 | O | | N-Wire clock for data trace |
| NJTCK | AF18 | I | | N-Wire Clock |
| NTCED | AD17 | O | | N-Wire trace end output |
| NBKTIO | AE16 | I/O | | N-Wire event trigger output |
| NRMJTDI | AC17 | I | | N-Wire data input/Rmode |
| NJTDO | AF16 | O | | N-Wire data output |
| NTRCD0 | AD15 | O | | N-Wire Trace data output |
| NTRCD1 | AE15 | O | | N-Wire Trace data output |
| NTRCD2 | AF15 | O | | N-Wire Trace data output |
| NTRCD3 | AC14 | O | | N-Wire Trace data output |
| NJTRST_B | AD16 | I | L | N-Wire reset |

## 1.8 I/O Register Map

| Core | Offset | Register Length (Byte) | Name | Access by VR4120A | Description |
|---|---|---|---|---|---|
| ATM | F000H | 4 | A_GMR | R/W | General Mode Register |
| ATM | F004H | 4 | A_GSR | R | General Status Register |
| ATM | F008H | 4 | A_IMR | R/W | Interrupt Mask Register |
| ATM | F00CH | 4 | A_RQU | R | Receive Queue Underrunning |
| ATM | F010H | 4 | A_RQA | R | Receive Queue Alert |
| ATM | F014H | - | N/A | - | Reserved for future use |
| ATM | F018H | 4 | A_VER | R | Version Number |
| ATM | F01CH | - | N/A | - | Reserved for future use |
| ATM | F020H | 4 | A_CMR | R/W | Command Register |
| ATM | F024H | - | N/A | - | Reserved for future use |
| ATM | F028H | 4 | A_CER | R/W | Command Extension Register |
| ATM | F02CH-F04CH | - | N/A | - | Reserved for future use |
| ATM | F050H | 4 | A_MSA0 | R/W | Mailbox0 Start Address |
| ATM | F054H | 4 | A_MSA1 | R/W | Mailbox1 Start Address |
| ATM | F058H | 4 | A_MSA2 | R/W | Mailbox2 Start Address |
| ATM | F05CH | 4 | A_MSA3 | R/W | Mailbox3 Start Address |
| ATM | F060H | 4 | A_MBA0 | R/W | Mailbox0 Bottom Address |
| ATM | F064H | 4 | A_MBA1 | R/W | Mailbox1 Bottom Address |
| ATM | F068H | 4 | A_MBA2 | R/W | Mailbox2 Bottom Address |
| ATM | F06CH | 4 | A_MBA3 | R/W | Mailbox3 Bottom Address |
| ATM | F070H | 4 | A_MTA0 | R/W | Mailbox0 Tail Address |
| ATM | F074H | 4 | A_MTA1 | R/W | Mailbox1 Tail Address |
| ATM | F078H | 4 | A_MTA2 | R/W | Mailbox2 Tail Address |
| ATM | F07CH | 4 | A_MTA3 | R/W | Mailbox3 Tail Address |
| ATM | F080H | 4 | A_MWA0 | R/W | Mailbox0 Write Address |
| ATM | F084H | 4 | A_MWA1 | R/W | Mailbox1 Write Address |
| ATM | F088H | 4 | A_MWA2 | R/W | Mailbox2 Write Address |
| ATM | F08CH | 4 | A_MWA3 | R/W | Mailbox3 Write Address |
| ATM | F090H | 4 | A_RCC | R | Valid Receiving Cell Counter |
| ATM | F094H | 4 | A_TCC | R | Valid Transmitting Cell Counter |
| ATM | F098H | 4 | A_RUEC | R | Receive Unprovisioned VPI/VCI Error Cell Counter |
| ATM | F09CH | 4 | A_RIDC | R | Receiving Internal Discarded Cell Counter |
| ATM | F0A0H-F0AFH | - | N/A | - | Reserved for future use |
| ATM | F0B0H-F0B3H | 4 | A_APR | R/W | ABR Parameter Register |
| ATM | F0B4H-F0BCH | - | N/A | - | Reserved for future use |
| ATM | F0C0H | 4 | A_T1R | R/W | T1 Timer Register |
| ATM | F0C4H | - | N/A | - | Reserved for future use |
| ATM | F0C8H | 4 | A_TSR | R/W | Time Stamp Register |
| ATM | F200H-F2FFH | - | N/A | - | Can not access from VR4120A RISC Core. |
| ATM | F300H | 4 | A_IBBAR | R/W | IBUS Base Address Register |
| ATM | F304H | 4 | A_INBAR | R/W | Instruction Base Address Register |
| ATM | F308H- F31FH | - | N/A | - | Reserved for future use |
| ATM | F320H | 4 | A_UMCMD | R/W | UTOPIA Management Interface Command Register |
| ATM | F324H- F3FFH | - | N/A | - | Reserved for future use |
| ATM | F400H-F4FFH | - | N/A | - | Can not access from VR4120A RISC Core. |
| ATM | F500H-FFFFH | - | N/A | - | Reserved for future use |
| Ether | 00H | 4 | En_MACC1 | R/W | MAC configuration register 1 |
| Ether | 04H | 4 | En_MACC2 | R/W | MAC configuration register 2 |
| Ether | 08H | 4 | En_IPGT | R/W | Back-to-Back IPG register |
| Ether | 0CH | 4 | En_IPGR | R/W | Non Back-to-Back IPG register |
| Ether | 10H | 4 | En_CLRT | R/W | Collision register |
| Ether | 14H | 4 | En_LMAX | R/W | Max packet length register |
| Ether | 18H-1CH | - | N/A | - | Reserved for future use |
| Ether | 20H | 4 | En_RETX | R/W | Retry count register |
| Ether | 24H-50H | - | N/A | - | Reserved for future use |
| Ether | 54H | 4 | En_LSA2 | R/W | Station Address register 2 |
| Ether | 58H | 4 | En_LSA1 | R/W | Station Address register 1 |
| Ether | 5CH | 4 | En_PTVR | R | Pause timer value read register |
| Ether | 60H | - | N/A | - | Reserved for future use |
| Ether | 64H | 4 | En_VLTP | R/W | VLAN type register |
| Ether | 80H | 4 | En_MIIC | R/W | MII configuration register |
| Ether | 84H-90H | - | N/A | - | Reserved for future use |
| Ether | 94H | 4 | En_MCMD | W | MII command register |
| Ether | 98H | 4 | En_MADR | R/W | MII address register |
| Ether | 9CH | 4 | En_MWTD | R/W | MII write data register |
| Ether | A0H | 4 | En_MRDD | R | MII read data register |
| Ether | A4H | 4 | En_MIND | R | MII indicator register |
| Ether | A8H-C4H | - | N/A | - | Reserved for future use |
| Ether | CCH | 4 | En_HT1 | R/W | Hash table register 1 |
| Ether | D0H | 4 | En_HT2 | R/W | Hash table register 2 |
| Ether | D4H-D8H | - | N/A | - | Reserved for future use |

| Core | Offset | Register Length (Byte) | Name | Access by VR4120A | Description |
|---|---|---|---|---|---|
| Ether | DCH | 4 | En_CAR1 | R/W | Carry register 1 |
| Ether | E0H | 4 | En_CAR2 | R/W | Carry register 2 |
| Ether | E4H-12CH | - | N/A | - | Reserved for future use |
| Ether | 130H | 4 | En_CAM1 | R/W | Carry mask register 1 |
| Ether | 134H | 4 | En_CAM2 | R/W | Carry mask register 2 |
| Ether | 138H-13CH | - | N/A | - | Reserved for future use |
| Ether | 140H | 4 | En_RBYT | R/W | Receive Byte Counter |
| Ether | 144H | 4 | En_RPKT | R/W | Receive Packet Counter |
| Ether | 148H | 4 | En_RFCS | R/W | Receive FCS Error Counter |
| Ether | 14CH | 4 | En_RMCA | R/W | Receive Multicast Packet Counter |
| Ether | 150H | 4 | En_RBCA | R/W | Receive Broadcast Packet Counter |
| Ether | 154H | 4 | En_RXCF | R/W | Receive Control Frame Packet Counter |
| Ether | 158H | 4 | En_RXPF | R/W | Receive PAUSE Frame Packet Counter |
| Ether | 15CH | 4 | En_RXUO | R/W | Receive Unknown OP code Counter |
| Ether | 160H | 4 | En_RALN | R/W | Receive Alignment Error Counter |
| Ether | 164H | 4 | En_RFLR | R/W | Receive Frame Length Out of Range Counter |
| Ether | 168H | 4 | En_RCDE | R/W | Receive Code Error Counter |
| Ether | 16CH | 4 | En_RFCR | R/W | Receive False Carrier Counter |
| Ether | 170H | 4 | En_RUND | R/W | Receive Undersize Packet Counter |
| Ether | 174H | 4 | En_ROVR | R/W | Receive Oversize Packet Counter |
| Ether | 178H | 4 | En_RFRG | R/W | Receive Error Undersize Packet Counter |
| Ether | 17CH | 4 | En_RJBR | R/W | Receive Error Oversize Packet Counter |
| Ether | 180H | 4 | En_R64 | R/W | Receive 64 Byte Frame Counter |
| Ether | 184H | 4 | En_R127 | R/W | Receive 65 to 127 Byte Frame Counter |
| Ether | 188H | 4 | En_R255 | R/W | Receive 128 to 255 Byte Frame Counter |
| Ether | 18CH | 4 | En_R511 | R/W | Receive 256 to 511 Byte Frame Counter |
| Ether | 190H | 4 | En_R1K | R/W | Receive 512 to 1023 Byte Frame Counter |
| Ether | 194H | 4 | En_RMAX | R/W | Receive Over 1023 Byte Frame Counter |
| Ether | 198H | 4 | En_RVBT | R/W | Receive Valid Byte Counter |
| Ether | 1C0H | 4 | En_TBYT | R/W | Transmit Byte Counter |
| Ether | 1C4H | 4 | En_TPCT | R/W | Transmit Packet Counter |
| Ether | 1C8H | 4 | En_TFCS | R/W | Transmit CRC Error Packet Counter |
| Ether | 1CCH | 4 | En_TMCA | R/W | Transmit Multicast Packet Counter |
| Ether | 1D0H | 4 | En_TBCA | R/W | Transmit Broadcast Packet Counter |
| Ether | 1D4H | 4 | En_TUCA | R/W | Transmit Unicast Packet Counter |
| Ether | 1D8H | 4 | En_TXPF | R/W | Transmit PAUSE control Frame Counter |
| Ether | 1DCH | 4 | En_TDFR | R/W | Transmit Single Deferral Packet Counter |
| Ether | 1E0H | 4 | En_TXDF | R/W | Transmit Excessive Deferral Packet Counter |
| Ether | 1E4H | 4 | En_TSCL | R/W | Transmit Single Collision Packet Counter |
| Ether | 1E8H | 4 | En_TMCL | R/W | Transmit Multiple collision Packet Counter |
| Ether | 1ECH | 4 | En_TLCL | R/W | Transmit Late Collision Packet Counter |
| Ether | 1F0H | 4 | En_TXCL | R/W | Transmit Excessive Collision Packet Counter |
| Ether | 1F4H | 4 | En_TNCL | R/W | Transmit Total Collision Counter |
| Ether | 1F8H | 4 | En_TCSE | R/W | Transmit Carrier Sense Error Counter |
| Ether | 1FCH | 4 | En_TIME | R/W | Transmit Internal MAC Error Counter |
| Ether | 200H | 4 | En_TXCR | R/W | Transmit Configuration Register |
| Ether | 204H | 4 | En_TXFCR | R/W | Transmit FIFO Control Register |
| Ether | 208H | 4 | En_TXDTR | W | Transmit Data Register |
| Ether | 20CH | 4 | En_TXSR | R | Transmit Status Register |
| Ether | 210H | 4 | N/A | - | Reserved for future use |
| Ether | 214H | 4 | En_TXDPR | R/W | Transmit Descriptor Register |
| Ether | 218H | 4 | En_RXCR | R/W | Receive Configuration Register |
| Ether | 21CH | 4 | En_RXFCR | R/W | Receive FIFO Control Register |
| Ether | 220H | 4 | En_RXDTR | R | Receive Data Register |
| Ether | 224H | 4 | En_RXSR | R | Receive Status Register |
| Ether | 228H | 4 | N/A | - | Reserved for future use |
| Ether | 22CH | 4 | En_RXDPR | R/W | Receive Descriptor Register |
| Ether | 230H | 4 | En_RXPDR | R/W | Receive Pool Descriptor Register |
| SYSCNT | 00H | 4 | S_GMR | R/W | General Mode Register |
| SYSCNT | 04H | 4 | S_GSR | R | General Status Register |
| SYSCNT | 08H | 4 | S_ISR | RC | Interrupt Status Register |
| SYSCNT | 0CH | 4 | S_IMR | W | Interrupt Mask Register |
| SYSCNT | 10H | 4 | S_NSR | R | NMI Status Register |
| SYSCNT | 14H | 4 | S_NMR | R/W | NMI Enable Register |
| SYSCNT | 18H | 4 | S_VER | R | Version Register |
| SYSCNT | 1CH | 4 | S_IOR | R/W | IO Port Register |
| SYSCNT | 20H-2FH | - | N/A | - | Reserved |
| SYSCNT | 30H | 4 | S_WRCR | W | Warm Reset Control Register |
| SYSCNT | 34H | 4 | S_WRSR | R | Warm Reset Status Register |
| SYSCNT | 38H | 4 | S_PWCR | W | Power Control Register |
| SYSCNT | 3CH | 4 | S_PWSR | R | Power Control Status Register |
| SYSCNT | 40H-48H | - | N/A | - | Reserved |
| SYSCNT | 4CH | 4 | S_ITCNTR | R/W | IBUS Timeout Timer Control Register |
| SYSCNT | 50H | 4 | S_ITSETR | R/W | IBUS Timeout Timer Set Register |

| Core | Offset | Register Length (Byte) | Name | Access by VR4120A | Description |
|---|---|---|---|---|---|
| SYSCNT | 54H-7FH | - | N/A | - | Reserved |
| SYSCNT | 80H | 4 | UARTDLL | R/W | UART, Divisor Latch LSB Register [DLAB=1] |
| SYSCNT | 80H | 4 | UARTRBR | R | UART, Receiver Buffer Register [DLAB=0,READ] |
| SYSCNT | 80H | 4 | UARTTHR | W | UART, Transmitter Holding Register [DLAB=0,WRITE] |
| SYSCNT | 84H | 4 | UARTDLM | R/W | UART, Divisor Latch MSB Register [DLAB=1] |
| SYSCNT | 84H | 4 | UARTIER | R/W | UART, Interrupt Enable Register [DLAB=0] |
| SYSCNT | 88H | 4 | UARTFCR | W | UART, FIFO control Register [WRITE] |
| SYSCNT | 88H | 4 | UARTIIR | R | UART, Interrupt ID Register [READ] |
| SYSCNT | 8CH | 4 | UARTLCR | R/W | UART, Line control Register |
| SYSCNT | 90H | 4 | UARTMCR | R/W | UART, Modem Control Register |
| SYSCNT | 94H | 4 | UARTLSR | R/W | UART, Line status Register |
| SYSCNT | 98H | 4 | UARTMSR | R/W | UART, Modem Status Register |
| SYSCNT | 9CH | 4 | UARTSCR | R/W | UART, Scratch Register |
| SYSCNT | A0H | 4 | DSUCNTR | R/W | DSU Control Register |
| SYSCNT | A4H | 4 | DSUSETR | R/W | DSU Dead Time Set Register |
| SYSCNT | A8H | 4 | DSUCLRR | W | DSU Clear Register |
| SYSCNT | ACH | 4 | DSUTIMR | R/W | DSU Elapsed Time Register |
| SYSCNT | B0H | 4 | TMMR | R/W | Timer Mode Register |
| SYSCNT | B4H | 4 | TM0CSR | R/W | Timer CH0 Count Set Register |
| SYSCNT | B8H | 4 | TM1CSR | R/W | Timer CH1 Count Set Register |
| SYSCNT | BCH | 4 | TM0CCR | R | Timer CH0 Current Count Register |
| SYSCNT | C0H | 4 | TM1CCR | R | Timer CH1 Current Count Register |
| SYSCNT | C4H-CFH | - | N/A | - | Reserved |
| SYSCNT | D0H | 4 | ECCR | W | EEPROM™ Command Control Register |
| SYSCNT | D4H | 4 | ERDR | R | EEPROM Read Data Register |
| SYSCNT | D8H | 4 | MACAR1 | R | MAC Address Register 1 |
| SYSCNT | DCH | 4 | MACAR2 | R | MAC Address Register 2 |
| SYSCNT | E0H | 4 | MACAR3 | R | MAC Address Register 3 |
| SYSCNT | E4H-FFH | - | N/A | - | Reserved |
| SYSCNT | 100H | 4 | RMMDR | R/W | Boot ROM Mode Register |
| SYSCNT | 104H | 4 | RMATR | R/W | Boot ROM Access Timing Register |
| SYSCNT | 108H | 4 | SDMDR | R/W | SDRAM Mode Register |
| SYSCNT | 10CH | 4 | SDTSR | R/W | SDRAM Type Selection Register |
| SYSCNT | 110H | 4 | SDPTR | R/W | SDRAM Precharge Timing Register |
| SYSCNT | 114H | 4 | SDRMR | R/W | SDRAM Precharge Mode Register |
| SYSCNT | 118H | 4 | SDRCR | R | SDRAM Precharge Timer Count Register |
| SYSCNT | 11CH | 4 | SDRMR | R/W | SDRAM Refresh Mode Register |
| SYSCNT | 120H | 4 | SDRCR | R | SDRAM Refresh Timer Count Register |
| SYSCNT | 124H | 4 | MBCR | R/W | Memory Bus Control Register |
| SYSCNT | 128H-FFFH | - | N/A | - | Reserved |
| USB | 0x00 | 4 | U_GMR | R/W | USB General Mode Register |
| USB | 0x04 | 4 | U_VER | R | USB Frame number/Version Register |
| USB | 0x08 | - | N/A | R/W | Reserved for future use |
| USB | 0x0c | - | N/A | R | Reserved for future use |
| USB | 0x10 | 4 | U_GSR1 | R | USB General Status Register 1 |
| USB | 0x14 | 4 | U_IMR1 | R/W | USB Interrupt Mask Register 1 |
| USB | 0x18 | 4 | U_GSR2 | R | USB General Status Resister 2 |
| USB | 0x1c | 4 | U_IMR2 | R/W | USB Interrupt Mask Register 2 |
| USB | 0x20 | 4 | U_EP0CR | R/W | USB EP0 Control Register |
| USB | 0x24 | 4 | U_EP1CR | R/W | USB EP1 Control Register |
| USB | 0x28 | 4 | U_EP2CR | R/W | USB EP2 Control Register |
| USB | 0x2c | 4 | U_EP3CR | R/W | USB EP3 Control Register |
| USB | 0x30 | 4 | U_EP4CR | R/W | USB EP4 Control Register |
| USB | 0x34 | 4 | U_EP5CR | R/W | USB EP5 Control Register |
| USB | 0x38 | 4 | U_EP6CR | R/W | USB EP6 Control Register |
| USB | 0x3c | - | N/A | - | Reserved for future use |
| USB | 0x40 | 4 | U_CMR | R/W | USB Command Register |
| USB | 0x44 | 4 | U_CA | R/W | USB Command Address Register |
| USB | 0x48 | 4 | U_TEPSR | R/W | USB Tx EndPoint Status Register |
| USB | 0x4c | - | N/A | - | Reserved for future use |
| USB | 0x50 | 4 | U_RP0IR | R/W | USB Rx Pool0 Information Register |
| USB | 0x54 | 4 | U_RP0AR | R | USB Rx Pool0 Address Register |
| USB | 0x58 | 4 | U_RP1IR | R/W | USB Rx Pool1 Information Register |
| USB | 0x5c | 4 | U_RP1AR | R | USB Rx Pool1 Address Register |
| USB | 0x60 | 4 | U_RP2IR | R/W | USB Rx Pool2 Information Register |
| USB | 0x64 | 4 | U_RP2AR | R | USB Rx Pool2 Address Register |
| USB | 0x68 | - | N/A | - | Reserved for future use |
| USB | 0x6c | - | N/A | - | Reserved for future use |
| USB | 0x70 | 4 | U_TMSA | R/W | USB Tx MailBox Start Address Register |
| USB | 0x74 | 4 | U_TMBA | R/W | USB Tx MailBox Bottom Address Register |
| USB | 0x78 | 4 | U_TMRA | R/W | USB Tx MailBox Read Address Register |
| USB | 0x7c | 4 | U_TMWA | R | USB Tx MailBox Write Address Register |
| USB | 0x80 | 4 | U_RMSA | R/W | USB Rx MailBox Start Address Register |
| USB | 0x84 | 4 | U_RMBA | R/W | USB Rx MailBox Bottom Address Register |

| Core | Offset | Register Length (Byte) | Name | Access by VR4120A | Description |
|---|---|---|---|---|---|
| USB | 0x88 | 4 | U_RMRA | R/W | USB Rx MailBox Read Address Register |
| USB | 0x8c | 4 | U_RMWA | R | USB Rx MailBox Write Address Register |
| USB | 0x90-0xff | - | N/A | - | Reserved for future use |
| USB | 0x100 | 4 | U_TDN | R | USB EP0 Tx Data Phase NAK Counter |
| USB | 0x104 | 4 | U_TDS | R | USB EP0 Tx Data Phase STALL Counter |
| USB | 0x108 | - | N/A | - | Reserved for future use |
| USB | 0x10c | - | N/A | - | Reserved for future use |
| USB | 0x110 | 4 | U_THT | R | USB EP0 Tx Handshake Phase Timeout Counter |
| USB | 0x114 | - | N/A | - | Reserved for future use |
| USB | 0x118 | - | N/A | - | Reserved for future use |
| USB | 0x11c | - | N/A | - | Reserved for future use |
| USB | 0x120 | 4 | U_RDT | R | USB EP0 Rx Data Phase Timeout Counter |
| USB | 0x124 | 4 | U_RDCER | R | USB EP0 Rx Data Phase CRC Error Counter |
| USB | 0x128 | 4 | U_RDBER | R | USB EP0 Rx Data Phase Bitstuff Error Counter |
| USB | 0x12c | 4 | U_RDTER | R | USB EP0 Rx Data Phase Data Toggle Error Counter |
| USB | 0x130 | 4 | U_RHT | R | USB EP0 Rx Handshake Phase Timeout Counter |
| USB | 0x134 | 4 | U_RHN | R | USB EP0 Rx Handshake Phase NAK Counter |
| USB | 0x138 | 4 | U_RHS | R | USB EP0 Rx Handshake Phase STALL Counter |
| USB | 0x13c | - | N/A | R | Reserved for future use |
| USB | 0x140-15f | - | N/A | - | Reserved for future use |
| USB | 0x160 | 4 | U_DT2 | R | USB EP2 Data Phase Timeout Counter |
| USB | 0x164 | 4 | U_DCER2 | R | USB EP2 Data Phase CRC Error Counter |
| USB | 0x168 | 4 | U_DBER2 | R | USB EP2 Data Phase Bitstuff Error Counter |
| USB | 0x16c | - | N/A | - | Reserved for future use |
| USB | 0x170-17f | - | N/A | - | Reserved for future use |
| USB | 0x180 | 4 | U_DN3 | R | USB EP3 Data Phase NAK Counter |
| USB | 0x184 | 4 | U_DS3 | R | USB EP3 Data Phase STALL Counter |
| USB | 0x188 | - | N/A | - | Reserved for future use |
| USB | 0x18c | - | N/A | - | Reserved for future use |
| USB | 0x190 | 4 | U_HT3 | R | USB EP3 Handshake Phase Timeout Counter |
| USB | 0x194 | - | N/A | - | Reserved for future use |
| USB | 0x198 | - | N/A | - | Reserved for future use |
| USB | 0x19c | - | N/A | - | Reserved for future use |
| USB | 0x1a0 | 4 | U_DT4 | R | USB EP4 Data Phase Timeout Counter |
| USB | 0x1a4 | 4 | U_DCER4 | R | USB EP4 Data Phase CRC Error Counter |
| USB | 0x1a8 | 4 | U_DBER4 | R | USB EP4 Data Phase Bitstuff Error Counter |
| USB | 0x1ac | 4 | U_DTER4 | R | USB EP4 Data Phase Data Toggle Error Counter |
| USB | 0x1b0 | 4 | U_HT4 | R | USB EP4 Handshake Phase Timeout Counter |
| USB | 0x1b4 | 4 | U_HN4 | R | USB EP4 Handshake Phase NAK Counter |
| USB | 0x1b8 | 4 | U_HS4 | R | USB EP4 Handshake Phase STALL Counter |
| USB | 0x1bc | - | N/A | - | Reserved for future use |
| USB | 0x1c0 | 4 | U_DN5 | R | USB EP5 Data Phase NAK Counter |
| USB | 0x1c4 | 4 | U_DS5 | R | USB EP5 Data Phase STALL Counter |
| USB | 0x1c8 | - | N/A | - | Reserved for future use |
| USB | 0x1cc | - | N/A | - | Reserved for future use |
| USB | 0x1d0 | 4 | U_HT5 | R | USB EP5 Handshake Phase Timeout Counter |
| USB | 0x1d4 | - | N/A | - | Reserved for future use |
| USB | 0x1d8 | - | N/A | - | Reserved for future use |
| USB | 0x1dc | - | N/A | - | Reserved for future use |
| USB | 0x1e0 | 4 | U_DT6 | R | USB EP6 Data Phase Timeout Counter |
| USB | 0x1e4 | 4 | U_DCER6 | R | USB EP6 Data Phase CRC Error Counter |
| USB | 0x1e8 | 4 | U_DBER6 | R | USB EP6 Data Phase Bitstuff Error Counter |
| USB | 0x1ec | 4 | U_DTER6 | R | USB EP6 Data Phase Data Toggle Error Counter |
| USB | 0x1f0 | 4 | U_HT6 | R | USB EP6 Handshake Phase Timeout Counter |
| USB | 0x1f4 | 4 | U_HN6 | R | USB EP6 Handshake Phase NAK Counter |
| USB | 0x1f8 | 4 | U_HS6 | R | USB EP6 Handshake Phase STALL Counter |
| USB | 0x1fc | - | N/A | - | Reserved for future use |

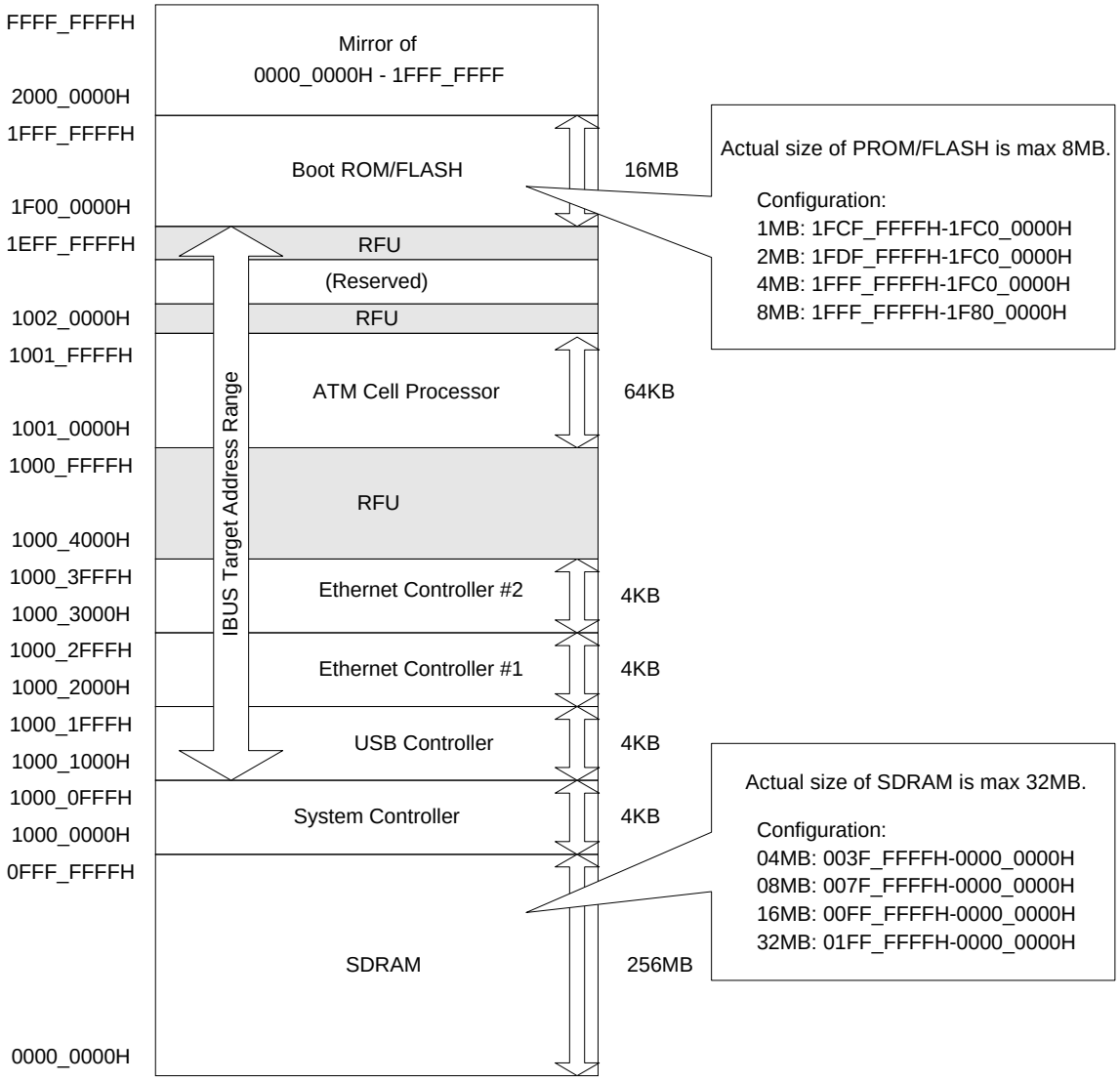
**Base address**

ATM Cell Processor (ATM) - 1001_0000H
Ethernet Controller (Ether) #1 (n = 1) - 1000_2000H
Ethernet Controller (Ether) #2 (n = 2) - 1000_3000H
USB Controller (USB) - 1000_1000H
System Controller (SYSCNT) - 1000_0000H

## 1.9 Memory Map

Using a 32-bit address, the processor physical address space encompasses 4Gbytes. VR4120A uses this 4Gbyte physical address space as shown in the following figure.

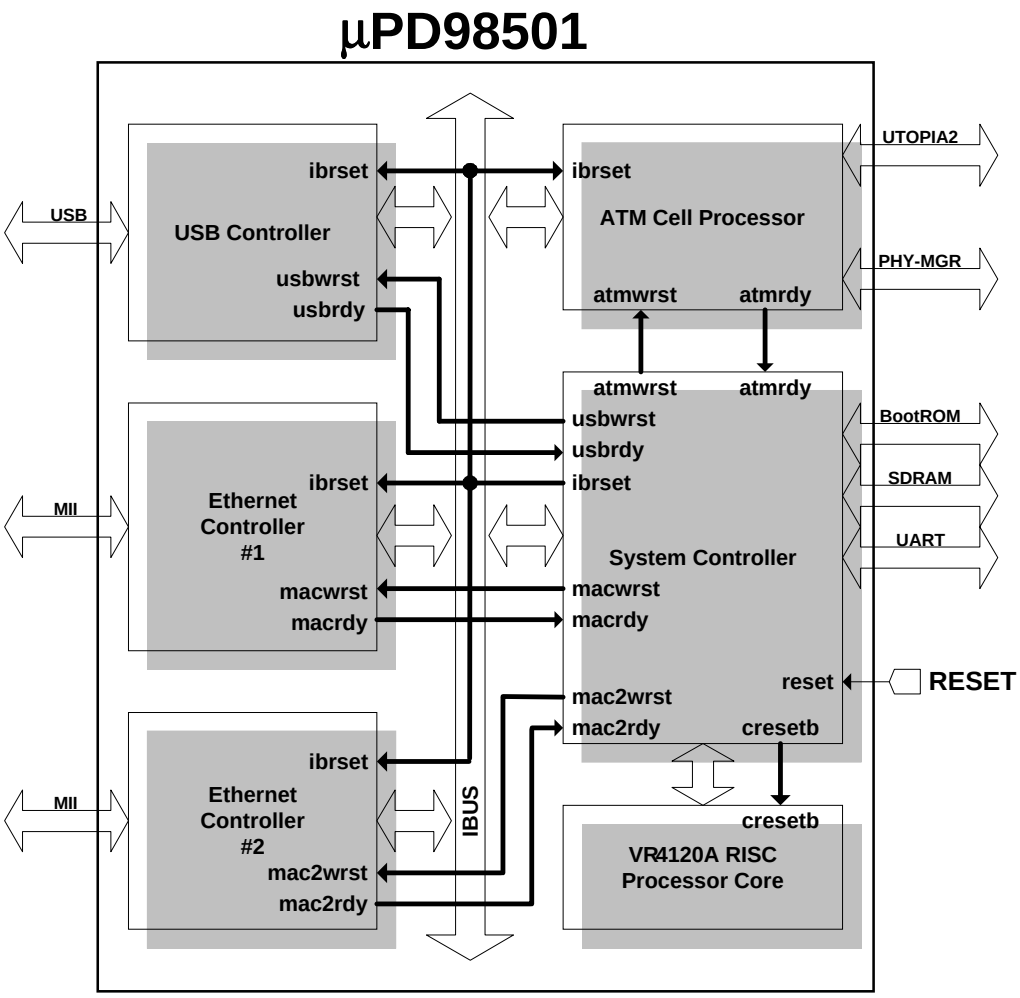
**Figure 1-9. Memory Map**



## 1.10 Reset Configuration

The rising edge of external system reset (RESET) serves as the μPD98501's reset. The System Controller generates the internal reset signal for the global reset of the μPD98501. After 4 IBUS clock, the System Controller deassert the internal reset signal synchronously with IBUS clock (66MHz). And also the System Controller generates the internal Cold Reset signal and Hot Reset signal for performing the cold reset of VR4120A. Once power to the μPD98501 is established, the System Controller asserts both Cold Reset signal and Hot Reset signal. After approximately 16ms Cold Reset signal assertion, the System Controller deasserts the Cold Reset signal synchronously with MOUT. And after 16 MOUT clock cycles at deassertion of the Cold Reset, the System Controller deasserts the Hot Reset signal synchronously with MOUT.

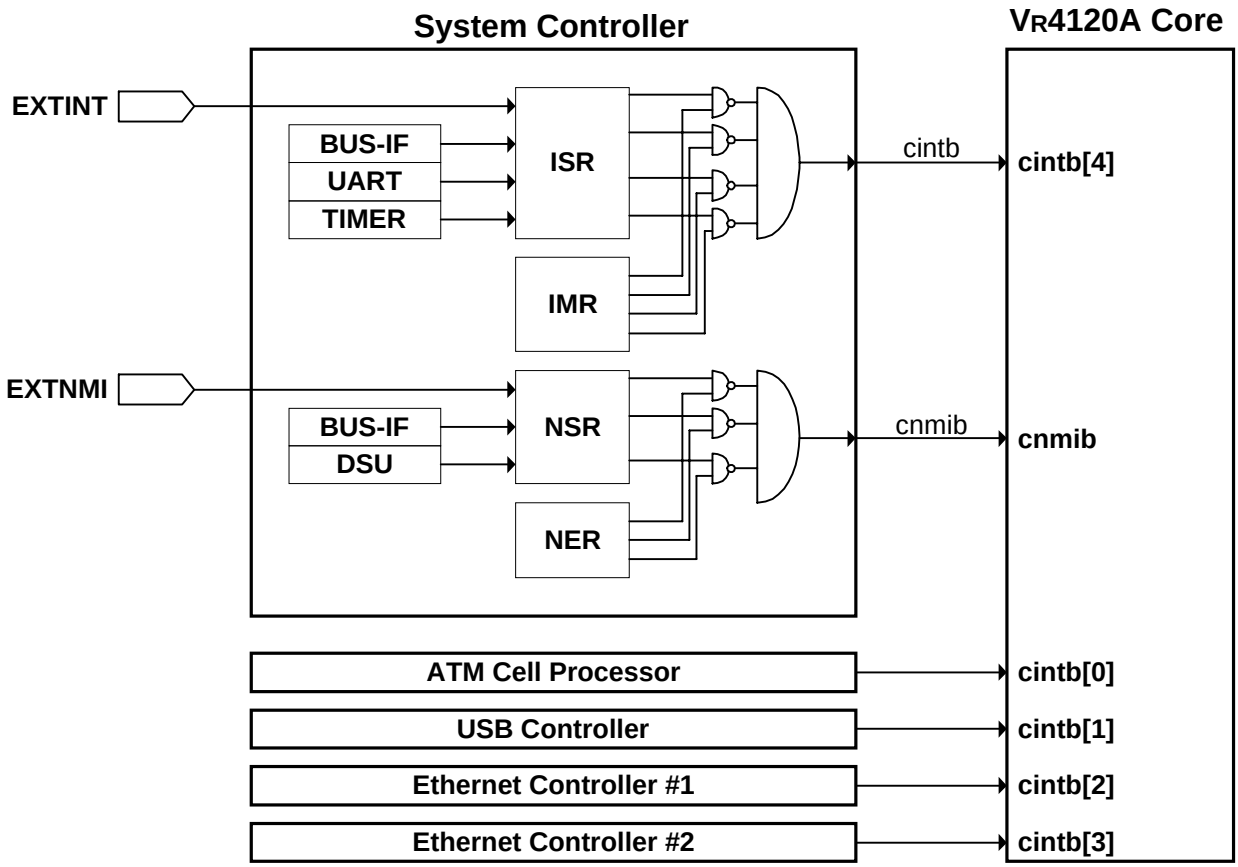
**Figure 1-10. Reset Configuration**



## 1.11 Interrupts

The controller supports maskable interrupts and Non-Maskable to the CPU.

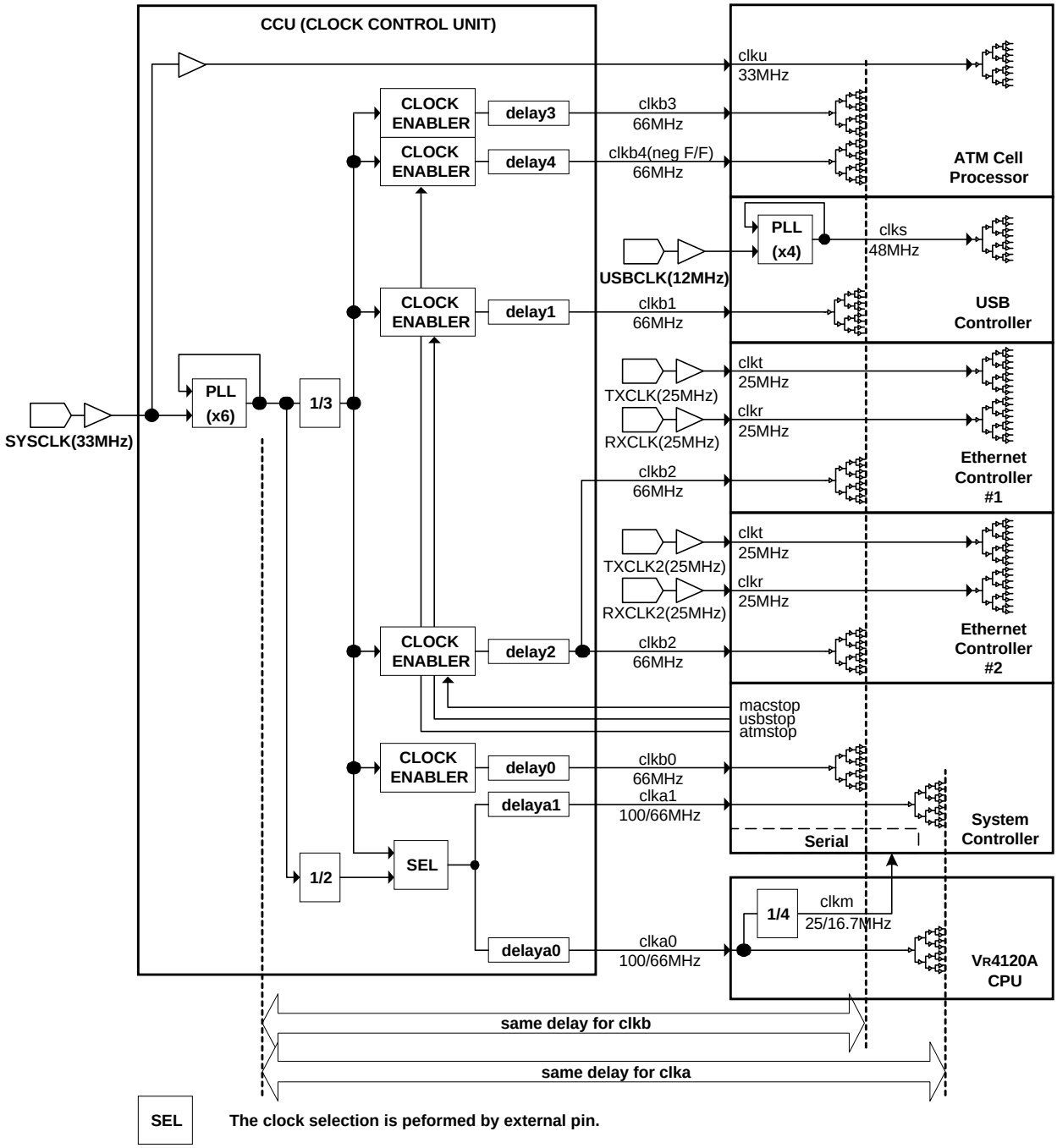
**Figure 1-11. Interrupt Signal Connection**



## 1.12 Clock Control Unit

This section describe μPD98501's internal clock is supplied by Clock Control Unit (CCU) with following figure.

**Figure 1-12. Block Diagram of Clock Control Unit**



# CHAPTER 2 VR4120A

**Caution The μPD98501 doesn't support MIPS16 instructions.**

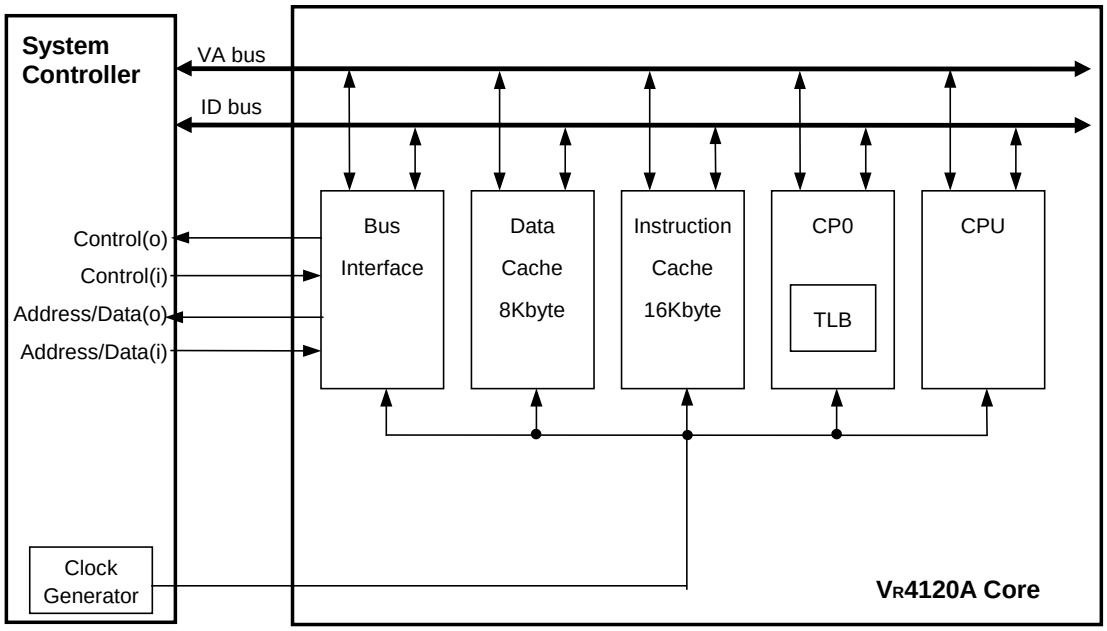
This chapter describes an VR4120A RISC Processor Core operation (MIPS instruction, Pipeline, etc.). Following in this Document, it is call for VR4120A RISC Processor Core with "VR4120A" or "VR4120A Core" simply.

## 2.1 Overview for VR4120A

Figure 2-1 shows the internal block diagram of the VR4120A core.

In addition to the conventional high-performance integer operation units, this CPU core has the full-associative format translation look aside buffer (TLB), which has 32 entries that provide mapping to 2- page pairs (odd and even) for one entry. Moreover, it also has instruction caches, data caches, and bus interface.

**Figure 2-1. VR4120A Core Internal Block Diagram**



### 2.1.1 Internal block configuration

#### 2.1.1.1 CPU

CPU has hardware resources to process an integer instruction. They are the 64-bit register file, 64-bit integer data bus, and multiply-and-accumulate operation unit.

#### 2.1.1.2 Coprocessor 0 (CP0)

CP0 incorporates a memory management unit (MMU) and exception handling function. MMU checks whether there is an access between different memory segments (user, supervisor, and kernel) by executing address conversion. The translation lookaside buffer (TLB) converts virtual addresses to physical addresses.

#### 2.1.1.3 Instruction cache

The instruction cache employs direct mapping, virtual index, and physical tag. Its capacity is 16 Kbytes.

#### 2.1.1.4 Data cache

The data cache employs direct mapping, virtual index, physical tag, and write back. Its capacity is 8 Kbytes.

#### 2.1.1.5 CPU bus interface

The CPU bus interface controls data transmission/reception between the VR4120A and the BCU, which is one of peripheral units. The VR4120A interface consists of two 32-bit multiplexed address/data buses (one is for input, and another is for output), clock signals, and control signals such as interrupts.

### 2.1.2 VR4120A registers

The VR4120A has the following registers.

- ✧ general-purpose register (GPR): 64 bits × 32

In addition, the processor provides the following special registers:

- ✧ 64-bit Program Counter (PC)
- ✧ 64-bit HI register, containing the integer multiply and divide upper doubleword result
- ✧ 64-bit LO register, containing the integer multiply and divide lower doubleword result

Two of the general-purpose registers have assigned the following functions:

- ✧ r0 is hardwired to a value of zero, and can be used as the target register for any instruction whose result is to be discarded. r0 can also be used as a source when a zero value is needed.
- ✧ r31 is the link register used by link instruction, such as JAL (Jump and Link) instructions. This register can be used for other instructions. However, be careful that use of the register by a link instruction will not coincide with use of the register for other operations.

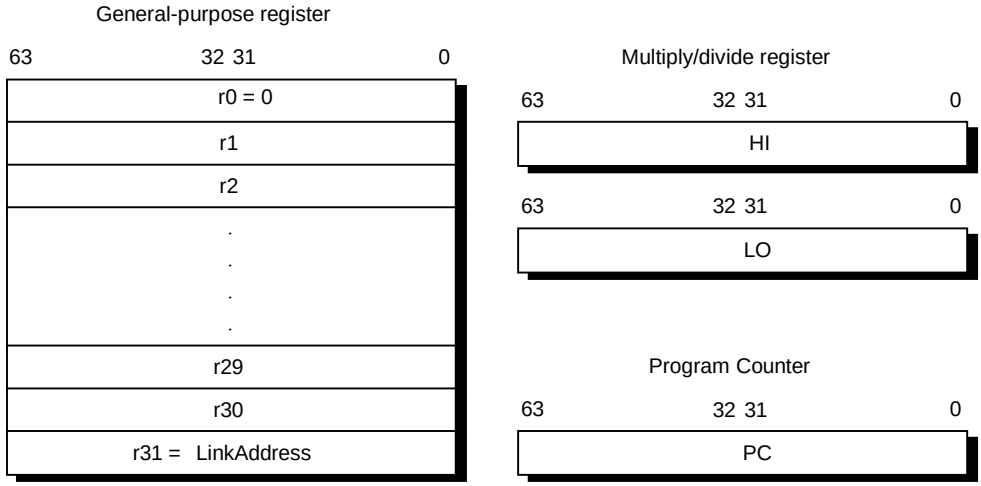
The register group is provided within the CP0 (system control coprocessor), to process exceptions and to manage addresses.

CPU registers can operate as either 32-bit or 64-bit registers, depending on the VR4120A processor operation mode.

The operation of the CPU register differs depending on what instructions are executed: 32-bit instructions or MIPS16 instructions. For details, refer to **Section 2.3 MIPS16 Instruction Set**.

Figure 2-2 shows the CPU registers.

**Figure 2-2. VR4120A Registers**




The VR4120A has no Program Status Word (PSW) register as such; this is covered by the Status and Cause registers incorporated within the System Control Coprocessor (CP0).

The CP0 registers are used for exception handling or address management. The overview of these registers is described in **2.1.5 Coprocessors (CP0)**.

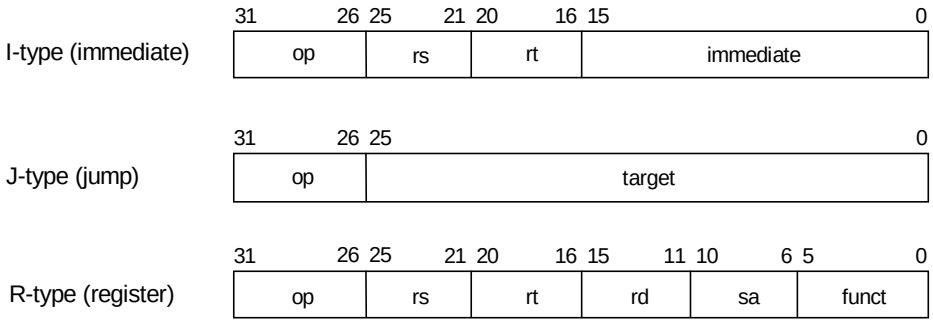
### 2.1.3 VR4120A instruction set overview

For CPU instructions, there are two types of instructions – 32-bit length instruction (MIPS III) and 16-bit length instruction (MIPS16).

#### 2.1.3.1 MIPS III instruction

All the CPU instructions are 32-bit length when executing MIPS III instructions, and they are classified into three instruction formats as shown in Figure 2-3: immediate (I-type), jump (J-type), and register (R-type). The field of each instruction format is described in **Section 2.2 MIPS III Instruction Set Summary**.

**Figure 2-3. CPU Instruction Formats (32-bit Length Instruction)**




The instruction set can be further divided into the following five groupings:

(a) Load and store instructions move data between memory and general-purpose registers. They are all immediate (I-type) instructions, since the only addressing mode supported is base register plus 16-bit, signed immediate offset.
(b) Computational instructions perform arithmetic, logical, shift, and multiply and divide operations on values in registers. They include R-type (in which both the operands and the result are stored in registers) and I-type (in which one operand is a 16-bit signed immediate value) formats.
(c) Jump and branch instructions change the control flow of a program. Jumps are always made to an absolute address formed by combining a 26-bit target address with the high-order bits of the Program Counter (J-type format) or register address (R-type format). The format of the branch instructions is I type. Branches have 16-bit offsets relative to the Program Counter. JAL instructions save their return address in register 31.
(d) Coprocessor 0 (System Control Coprocessor, CP0) instructions perform operations on CP0 registers to control the memory-management and exception-handling facilities of the processor.
(e) Special instructions perform system calls and breakpoint operations, or cause a branch to the general exception-handling vector based upon the result of a comparison. These instructions occur in both R-type and I-type formats.

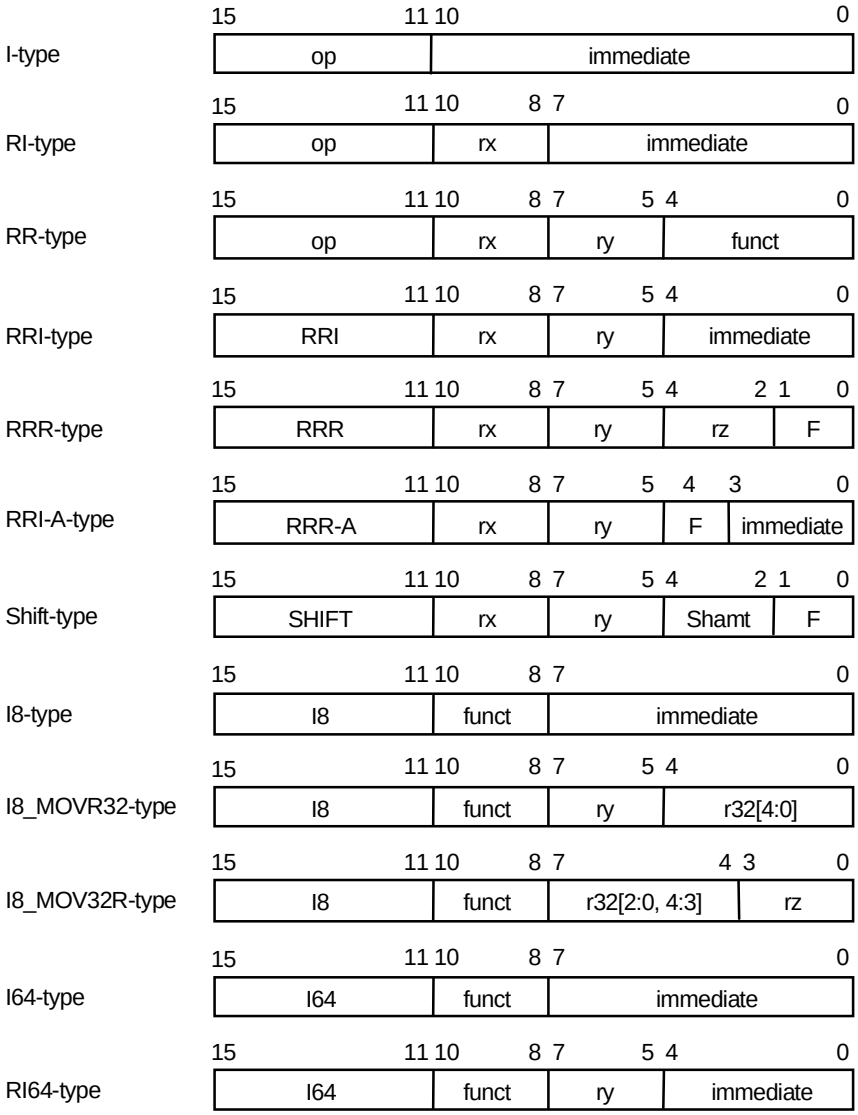
For the operation of each instruction, refer to **Section 2.2 MIPS III Instruction Set Summary** and **APPENDIX A MIPS III INSTRUCTION SET DETAILS**.

#### 2.1.3.2 MIPS16 instruction

All the CPU instructions except for JAL and JALX are 16-bit length when executing MIPS16 instructions, and they are classified into thirteen instruction formats as shown in Figure 2-4.

The field of each instruction format is described in **Section 2.3 MIPS16 Instruction Set**.

**Figure 2-4. CPU Instruction Formats (16-bit Length Instruction)**




The instruction set can be further divided into the following four groupings:

(a) Load and store instructions move data between memory and general-purpose registers. They include RRI-, RI-, I8-, and RI64-types.

(b) Computational instructions perform arithmetic, logical, shift, and multiply and divide operations on values in registers. They include RI-, RRIA-, I8-, RI64-, I64-, RR-, RRR-, I8_MOVR32-, and I8_MOV32R-types.

(c) Jump and branch instructions change the control flow of a program. They include JAL-/JALX-, RR-, RI-, I8-, and I-types.

(d) Special instructions are break and extend instructions. The break instruction transfers control to an exception handler. The extend instruction extends the immediate field of the next instruction. They are RR- and I-types. When extending the immediate field of the next instruction by using the extend instruction, one cycle is needed for executing the extend instruction, and another cycle is needed for executing the next instruction.

For more details of each instruction's operation, refer to **Section 2.3 MIPS16 Instruction Set** and **APPENDIX B MIPS16 INSTRUCTION SET FORMAT**.

### 2.1.4 Data formats and addressing

The VR4120A uses following four data formats:

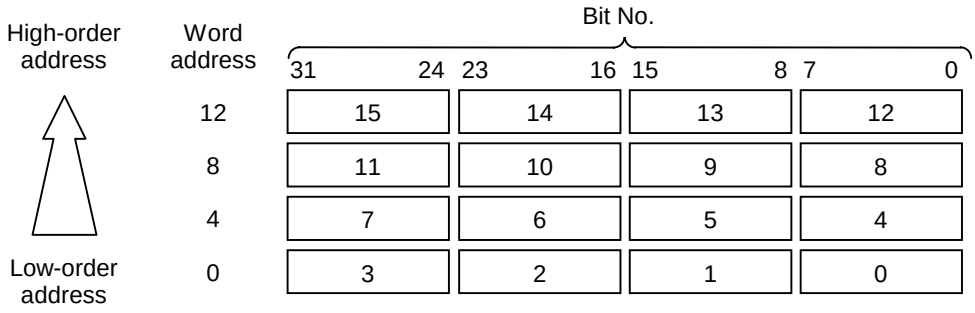
- ✧ Doubleword (64 bits)
- ✧ Word (32 bits)
- ✧ Halfword (16 bits)
- ✧ Byte (8 bits)

For the μPD98501, byte ordering within all of the larger data formats - halfword, word, doubleword - can be configured in either big-endian or little-endian order.

Endianness refers to the location of byte 0 within the multi-byte data structure.

When configured as a little-endian system, byte 0 is always the least-significant (rightmost) byte, which is compatible with iAPX™ and DEC VAX™ conventions. Figures 2-5 and 2-6 show this configuration.

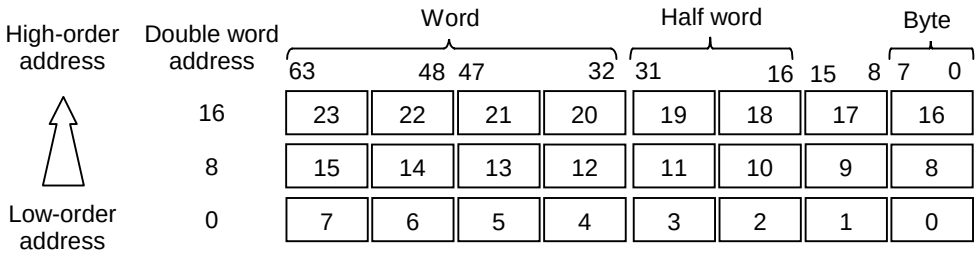
**Figure 2-5. Little-Endian Byte Ordering in Word Data**




**Remarks 1.** The lowest byte is the lowest address.

**2.** The address of word data is specified by the lowest byte's address.

**Figure 2-6. Little-Endian Byte Ordering in Double Word Data**




**Remarks 1.** The lowest byte is the lowest address.

**2.** The address of word data is specified by the lowest byte's address.

The CPU core uses the following byte boundaries for halfword, word, and doubleword accesses:

- ✧ Halfword: An even byte boundary (0, 2, 4...)
- ✧ Word: A byte boundary divisible by four (0, 4, 8...)
- ✧ Doubleword: A byte boundary divisible by eight (0, 8, 16...)

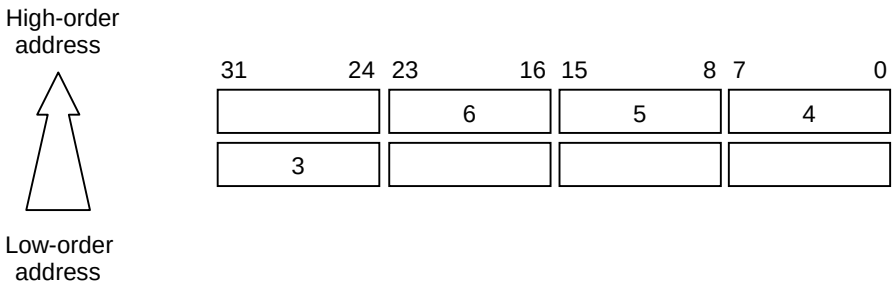
The following special instructions to load and store data that are not aligned on 4-byte (word) or 8-byte (doubleword) boundaries:

| | | | |
|---|---|---|---|
| LWL | LWR | SWL | SWR |
| LDL | LDR | SDL | SDR |

These instructions are used in pairs to provide an access to misaligned data. Accessing misaligned data incurs one additional instruction cycle over that required for accessing aligned data.

Figure 2-7 shows the access of a misaligned word that has byte address 3 for the little-endian conventions.

**Figure 2-7. Misaligned Word Accessing (Little-Endian)**



### 2.1.5 Coprocessors (CP0)

MIPS ISA defines 4 types of coprocessors (CP0 to CP3).

- CP0 translates virtual addresses to physical addresses, switches the operating mode (kernel, supervisor, or user mode), and manages exceptions. It also controls the cache subsystem to analyze a cause and to return from the error state.
- CP1 is reserved for floating-point instructions.
- CP2 is reserved for future definition by MIPS.
- CP3 is no longer defined. CP3 instructions are reserved for future extensions.

Figure 2-8 shows the definitions of the CP0 register, and Table 2-1 shows simple descriptions of each register. For the detailed descriptions of the registers related to the virtual system memory, refer to **Section 2.5 Memory Management System**. For the detailed descriptions of the registers related to exception handling, refer to **Section 2.6 Exception Processing**.

**Figure 2-8. CP0 Registers**

| Register No. | Register name | Register No. | Register name |
|---|---|---|---|
| 0 | Index Note 1 | 16 | Config Note 1 |
| 1 | Random Note 1 | 17 | LLAddr Note 1 |
| 2 | EntryLo0 Note 1 | 18 | WatchLo Note 2 |
| 3 | EntryLo1 Note 1 | 19 | WatchHi Note 2 |
| 4 | Context Note 2 | 20 | XContext Note 2 |
| 5 | PageMask Note 1 | 21 | RFU |
| 6 | Wired Note 1 | 22 | RFU |
| 7 | RFU | 23 | RFU |
| 8 | BadVAddr Note 1 | 24 | RFU |
| 9 | Count Note 2 | 25 | RFU |
| 10 | EntryHi Note 1 | 26 | PErr Note 2 |
| 11 | Compare Note 2 | 27 | CacheErr Note 2 |
| 12 | Status Note 2 | 28 | TagLo Note 1 |
| 13 | Cause Note 2 | 29 | TagHi Note 1 |
| 14 | EPC Note 2 | 30 | ErrorEPC Note 2 |
| 15 | PRId Note 1 | 31 | RFU |

**Notes 1.** for Memory management
**2.** for Exception handling

**Remark** RFU: Reserved for future use

**Table 2-1. System Control Coprocessor (CP0) Register Definitions**

| Register Number | Register Name | Description |
|---|---|---|
| 0 | Index | Programmable pointer to TLB array |
| 1 | Random | Pseudo-random pointer to TLB array (read only) |
| 2 | EntryLo0 | Low half of TLB entry for even VPN |
| 3 | EntryLo1 | Low half of TLB entry for odd VPN |
| 4 | Context | Pointer to kernel virtual PTE in 32-bit mode |
| 5 | PageMask | TLB page mask |
| 6 | Wired | Number of wired TLB entries |
| 7 | — | Reserved for future use |
| 8 | BadVAddr | Virtual address where the most recent error occurred |
| 9 | Count | Timer count |
| 10 | EntryHi | High half of TLB entry (including ASID) |
| 11 | Compare | Timer compare |
| 12 | Status | Status register |
| 13 | Cause | Cause of last exception |
| 14 | EPC | Exception Program Counter |
| 15 | PRId | Processor revision identifier |
| 16 | Config | Configuration register (specifying memory mode system) |
| 17 | LLAddr | Reserved for future use |
| 18 | WatchLo | Memory reference trap address low bits |
| 19 | WatchHi | Memory reference trap address high bits |
| 20 | XContext | Pointer to kernel virtual PTE in 64-bit mode |
| 21 to 25 | — | Reserved for future use |
| 26 | PErr Note | Cache parity bits |
| 27 | CacheErr Note | Index and status of cache error |
| 28 | TagLo | Cache Tag register (low) |
| 29 | TagHi | Cache Tag register (high) |
| 30 | ErrorEPC | Error Exception Program Counter |
| 31 | — | Reserved for future use |

**Note** This register is defined to maintain compatibility with the VR4100™. This register is not used in the μPD98501 hardware.

### 2.1.6 Floating-point unit (FPU)

The VR4120A does not support the floating-point unit (FPU). Coprocessor Unusable exception will occur if any FPU instructions are executed. If necessary, FPU instructions should be emulated by software in an exception handler.

### 2.1.7 CPU core memory management system (MMU)

The VR4120A has a 32-bit physical addressing range of 4 Gbytes. However, since it is rare for systems to implement a physical memory space as large as that memory space, the CPU provides a logical expansion of memory space by translating addresses composed in the large virtual address space into available physical memory addresses. The VR4120A supports the following two addressing modes:

- 32-bit mode, in which the virtual address space is divided into 2 Gbytes for user process and 2 Gbytes for the kernel.
- 64-bit mode, in which the virtual address is expanded to1 Tbyte ($2^{40}$ bytes) of user virtual address space.

A detailed description of these address spaces is given in **Section 2.5 Memory Management System**.

### 2.1.8 Translation lookaside buffer (TLB)

Virtual memory mapping is performed using the translation lookaside buffer (TLB). The TLB converts virtual addresses to physical addresses. It runs by a full-associative method. It has 32 entries, each mapping a pair of pages having a variable size (1 KB to 256 KB).

#### 2.1.8.1 Joint TLB (JTLB)

JTLB holds both an instruction address and data address.

For fast virtual-to-physical address decoding, the VR4120A uses a large, fully associative TLB (joint TLB) that translates 64 virtual pages to their corresponding physical addresses. The TLB is organized as 32 pairs of even-odd entries, and maps a virtual address and address space identifier (ASID) into the 4-Gbyte physical address space.

The page size can be configured, on a per-entry basis, to map a page size of 1 KB to 256 KB. A CP0 register stores the size of the page to be mapped, and that size is entered into the TLB when a new entry is written. Thus, operating systems can provide special purpose maps; for example, a typical frame buffer can be memory-mapped using only one TLB entry.

Translating a virtual address to a physical address begins by comparing the virtual address from the processor with the physical addresses in the TLB; there is a match when the virtual page number (VPN) of the address is the same as the VPN field of the entry, and either the Global (G) bit of the TLB entry is set, or the ASID field of the virtual address is the same as the ASID field of the TLB entry.

This match is referred to as a TLB hit. If there is no match, a TLB Miss exception is taken by the processor and software is allowed to refill the TLB from a page table of virtual/physical addresses in memory.

### 2.1.9 Operating modes

The VR4120A has three operating modes:

- ✧ User mode
- ✧ Supervisor mode
- ✧ Kernel mode

The manner in which memory addresses are translated or mapped depends on these operating modes. Refer to **Section 2.5 Memory Management System** for details.

### 2.1.10 Cache

The VR4120A chip incorporates instruction and data caches, which are independent of each other. This configuration enables high-performance pipeline operations. Both caches have a 64-bit data bus, enabling a one-clock access. These buses can be accessed in parallel. The instruction cache of the VR4120A has a storage capacity of 4 KB, while the data cache has a capacity of 1 KB.

A detailed description of caches is given in **Section 2.8 Cache Memory**.

### 2.1.11 Instruction Pipeline

The VR4120A has a 6-stage instruction pipeline. Under normal circumstances, one instruction is issued each cycle.

A detailed description of pipeline is provided in **Section 2.4 Pipeline**.

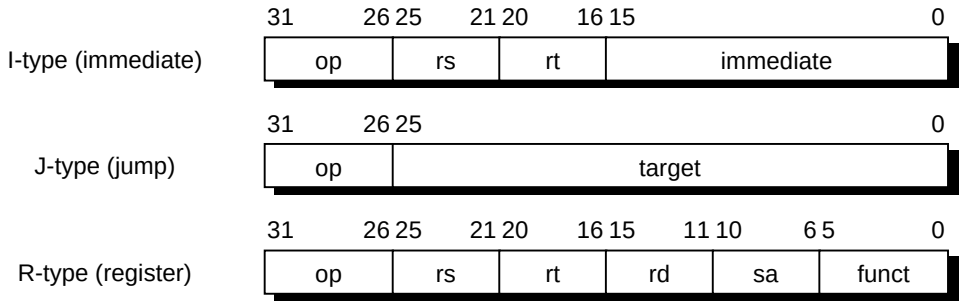
## 2.2 MIPS III Instruction Set Summary

This section is an overview of the MIPS III ISA central processing unit (CPU) instruction set; refer to **APPENDIX A MIPS III INSTRUCTION SET DETAILS** for detailed descriptions of individual CPU instructions.

### 2.2.1 MIPS III ISA instruction formats

Each MIPS III ISA CPU instruction consists of a single 32-bit word, aligned on a word boundary. There are three instruction formats - immediate (I-type), jump (J-type), and register (R-type) - as shown in Figure 2-9. The use of a small number of instruction formats simplifies instruction decoding, allowing the compiler to synthesize more complicated and less frequently used instruction and addressing modes from these three formats as needed.

**Figure 2-9. MIPS III ISA CPU Instruction Formats**




| | |
|---|---|
| op: | 6-bit operation code |
| rs: | 5-bit source register specifier |
| rt: | 5-bit target (source/destination) register specifier or branch condition |
| immediate: | 16-bit immediate value, branch displacement, or address displacement |
| target: | 26-bit unconditional branch target address |
| rd: | 5-bit destination register specifier |
| sa: | 5-bit shift amount |
| funct: | 6-bit function field |

#### 2.2.1.1 Support of the MIPS ISA

The VR4120A CORE does not support a multiprocessor operating environment. Thus the synchronization support instructions defined in the MIPS II and MIPS III ISA - the load linked and store conditional instructions - cause reserved instruction exception. The load/link (LL) bit is eliminated.

**Caution That the SYNC instruction is handled as a NOP instruction since all load/store instructions in this processor are executed in program order.**

### 2.2.2 Instruction classes

The CPU instructions are classified into five classes.

#### 2.2.2.1 Load and store instructions

Load and store are immediate (I-type) instructions that move data between memory and general registers. The only addressing mode that load and store instructions directly support is base register plus 16-bit signed immediate offset.

**(1) Scheduling a load delay slot**

A load instruction that does not allow its result to be used by the instruction immediately following is called a delayed load instruction. The instruction slot immediately following this delayed load instruction is referred to as the load delay slot.

In the VR4000 Series, a load instruction can be followed directly by an instruction that accesses a register that is loaded by the load instruction. In this case, however, an interlock occurs for a necessary number of cycles. Any instruction can follow a load instruction, but the load delay slot should be scheduled appropriately for both performance and compatibility with the VR Series™ microprocessors. For detail, see **Section 2.4 Pipeline**.

**(2) Store delay slot**

When a store instruction is writing data to a cache, the data cache is kept busy at the DC and WB stages. If an instruction (such as load) that follows directly the store instruction accesses the data cache in the DC stage, a hardware-driven interlock occurs. To overcome this problem, the store delay slot should be scheduled.

**Table 2-2. Number of Delay Slot Cycles Necessary for Load and Store Instructions**

| Instruction | Necessary Number of PCycles |
|---|---|
| Load | 1 |
| Store | 1 |

**(3) Defining access types**

Access type indicates the size of a VR4120A data item to be loaded or stored, set by the load or store instruction opcode. Access types and accessed byte are shown in Table 2-3.

Regardless of access type or byte ordering (endianness), the address given specifies the low-order byte in the addressed field. For a little-endian configuration, the low-order byte is the least-significant byte.

The access type, together with the low-order three bits of the address, define the bytes accessed within the addressed doubleword (shown in Table 2-3). Only the combinations shown in Table 2-3 are permissible; other combinations cause address error exceptions.

Tables 2-4 and 2-5 list the ISA-defined load/store instructions and extended-ISA instructions, respectively.

**Table 2-3. Byte Specification Related to Load and Store Instructions**

| Access Type (Value) | Low-Order Address Bit | | | Accessed Byte | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Little Endian | | | | | | | |
| | 2 | 1 | 0 | 63 | | | | | | | 0 |
| Doubleword (7) | 0 | 0 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 7-byte (6) | 0 | 0 | 0 | | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | 0 | 0 | 1 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | |
| 6-byte (5) | 0 | 0 | 0 | | | 5 | 4 | 3 | 2 | 1 | 0 |
| | 0 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | | |
| 5-byte (4) | 0 | 0 | 0 | | | | 4 | 3 | 2 | 1 | 0 |
| | 0 | 1 | 1 | 7 | 6 | 5 | 4 | 3 | | | |
| Word (3) | 0 | 0 | 0 | | | | | 3 | 2 | 1 | 0 |
| | 1 | 0 | 0 | 7 | 6 | 5 | 4 | | | | |
| Triple byte (2) | 0 | 0 | 0 | | | | | | 2 | 1 | 0 |
| | 0 | 0 | 1 | | | | | 3 | 2 | 1 | |
| | 1 | 0 | 0 | | 6 | 5 | 4 | | | | |
| | 1 | 0 | 1 | 7 | 6 | 5 | | | | | |
| Halfword (1) | 0 | 0 | 0 | | | | | | | 1 | 0 |
| | 0 | 1 | 0 | | | | | 3 | 2 | | |
| | 1 | 0 | 0 | | | 5 | 4 | | | | |
| | 1 | 1 | 0 | 7 | 6 | | | | | | |
| Byte (0) | 0 | 0 | 0 | | | | | | | | 0 |
| | 0 | 0 | 1 | | | | | | | 1 | |
| | 0 | 1 | 0 | | | | | | 2 | | |
| | 0 | 1 | 1 | | | | | 3 | | | |
| | 1 | 0 | 0 | | | | 4 | | | | |
| | 1 | 0 | 1 | | | 5 | | | | | |
| | 1 | 1 | 0 | | 6 | | | | | | |
| | 1 | 1 | 1 | 7 | | | | | | | |

**Table 2-4. Load/Store Instruction**

| Instruction | Format and Description | op | base | rt | offset |
|---|---|---|---|---|---|
| Load Byte | LB rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. The bytes of the memory location specified by the address are sign extended and loaded into register rt. | | | | |
| Load Byte Unsigned | LBU rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. The bytes of the memory location specified by the address are zero extended and loaded into register rt. | | | | |
| Load Halfword | LH rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. The halfword of the memory location specified by the address is sign extended and loaded to register rt. | | | | |
| Load Halfword Unsigned | LHU rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. The halfword of the memory location specified by the address is zero extended and loaded to register rt. | | | | |
| Load Word | LW rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. The word of the memory location specified by the address is sign extended and loaded to register rt. In the 64-bit mode, it is further sign extended to 64 bits. | | | | |
| Load Word Left | LWL rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. Shifts to the left the word whose address is specified so that the address-specified byte is at the left-most position of the word. The result of the shift operation is merged with the contents of register rt and loaded to register rt. In the 64-bit mode, it is further sign extended to 64 bits. | | | | |
| Load Word Right | LWR rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. Shifts to the right the word whose address is specified so that the address-specified byte is at the right-most position of the word. The result of the shift operation is merged with the contents of register rt and loaded to register rt. In the 64-bit mode, it is further sign extended to 64 bits. | | | | |
| Store Byte | SB rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. The least significant byte of register rt is stored to the memory location specified by the address. | | | | |
| Store Halfword | SH rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. The least significant halfword of register rt is stored to the memory location specified by the address. | | | | |
| Store Word | SW rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. The lower word of register rt is stored to the memory location specified by the address. | | | | |

**Table 2-5. Load/Store Instruction (Extended ISA)**

| Instruction | Format and Description | op | base | rt | offset |
|---|---|---|---|---|---|
| Store Word Left | SWL rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. Shifts to the right the contents of register rt so that the left-most byte of the word is in the position of the address-specified byte. The result is stored to the lower word in memory. | | | | |
| Store Word Right | SWR rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. Shifts to the left the contents of register rt so that the right-most byte of the word is in the position of the address-specified byte. The result is stored to the upper word in memory. | | | | |
| Load Doubleword | LD rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. The doubleword of the memory location specified by the address are loaded into register rt. | | | | |
| Load Doubleword Left | LDL rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. Shifts to the left the double word whose address is specified so that the address-specified byte is at the left-most position of the double word. The result of the shift operation is merged with the contents of register rt and loaded to register rt. | | | | |
| Load Doubleword Right | LDR rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. Shifts to the right the double word whose address is specified so that the address-specified byte is at the right-most position of the double word. The result of the shift operation is merged with the contents of register rt and loaded to register rt. | | | | |
| Load Word Unsigned | LWU rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. The word of the memory location specified by the address are zero extended and loaded into register rt. | | | | |
| Store Doubleword | SD rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. The contents of register rt are stored to the memory location specified by the address. | | | | |
| Store Doubleword Left | SDL rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. Shifts to the right the contents of register rt so that the left-most byte of the double word is in the position of the address-specified byte. The result is stored to the lower doubleword in memory. | | | | |
| Store Doubleword Right | SDR rt, offset (base)<br>The offset is sign extended and then added to the contents of the register base to form the virtual address. Shifts to the left the contents of register rt so that the right-most byte of the double word is in the position of the address-specified byte. The result is stored to the upper doubleword in memory. | | | | |

#### 2.2.2.2 Computational instructions

Computational instructions perform arithmetic, logical, and shift operations on values in registers. Computational instructions can be either in register (R-type) format, in which both operands are registers, or in immediate (I-type) format, in which one operand is a 16-bit immediate.

Computational instructions are classified as:

(1) ALU immediate instructions

(2) Three-operand type instructions

(3) Shift instructions

(4) Multiply/divide instructions

To maintain data compatibility between the 64- and 32-bit modes, it is necessary to sign-extend 32-bit operands correctly. If the sign extension is not correct, the 32-bit operation result is meaningless.

**Table 2-6. ALU Immediate Instruction**

| Instruction | Format and Description | op | rs | rt | immediate |
|---|---|---|---|---|---|
| Add Immediate | ADDI rt, rs, immediate<br>The 16-bit immediate is sign extended and then added to the contents of register rs to form a 32-bit result. The result is stored into register rt. In the 64-bit mode, the operand must be sign extended. An exception occurs on the generation of 2's complement overflow. | | | | |
| Add Immediate Unsigned | ADDIU rt, rs, immediate<br>The 16-bit immediate is sign extended and then added to the contents of register rs to form a 32-bit result. The result is stored into register rt. In the 64-bit mode, the operand must be sign extended. No exception occurs on the generation of integer overflow. | | | | |
| Set On Less Than Immediate | SLTI rt, rs, immediate<br>The 16-bit immediate is sign extended and then compared to the contents of register rt treating both operands as signed integers. If rs is less than the immediate, the result is set to 1; otherwise, the result is set to 0. The result is stored to register rt. | | | | |
| Set On Less Than Immediate Unsigned | SLTIU rt, rs, immediate<br>The 16-bit immediate is sign extended and then compared to the contents of register rt treating both operands as unsigned integers. If rs is less than the immediate, the result is set to 1; otherwise, the result is set to 0. The result is stored to register rt. | | | | |
| And Immediate | ANDI rt, rs, immediate<br>The 16-bit immediate is zero extended and then ANDed with the contents of the register. The result is stored into register rt. | | | | |
| Or Immediate | ORI rt, rs, immediate<br>The 16-bit immediate is zero extended and then ORed with the contents of the register. The result is stored into register rt. | | | | |
| Exclusive Or Immediate | XORI rt, rs, immediate<br>The 16-bit immediate is zero extended and then Ex-ORed with the contents of the register. The result is stored into register rt. | | | | |
| Load Upper Immediate | LUI rt, immediate<br>The 16-bit immediate is shifted left by 16 bits to set the lower 16 bits of word to 0. The result is stored into register rt. In the 64-bit mode, the operand must be sign extended. | | | | |

**Table 2-7. ALU Immediate Instruction (Extended ISA)**

| Instruction | Format and Description | op | rs | rt | immediate |
|---|---|---|---|---|---|
| Doubleword Add Immediate | DADDI rt, rs, immediate<br>The 16-bit immediate is sign extended to 64 bits and then added to the contents of register rs to form a 64-bit result. The result is stored into register rt.<br>An exception occurs on the generation of integer overflow. | | | | |
| Doubleword Add Immediate Unsigned | DADDIU rt, rs, immediate<br>The 16-bit immediate is sign extended to 64 bits and then added to the contents of register rs to form a 64-bit result. The result is stored into register rt.<br>No exception occurs on the generation of overflow. | | | | |

**Table 2-8. Three-Operand Type Instruction**

| Instruction | Format and Description | op | rs | rt | rd | sa | funct |
|---|---|---|---|---|---|---|---|
| Add | ADD rd, rs, rt<br>The contents of registers rs and rt are added together to form a 32-bit result. The result is stored into register rd. In the 64-bit mode, the operand must be sign extended.<br>An exception occurs on the generation of integer overflow. | | | | | | |
| Add Unsigned | ADDU rd, rs, rt<br>The contents of registers rs and rt are added together to form a 32-bit result. The result is stored into register rd. In the 64-bit mode, the operand must be sign extended.<br>No exception occurs on the generation of integer overflow. | | | | | | |
| Subtract | SUB rd, rs, rt<br>The contents of register rt are subtracted from the contents of register rs. The 32-bit result is stored into register rd. In the 64-bit mode, the operand must be sign extended.<br>An exception occurs on the generation of integer overflow. | | | | | | |
| Subtract Unsigned | SUBU rd, rs, rt<br>The contents of register rt are subtracted from the contents of register rs. The 32-bit result is stored into register rd. In the 64-bit mode, the operand must be sign extended.<br>No exception occurs on the generation of integer overflow. | | | | | | |
| Set On Less Than | SLT rd, rs, rt<br>The contents of registers rs and rt are compared, treating both operands as signed integers.<br>If the contents of register rs is less than that of register rt, the result is set to 1; otherwise, the result is set to 0. The result is stored to register rd. | | | | | | |
| Set On Less Than Unsigned | SLTU rd, rs, rt<br>The contents of registers rs and rt are compared treating both operands as unsigned integers.<br>If the contents of register rs is less than that of register rt, the result is set to 1; otherwise, the result is set to 0. The result is stored to register rd. | | | | | | |
| And | AND rd, rt, rs<br>The contents of register rs are logical ANDed with that of general register rt bit-wise. The result is stored to register rd. | | | | | | |
| Or | OR rd, rt, rs<br>The contents of register rs are logical ORed with that of general register rt bit-wise. The result is stored to register rd. | | | | | | |
| Exclusive Or | XOR rd, rt, rs<br>The contents of register rs are logical Ex-ORed with that of general register rt bit-wise. The result is stored to register rd. | | | | | | |
| Nor | NOR rd, rt, rs<br>The contents of register rs are logical NORed with that of general register rt bit-wise. The result is stored to register rd. | | | | | | |

**Table 2-9. Three-Operand Type Instruction (Extended ISA)**

| Instruction | Format and Description | op | rs | rt | rd | sa | funct |
|---|---|---|---|---|---|---|---|
| Doubleword Add | DADD rd, rt, rs<br>The contents of register rs are added to that of register rt. The 64-bit result is stored into register rd.<br>An exception occurs on the generation of integer overflow. | | | | | | |
| Doubleword Add Unsigned | DADDU rd, rt, rs<br>The contents of register rs are added to that of register rt. The 64-bit result is stored into register rd.<br>No exception occurs on the generation of integer overflow. | | | | | | |
| Doubleword Subtract | DSUB rd, rt, rs<br>The contents of register rt are subtracted from that of register rs. The 64-bit result is stored into register rd.<br>An exception occurs on the generation of integer overflow. | | | | | | |
| Doubleword Subtract Unsigned | DSUBU rd, rt, rs<br>The contents of register rt are subtracted from that of register rs. The 64-bit result is stored into register rd.<br>No exception occurs on the generation of integer overflow. | | | | | | |

**Table 2-10. Shift Instruction**

| Instruction | Format and Description | op | rs | rt | rd | sa | funct |
|---|---|---|---|---|---|---|---|
| Shift Left Logical | SLL rd, rs, sa<br>The contents of register rt are shifted left by sa bits and zeros are inserted into the emptied lower bits.<br>The 32-bit result is stored into register rd. In the 64-bit mode, the operand must be sign extended. | | | | | | |
| Shift Right Logical | SRL rd, rs, sa<br>The contents of register rt are shifted right by sa bits and zeros are inserted into the emptied higher bits.<br>The 32-bit result is stored into register rd. In the 64-bit mode, the operand must be sign extended. | | | | | | |
| Shift Right Arithmetic | SRA rd, rt, sa<br>The contents of register rt are shifted right by sa bits and the emptied higher bits are sign extended.<br>The 32-bit result is stored into register rd. In the 64-bit mode, the operand must be sign extended. | | | | | | |
| Shift Left Logical Variable | SLLV rd, rt, rs<br>The contents of register rt are shifted left and zeros are inserted into the emptied lower bits. The lower five bits of register rs specify the shift count.<br>The 32-bit result is stored into register rd. In the 64-bit mode, the operand must be sign extended. | | | | | | |
| Shift Right Logical Variable | SRLV rd, rt, rs<br>The contents of register rt are shifted right and zeros are inserted into the emptied higher bits. The lower five bits of register rs specify the shift count.<br>The 32-bit result is stored into register rd. In the 64-bit mode, the operand must be sign extended. | | | | | | |
| Shift Right Arithmetic Variable | SRAV rd, rt, rs<br>The contents of register rt are shifted right and the emptied higher bits are sign extended. The lower five bits of register rs specify the shift count.<br>The 32-bit result is stored into register rd. In the 64-bit mode, the operand must be sign extended. | | | | | | |

**Table 2-11. Shift Instruction (Extended ISA)**

| Instruction | Format and Description | op | rs | rt | rd | sa | funct |
|---|---|---|---|---|---|---|---|
| Doubleword Shift Left Logical | DSLL rd, rt, sa<br>The contents of register rt are shifted left by sa bits and zeros are inserted into the emptied lower bits.<br>The 64-bit result is stored into register rd. | | | | | | |
| Doubleword Shift Right Logical | DSRL rd, rt, sa<br>The contents of register rt are shifted right by sa bits and zeros are inserted into the emptied higher bits.<br>The 64-bit result is stored into register rd. | | | | | | |
| Doubleword Shift Right Arithmetic | DSRA rd, rt, sa<br>The contents of register rt are shifted right by sa bits and the emptied higher bits are sign extended.<br>The 64-bit result is stored into register rd. | | | | | | |
| Doubleword Shift Left Logical Variable | DSLLV rd, rt, rs<br>The contents of register rt are shifted left and zeros are inserted into the emptied lower bits. The lower six bits of register rs specify the shift count.<br>The 64-bit result is stored into register rd. | | | | | | |
| Doubleword Shift Right Logical Variable | DSRLV rd, rt, rs<br>The contents of register rt are shifted right and zeros are inserted into the emptied higher bits. The lower six bits of register rs specify the shift count. The 64-bit result is stored into register rd. | | | | | | |
| Doubleword Shift Right Arithmetic Variable | DSRAV rd, rt, rs<br>The contents of register rt are shifted right and the emptied higher bits are sign extended. The lower six bits of register rs specify the shift count.<br>The 64-bit result is stored into register rd. | | | | | | |
| Doubleword Shift Left Logical + 32 | DSLL32 rd, rt, sa<br>The contents of register rt are shifted left by 32 + sa bits and zeros are inserted into the emptied lower bits.<br>The 64-bit result is stored into register rd. | | | | | | |
| Doubleword Shift Right Logical + 32 | DSRL32 rd, rt, sa<br>The contents of register rt are shifted right by 32 + sa bits and zeros are inserted into the emptied higher bits.<br>The 64-bit result is stored into register rd. | | | | | | |
| Doubleword Shift Right Arithmetic + 32 | DSRA32 rd, rt, sa<br>The contents of register rt are shifted right by 32 + sa bits and the emptied higher bits are sign extended.<br>The 64-bit result is stored into register rd. | | | | | | |

**Table 2-12. Multiply/Divide Instructions**

| Instruction | Format and Description | op | rs | rt | rd | sa | funct |
|---|---|---|---|---|---|---|---|
| Multiply | MULT rs, rt<br>The contents of registers rt and rs are multiplied, treating both operands as 32-bit signed integers. The 64-bit result is stored into special registers HI and LO. In the 64-bit mode, the operand must be sign extended. | | | | | | |
| Multiply Unsigned | MULTU rs, rt<br>The contents of registers rt and rs are multiplied, treating both operands as 32-bit unsigned integers. The 64-bit result is stored into special registers HI and LO. In the 64-bit mode, the operand must be sign extended. | | | | | | |
| Divide | DIV rs, rt<br>The contents of register rs are divided by that of register rt, treating both operands as 32-bit signed integers. The 32-bit quotient is stored into special register LO, and the 32-bit remainder is stored into special register HI. In the 64-bit mode, the operand must be sign extended. | | | | | | |
| Divide Unsigned | DIVU rs, rt<br>The contents of register rs are divided by that of register rt, treating both operands as 32-bit unsigned integers. The 32-bit quotient is stored into special register LO, and the 32-bit remainder is stored into special register HI. In the 64-bit mode, the operand must be sign extended. | | | | | | |
| Move From HI | MFHI rd<br>The contents of special register HI are loaded into register rd. | | | | | | |
| Move From LO | MFLO rd<br>The contents of special register LO are loaded into register rd. | | | | | | |
| Move To HI | MTHI rs<br>The contents of register rs are loaded into special register HI. | | | | | | |
| Move To LO | MTLO rs<br>The contents of register rs are loaded into special register LO. | | | | | | |

**Table 2-13. Multiply/Divide Instructions (Extended ISA)**

| Instruction | Format and Description | op | rs | rt | rd | sa | funct |
|---|---|---|---|---|---|---|---|
| Doubleword Multiply | DMULT rs, rt<br>The contents of registers rt and rs are multiplied, treating both operands as signed integers.<br>The 128-bit result is stored into special registers HI and LO. | | | | | | |
| Doubleword Multiply Unsigned | DMULTU rs, rt<br>The contents of registers rt and rs are multiplied, treating both operands as unsigned integers.<br>The 128-bit result is stored into special registers HI and LO. | | | | | | |
| Doubleword Divide | DDIV rs, rt<br>The contents of register rs are divided by that of register rt, treating both operands as signed integers.<br>The 64-bit quotient is stored into special register LO, and the 64-bit remainder is stored into special register HI. | | | | | | |
| Doubleword Divide Unsigned | DDIVU rs, rt<br>The contents of register rs are divided by that of register rt, treating both operands as unsigned integers.<br>The 64-bit quotient is stored into special register LO, and the 64-bit remainder is stored into special register HI. | | | | | | |
| Multiply and Add Accumulate | MACC{h}{u}{s} rd, rs, rt<br>The contents of registers rt and rs are multiplied, treating both operands as 32-bit signed integers. The result is added to the combined value of special registers HI and LO. The 64-bit result is stored into special registers HI and LO.<br>If h=0, the same data as that stored in register LO is also stored in register rd; if h=1, the same data as that stored in register HI is also stored in register rd.<br>If u is specified, the operand is treated as unsigned data.<br>If s is specified, registers rs and rd are treated as a 16-bit value (32 bits sign- or zero-extended), and the value obtained by combining registers HI and LO is treated as a 32-bit value (64 bits sign- or zero-extended). Moreover, saturation processing is performed for the operation result in the format specified with u. | | | | | | |
| Doubleword Multiply and Add Accumulate | DMACC{h}{u}{s} rd, rs, rt<br>The contents of registers rt and rs are multiplied, treating both operands as 32-bit signed integers. The result is added to value of special register LO. The 64-bit result is stored into special register LO.<br>If h=0, the same data as that stored in register LO is also stored in register rd; if h=1, undefined data is stored in register rd.<br>If u is specified, the operand is treated as unsigned data.<br>If s is specified, registers rs and rd are treated as a 16-bit value (32 bits sign- or zero-extended), and register LO is treated as a 32-bit value (64 bits sign- or zero-extended). Moreover, saturation processing is performed for the operation result in the format specified with u. | | | | | | |

MFHI and MFLO instructions after a multiply or divide instruction generate interlocks to delay execution of the next instruction, inhibiting the result from being read until the multiply or divide instruction completes.

Table 2-14 gives the number of processor cycles (PCycles) required to resolve interlock or stall between various multiply or divide instructions and a subsequent MFHI or MFLO instruction.

**Table 2-14. Number of Stall Cycles in Multiply and Divide Instructions**

| Instruction | Number of Instruction Cycles |
|---|---|
| MULT | 1 |
| MULTU | 1 |
| DIV | 36 |
| DIVU | 36 |
| DMULT | 3 |
| DMULTU | 3 |
| DDIV | 68 |
| DDIVU | 68 |
| MACC | 0 |
| DMACC | 0 |

#### 2.2.2.3 Jump and branch instructions

Jump and branch instructions change the control flow of a program. All jump and branch instructions occur with a delay of one instruction: that is, the instruction immediately following the jump or branch instruction (this is known as the instruction in the delay slot) always executes while the target instruction is being fetched from memory.

For instructions involving a link (such as JAL and BLTZAL), the return address is saved in register r31.

**Table 2-15. Number of Delay Slot Cycles in Jump and Branch Instructions**

| Instruction | Necessary Number of Cycles |
|---|---|
| Branch instruction | 1 |
| Jump instruction | 1 |

**(1) Overview of jump instructions**

Subroutine calls in high-level languages are usually implemented with J or JAL instructions, both of which are J-type instructions. In J-type format, the 26-bit target address shifts left 2 bits and combines with the high-order 4 bits of the current program counter to form a 32-bit or 64-bit absolute address.

Returns, dispatches, and cross-page jumps are usually implemented with the JR or JALR instructions. Both are R-type instructions that take the 32-bit or 64-bit byte address contained in one of the general registers.

For more information, refer to **APPENDIX A MIPS III INSTRUCTION SET DETAILS**.

**(2) Overview of branch instructions**

A branch instruction has a PC-related signed 16-bit offset.

Tables 2-16 through 2-18 show the lists of Jump, Branch, and Expanded ISA instructions, respectively.

**Table 2-16. Jump Instruction**

| Instruction | Format and Description | op | target |
|---|---|---|---|
| Jump | JAL target<br>The contents of 26-bit target address is shifted left by two bits and combined with the high-order four bits of the PC. The program jumps to this calculated address with a delay of one instruction. | | |
| Jump And Link | J target<br>The contents of 26-bit target address is shifted left by two bits and combined with the high-order four bits of the PC. The program jumps to this calculated address with a delay of one instruction. The address of the instruction following the delay slot is stored into r31 (link register). | | |

| Instruction | Format and Description | op | target |
|---|---|---|---|
| Jump And Link Exchange | JALX target<br>The contents of 26-bit target address is shifted left by two bits and combined with the high-order four bits of the PC. The program jumps to this calculated address with a delay of one instruction, and then the ISA mode bit is reversed. The address of the instruction following the delay slot is stored into r31 (link register). | | |

| Instruction | Format and Description | op | rs | rt | rd | sa | funct |
|---|---|---|---|---|---|---|---|
| Jump Register | JR rs<br>The program jumps to the address specified in register rs with a delay of one instruction. | | | | | | |
| Jump And Link Register | JALR rs, rd<br>The program jumps to the address specified in register rs with a delay of one instruction.<br>The address of the instruction following the delay slot is stored into rd. | | | | | | |

There are the following common restrictions for Tables 2-17 and 2-18.

**(3) Branch address**

All branch instruction target addresses are computed by adding the address of the instruction in the delay slot to the 16-bit offset (shifted left by 2 bits and sign-extended to 64 bits). All branches occur with a delay of one instruction.

**(4) Operation when unbranched (Table 2-18)**

If the branch condition does not meet in executing a likely instruction, the instruction in its delay slot is nullified. For all other branch instructions, the instruction in its delay slot is unconditionally executed.

**Remark** The target instruction of the branch is fetched at the EX stage of the branch instruction. Comparison of the operands of the branch instruction and calculation of the target address is performed at phase 2 of the RF stage and phase 1 of the EX stage of the instruction. Branch instructions require one cycle of the branch delay slot defined by the architecture. Jump instructions also require one cycle of delay slot. If the branch condition is not satisfied in a branch likely instruction, the instruction in its delay slot is nullified.

There are special symbols used in the instruction formats of Tables 2-17 through 2-21.

| | |
|---|---|
| REGIMM | : Opcode |
| Sub | : Sub-operation code |
| CO | : Sub-operation identifier |
| BC | : BC sub-operation code |
| br | : Branch condition identifier |
| op | : Operation code |

**Table 2-17. Branch Instructions**

| Instruction | Format and Description | op | rs | rt | offset |
|---|---|---|---|---|---|
| Branch On Equal | BEQ rs, rt, offset<br>If the contents of register rs are equal to that of register rt, the program branches to the target address. | | | | |
| Branch On Not Equal | BNE rs, rt, offset<br>If the contents of register rs are not equal to that of register rt, the program branches to the target address. | | | | |
| Branch On Less Than Or Equal To Zero | BLEZ rs, offset<br>If the contents of register rs are less than or equal to zero, the program branches to the target address. | | | | |
| Branch On Greater Than Zero | BGTZ rs, offset<br>If the contents of register rs are greater than zero, the program branches to the target address. | | | | |

| Instruction | Format and Description | REGIMM | rs | sub | offset |
|---|---|---|---|---|---|
| Branch On Less Than Zero | BLTZ rs, offset<br>If the contents of register rs are less than zero, the program branches to the target address. | | | | |
| Branch On Greater Than Or Equal To Zero | BGEZ rs, offset<br>If the contents of register rs are greater than or equal to zero, the program branches to the target address. | | | | |
| Branch On Less Than Zero And Link | BLTZAL rs, offset<br>The address of the instruction that follows delay slot is stored to register r31 (link register). If the contents of register rs are less than zero, the program branches to the target address. | | | | |
| Branch On Greater Than Or Equal To Zero And Link | BGEZAL rs, offset<br>The address of the instruction that follows delay slot is stored to register r31 (link register). If the contents of register rs are greater than or equal to zero, the program branches to the target address. | | | | |

| Instruction | Format and Description | COP0 | BC | br | offset |
|---|---|---|---|---|---|
| Branch On Coprocessor 0 True | BC0T offset<br>Adds the 16-bit offset (shifted left by two bits and sign extended to 32 bits) to the address of the instruction in the delay slot to calculate the branch target address.<br>If the conditional signal of the coprocessor 0 is true, the program branches to the target address with one-instruction delay. | | | | |
| Branch On Coprocessor 0 False | BC0F offset<br>Adds the 16-bit offset (shifted left by two bits and sign extended to 32 bits) to the address of the instruction in the delay slot to calculate the branch target address.<br>If the conditional signal of the coprocessor 0 is false, the program branches to the target address with one-instruction delay. | | | | |

**Table 2-18. Branch Instructions (Extended ISA)**

| Instruction | Format and Description | op | rs | rt | offset |
|---|---|---|---|---|---|
| Branch On Equal Likely | BEQL rs, rt, offset<br>If the contents of register rs are equal to that of register rt, the program branches to the target address. If the branch condition is not met, the instruction in the delay slot is discarded. | | | | |
| Branch On Not Equal Likely | BNEL rs, rt, offset<br>If the contents of register rs are not equal to that of register rt, the program branches to the target address. If the branch condition is not met, the instruction in the delay slot is discarded. | | | | |
| Branch On Less Than Or Equal To Zero Likely | BLEZL rs, offset<br>If the contents of register rs are less than or equal to zero, the program branches to the target address. If the branch condition is not met, the instruction in the delay slot is discarded. | | | | |
| Branch On Greater Than Zero Likely | BGTZL rs, offset<br>If the contents of register rs are greater than zero, the program branches to the target address. If the branch condition is not met, the instruction in the delay slot is discarded. | | | | |

| Instruction | Format and Description | REGIMM | rs | sub | offset |
|---|---|---|---|---|---|
| Branch On Less Than Zero Likely | BLTZL rs, offset<br>If the contents of register rs are less than zero, the program branches to the target address. If the branch condition is not met, the instruction in the delay slot is discarded. | | | | |
| Branch On Greater Than Or Equal To Zero Likely | BGEZL rs, offset<br>If the contents of register rs are greater than or equal to zero, the program branches to the target address. If the branch condition is not met, the instruction in the delay slot is discarded. | | | | |
| Branch On Less Than Zero And Link Likely | BLTZALL rs, offset<br>The address of the instruction that follows delay slot is stored to register r31 (link register).<br>If the contents of register rs are less than zero, the program branches to the target address. If the branch condition is not met, the instruction in the delay slot is discarded. | | | | |
| Branch On Greater Than Or Equal To Zero And Link Likely | BGEZALL rs, offset<br>The address of the instruction that follows delay slot is stored to register r31 (link register).<br>If the contents of register rs are greater than or equal to zero, the program branches to the target address. If the branch condition is not met, the instruction in the delay slot is discarded. | | | | |

| Instruction | Format and Description | COP0 | BC | br | offset |
|---|---|---|---|---|---|
| Branch On Coprocessor 0 True Likely | BC0TL offset<br>Adds the 16-bit offset (shifted left by two bits and sign extended to 32 bits) to the address of the instruction in the delay slot to calculate the branch target address.<br>If the conditional signal of the coprocessor 0 is true, the program branches to the target address with one-instruction delay.<br>If the branch condition is not met, the instruction in the delay slot is discarded. | | | | |
| Branch On Coprocessor 0 False Likely | BC0FL offset<br>Adds the 16-bit offset (shifted left by two bits and sign extended to 32 bits) to the address of the instruction in the delay slot to calculate the branch target address.<br>If the conditional signal of the coprocessor 0 is false, the program branches to the target address with one-instruction delay.<br>If the branch condition is not met, the instruction in the delay slot is discarded. | | | | |

#### 2.2.2.4 Special instructions

Special instructions generate software exceptions. Their formats are R-type (Syscall, Break). The Trap instruction is available only for the VR4000 Series. All the other instructions are available for all VR Series.

**Table 2-19. Special Instructions**

| Instruction | Format and Description: SPECIAL \| rs \| rt \| rd \| sa \| funct |
|---|---|
| Synchronize | SYNC<br>Completes the load/store instruction executing in the current pipeline before the next load/store instruction starts execution. |
| System Call | SYSCALL<br>Generates a system call exception, and then transits control to the exception handling program. |
| Breakpoint | BREAK<br>Generates a break point exception, and then transits control to the exception handling program. |

**Table 2-20. Special Instructions (Expanded ISA) (1/2)**

| Instruction | Format and Description: SPECIAL \| rs \| rt \| rd \| sa \| funct |
|---|---|
| Trap If Greater Than Or Equal | TGE rs, rt<br>The contents of register rs are compared with that of register rt, treating both operands as signed integers. If the contents of register rs are greater than or equal to that of register rt, an exception occurs. |
| Trap If Greater Than Or Equal Unsigned | TGEU rs, rt<br>The contents of register rs are compared with that of register rt, treating both operands as unsigned integers. If the contents of register rs are greater than or equal to that of register rt, an exception occurs. |
| Trap If Less Than | TLT rs, rt<br>The contents of register rs are compared with that of register rt, treating both operands as signed integers. If the contents of register rs are less than that of register rt, an exception occurs. |
| Trap If Less Than Unsigned | TLTU rs, rt<br>The contents of register rs are compared with that of register rt, treating both operands as unsigned integers. If the contents of register rs are less than that of register rt, an exception occurs. |
| Trap If Equal | TEQ rs, rt<br>If the contents of registers rs and rt are equal, an exception occurs. |
| Trap If Not Equal | TNE rs, rt<br>If the contents of registers rs and rt are not equal, an exception occurs. |

**Table 2-20. Special Instructions (Expanded ISA) (2/2)**

| Instruction | Format and Description | REGIMM | rs | sub | immediate |
|---|---|---|---|---|---|
| Trap If Greater Than Or Equal Immediate | TGEI rs, immediate<br>The contents of register rs are compared with 16-bit sign-extended immediate data, treating both operands as signed integers. If the contents of register rs are greater than or equal to 16-bit sign-extended immediate data, an exception occurs. | | | | |
| Trap If Greater Than Or Equal Immediate Unsigned | TGEIU rs, immediate<br>The contents of register rs are compared with 16-bit zero-extended immediate data, treating both operands as unsigned integers. If the contents of register rs are greater than or equal to 16-bit sign-extended immediate data, an exception occurs. | | | | |
| Trap If Less Than Immediate | TLTI rs, immediate<br>The contents of register rs are compared with 16-bit sign-extended immediate data, treating both operands as signed integers. If the contents of register rs are less than 16-bit sign-extended immediate data, an exception occurs. | | | | |
| Trap If Less Than Immediate Unsigned | TLTIU rs, immediate<br>The contents of register rs are compared with 16-bit zero-extended immediate data, treating both operands as unsigned integers. If the contents of register rs are less than 16-bit sign-extended immediate data, an exception occurs. | | | | |
| Trap If Equal Immediate | TEQI rs, immediate<br>If the contents of register rs and immediate data are equal, an exception occurs. | | | | |
| Trap If Not Equal Immediate | TNEI rs, immediate<br>If the contents of register rs and immediate data are not equal, an exception occurs. | | | | |

### 2.2.2.5 System control coprocessor (CP0) instructions

System control coprocessor (CP0) instructions perform operations specifically on the CP0 registers to manipulate the memory management and exception handling facilities of the processor.

**Table 2-21. System Control Coprocessor (CP0) Instructions (1/2)**

| Instruction | Format and Description | COP0 | sub | rt | rd | 0 |
|---|---|---|---|---|---|---|
| Move To System Control Coprocessor | MTC0 rt, rd<br>The word data of general register rt in the CPU are loaded into general register rd in the CP0. | | | | | |
| Move From System Control Coprocessor | MFC0 rt, rd<br>The word data of general register rd in the CP0 are loaded into general register rt in the CPU. | | | | | |
| Doubleword Move To System Control Coprocessor 0 | DMTC0 rt, rd<br>The doubleword data of general register rt in the CPU are loaded into general register rd in the CP0. | | | | | |
| Doubleword Move From System Control Coprocessor 0 | DMFC0 rt, rd<br>The doubleword data of general register rd in the CP0 are loaded into general register rt in the CPU. | | | | | |

**Table 2-21. System Control Coprocessor (CP0) Instructions (2/2)**

| Instruction | Format and Description | COP0 | CO | funct |
|---|---|---|---|---|
| Read Indexed TLB Entry | TLBR<br>The TLB entry indexed by the index register is loaded into the entryHi, entryLo0, entryLo1, or page mask register. | | | |
| Write Indexed TLB Entry | TLBWI<br>The contents of the entryHi, entryLo0, entryLo1, or page mask register are loaded into the TLB entry indexed by the index register. | | | |
| Write Random TLB Entry | TLBWR<br>The contents of the entryHi, entryLo0, entryLo1, or page mask register are loaded into the TLB entry indexed by the random register. | | | |
| Probe TLB For Matching Entry | TLBP<br>The address of the TLB entry that matches with the contents of entryHi register is loaded into the index register. | | | |
| Return From Exception | ERET<br>The program returns from exception, interrupt, or error trap. | | | |

| Instruction | Format and Description | COP0 | CO | funct |
|---|---|---|---|---|
| STANDBY | STANDBY<br>The processor's operating mode is transited from fullspeed mode to standby mode. | | | |
| SUSPEND | SUSPEND<br>The processor's operating mode is transited from fullspeed mode to suspend mode. | | | |
| HIBERNATE | HIBERNATE<br>The processor's operating mode is transited from fullspeed mode to hibernate mode. | | | |

| Instruction | Format and Description | CACHE | base | op | offset |
|---|---|---|---|---|---|
| Cache Operation | Cache op, offset (base)<br>The 16-bit offset is sign extended to 32 bits and added to the contents of the register case, to form virtual address. This virtual address is translated to physical address with TLB. For this physical address, cache operation that is indicated by 5-bit sub-opcode is performed. | | | | |

## 2.3 MIPS16 Instruction Set

If the MIPS16 ASE (Application-Specific Extension), which is an expanded function for MIPS ISA (Instruction Set Architecture), is used, system costs can be considerably reduced by lowering the memory capacity requirement of embedded hardware. MIPS16 is an instruction set that uses the 16-bit instruction length, and is compatible with MIPS I, II, III, IV, and V[Note] instruction sets in any combination. Moreover, 32-bit instruction length binary data can be executed with the VR4120A.

**Note** The VR4100 Series™ currently supports the MIPS I, II, and III instruction sets.

### 2.3.1 Features

- 16-bit length instruction format
- Reduces memory capacity requirements to lower overall system cost
- MIPS16 instructions can be used with MIPS instruction binary
- Compatibility with MIPS I, II, III, IV, and V instruction sets
- Used with switching between MIPS16 instruction length mode and 32-bit MIPS instruction length mode.
- Supports 8-bit, 16-bit, 32-bit, and 64-bit data formats
- Provides 8 general registers and special registers
- Improved code generation efficiency using special 16-bit dedicated instructions

### 2.3.2 Register set

Tables 2-22 and 2-23 show the MIPS16 register sets. These register sets form part of the register sets that can be accessed in 32-bit instruction length mode. MIPS16 ASE can directly access 8 of the 32 registers that can be used in the 32-bit instruction length mode.

In addition to these 8 general registers, the special instructions of MIPS16 ASE reference the stack pointer register (sp), return address register (ra), condition code register (t8), and program counter (pc). sp and ra are mapped by fixing to the general registers in the 32-bit instruction length mode.

MIPS16 has 2 move instructions that are used in addressing 32 general registers.

**Table 2-22. General Registers**

| MIPS16 Register Encoding | 32-Bit MIPS Register Encoding | Symbol | Comment |
|---|---|---|---|
| 0 | 16 | s0 | General register |
| 1 | 17 | s1 | General register |
| 2 | 2 | v0 | General register |
| 3 | 3 | v1 | General register |
| 4 | 4 | a0 | General register |
| 5 | 5 | a1 | General register |
| 6 | 6 | a2 | General register |
| 7 | 7 | a3 | General register |
| N/A | 24 | t8 | MIPS16 condition code register. BTEQZ, BTNEZ, CMP, CMPI, SLT, SLTU, SLTI, and SLTIU instructions are implicitly referenced. |
| N/A | 29 | sp | Stack pointer register |
| N/A | 31 | ra | Return address register |

**Remarks 1.** The symbols are the general assembler symbols.

**2.** The MIPS register encoding numbers 0 to 7 correspond to the MIPS16 binary encoding of the registers, and are used to show the relationship between this encoding and the MIPS registers. The numbers 0 to 7 are not used to reference registers, except within binary MIPS16 instructions. Registers are referenced from the assembler using the MIPS name ($16, $17, $2, etc.) or the symbol name (s0, s1, v0, etc.). For example, when register number 17 is accessed with the register file, the programmer references either $17 or s1 even if the MIPS16 encoding of this register is 001.

**3.** The general registers not shown in this table cannot be accessed with a MIPS16 instruction set other than the Move instruction. The Move instruction of MIPS16 can access all 32 general registers.

**4.** To reference the MIPS16 condition code registers with this manual, either T, t8, or $24 has to be used, depending on the case. These three names reference the same physical register.

**Table 2-23. Special Registers**

| Symbol | Description |
|---|---|
| PC | Program counter. The PC-relative Add instruction and Load instruction can access this register. |
| HI | The upper word of the multiply or divide result is inserted |
| LO | The lower word of the multiply or divide result is inserted |

### 2.3.3 ISA mode

MIPS16 ASE supports procedure calling, and returns from the MIPS16 instruction length mode or the 32-bit MIPS instruction length mode to the MIPS16 instruction length mode or the 32-bit MIPS instruction length mode.

- The JAL instruction supports calling to the same ISA.
- The JALX instruction supports calling that inverses ISA.
- The JALR instruction supports calling to either ISA.
- The JR instruction supports also returning to either ISA.

MIPS16 ASE also supports a return operation from exception processing.

- The ERET instruction, which is defined only in 32-bit instruction length mode, supports returning to ISA when an exception has not occurred.

The ISA mode bit defines the instruction length mode to be executed. If the ISA mode bit is 0, the processor executes only 32-bit MIPS instructions. If the ISA mode bit is 1, the processor executes only MIPS16 instructions.

#### 2.3.3.1 Changing ISA mode bit by software

Only the JALX, JR, and JALR instructions change the ISA mode bit between the MIPS16 instruction mode and the 32-bit instruction length mode. The ISA mode bit cannot be directly overwritten by software. The JALX changes the ISA mode bit to select another ISA mode. The JR instruction and JALR instruction load the ISA mode bit from bit 0 of the general register that holds the target address. Bit 0 is not a part of the target address. Bit 0 of the target address is always 0, and no address exception is generated.

Moreover, the JAL, JALR, and JALX instructions save the ISA mode bit to bit 0 of the general register that acquires the return address. The contents of this general register are later used by the JR and JALR instruction for return and restoration of the ISA mode.

#### 2.3.3.2 Changing ISA mode bit by exception

Even if an exception occurs, the ISA mode changes. When an exception occurs, the ISA mode bit is cleared to 0 so that the exception is serviced with 32-bit code. Then the ISA mode status before the exception occurred is saved to the least significant bit of the EPC register or the error EPC register. During return from an exception, the ISA mode before the exception occurred is returned to by executing the JR or ERET instruction with the contents of this register. Moreover, the ISA mode bit is cleared to 0 after cold reset and soft reset of the CPU core, and the 32-bit instruction length mode returns to its initial state.

#### 2.3.3.3 Enabling change ISA mode bit

Changing the ISA mode bit is valid only MIPS16EN is set to active when the RTCRST is selected, and the MIPS16 instruction mode is enabled. The operation of the JALX, JALR, JR, and ERET instructions in the 32-bit instruction mode, differs depending on whether the MIPS16 instruction mode is enabled or prohibited. If the MIPS16 instruction mode is prohibited, the JALX instruction generates a reserved instruction exception. The JR and JALR instructions generate an address exception when bit 0 of the source register is 1. The ERET instruction generates an address exception when bit 0 of the EPC or error EPC register is 1. If the MIPS16 instruction mode is enabled, the JALX instruction executes JAL, and the ISA mode bit is inverted. The JR and JALR instructions load the ISA mode from bit 0 of the source register. The ERET instruction loads the ISA mode from bit 0 of the EPC or error EPC register. Bit 0 of the target address is always 0, and no address exception is generated even when bit 0 of the source register is 1.

### 2.3.4 Types of Instructions

This section describes the different types of instructions, and indicates the MIPS16 instructions included in each group.

Instructions are divided into the following types.

| | |
|---|---|
| Load and Store instructions | :Move data between memory and the general registers. |
| Computational instructions | :Perform arithmetic operations, logical operations, and shift operations on values in registers. |
| Jump and Branch instructions | :Change the control flow of a program. |
| Special instructions | :Break instructions and Extend instructions. Break transfers control to an exception handler. Extend enlarges the immediate field of the next instruction. Instructions that can be extended with Extend are indicated as **Note 1** in **Table 2-24 MIPS16 Instruction Set Outline**. |

**Table 2-24. MIPS16 Instruction Set Outline (1/3)**

| Op | Description |
|---|---|
| Load and Store instructions | |
| LB Note 1 | Load Byte |
| LBU Note 1 | Load Byte Unsigned |
| LH Note 1 | Load Halfword |
| LHU Note 1 | Load Halfword Unsigned |
| LW Note 1 | Load Word |
| LWU Notes 1, 2 | Load Word Unsigned |
| LD Notes 1, 2 | Load Doubleword |
| SB Note 1 | Store Byte |
| SH Note 1 | Store Halfword |
| SW Note 1 | Store Word |
| SD Notes 1, 2 | Store Doubleword |
| Arithmetic instructions: ALU immediate instructions | |
| LI Note 1 | Load Immediate |
| ADDIU Note 1 | Add Immediate Unsigned |
| DADDIU Notes 1, 2 | Doubleword Add Immediate Unsigned |
| SLTI Note 1 | Set on Less Than Immediate |
| SLTIU Note 1 | Set on Less Than Immediate Unsigned |
| CMPI Note 1 | Compare Immediate |

**Notes 1.** Extendable instruction. For details, see **APPENDIX B MIPS16 INSTRUCTION SET FORMAT**.

**2.** Can be used in 64-bit mode and 32-bit kernel mode.

**Table 2-24. MIPS16 Instruction Set Outline (2/3)**

| Op | Description |
|---|---|
| Arithmetic instructions: 2/3 operand register instructions | |
| ADDU | Add Unsigned |
| SUBU | Subtract Unsigned |
| DADDU Note 2 | Doubleword Add Unsigned |
| DSUBU Note 2 | Doubleword Subtract Unsigned |
| SLT | Set on Less Than |
| SLTU | Set on Less Than Unsigned |
| CMP | Compare |
| NEG | Negate |
| AND | AND |
| OR | OR |
| XOR | Exclusive OR |
| NOT | Not |
| MOVE | Move |
| Special instructions | |
| EXTEND | Extend |
| BREAK | Breakpoint |
| Multiply/Divide instructions | |
| MULT | Multiply |
| MULTU | Multiply Unsigned |
| DIV | Divide |
| DIVU | Divide Unsigned |
| MFHI | Move From HI |
| MFLO | Move From LO |
| DMULT Note 2 | Doubleword Multiply |
| DMULTU Note 2 | Doubleword Multiply Unsigned |
| DDIV Note 2 | Doubleword Divide |
| DDIVU Note 2 | Doubleword Divide Unsigned |
| Jump/Branch instructions | |
| JAL | Jump and Link |
| JALX | Jump and Link Exchange |
| JR | Jump Register |
| JALR | Jump and Link Register |
| BEQZ Note 1 | Branch on Equal to Zero |
| BNEZ Note 1 | Branch on Not Equal to Zero |
| BTEQZ Note 1 | Branch on T Equal to Zero |
| BTNEZ Note 1 | Branch on T Not Equal to Zero |
| B Note 1 | Branch Unconditional |

**Notes 1.** Extendable instruction. For details, see **APPENDIX B MIPS16 INSTRUCTION SET FORMAT**.

**2.** Can be used in 64-bit mode and 32-bit kernel mode.

**Table 2-24. MIPS16 Instruction Set Outline (3/3)**

| Op | Description |
|---|---|
| Shift instructions | |
| SLL Note 1 | Shift Left Logical |
| SRL Note 1 | Shift Right Logical |
| SRA Note 1 | Shift Right Arithmetic |
| SLLV | Shift Left Logical Variable |
| SRLV | Shift Right Logical Variable |
| SRAV | Shift Right Arithmetic Variable |
| DSLL Notes 1, 2 | Doubleword Shift Left Logical |
| DSRL Notes 1, 2 | Doubleword Shift Right Logical |
| DSRA Notes 1, 2 | Doubleword Shift Right Arithmetic |
| DSLLV Note 2 | Doubleword Shift Left Logical Variable |
| DSRLV Note 2 | Doubleword Shift Right Logical Variable |
| DSRAV Note 2 | Doubleword Shift Right Arithmetic Variable |

**Notes 1.** Extendable instruction. For details, see **APPENDIX B MIPS16 INSTRUCTION SET FORMAT**.

**2.** Can be used in 64-bit mode and 32-bit kernel mode.

### 2.3.5 Instruction format

The MIPS16 instruction set has a length of 16 bits and is located at the halfword boundary. One part of Jump instructions and instructions for which the Extend instruction extends immediate become 32 bits in length, but crossing the word boundary does not represent a problem.

The instruction format is shown below. Variable subfields are indicated with lower case letters (rx, ry, rz, immediate, etc.).

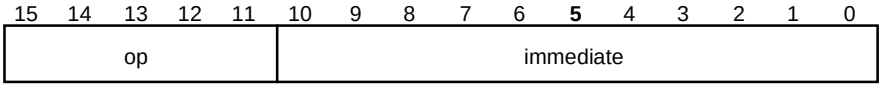
In the case of special functions, constants are input to the two instruction subfields op and funct. These values are indicated by upper case mnemonics. For example, in the case of the Load Byte instruction, op is LB, and in the case of the Add instruction, op is SPECIAL, and function is ADD.

The constants of the fields used in the instruction formats are shown below.

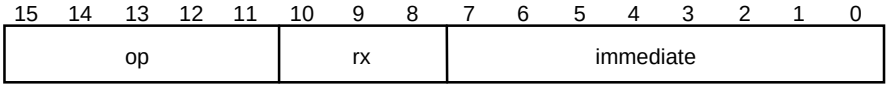
**Table 2-25. Field Definition**

| Field | Definition |
|---|---|
| op | 5-bit major operation code |
| rx | 3-bit source/destination register specification |
| ry | 3-bit source/destination register specification |
| immediate or imm | 4-bit, 5-bit, 8-bit, or 11-bit immediate value, branch displacement, or address displacement |
| rz | 3-bit source/destination register specification |
| Funct or F | Function field |

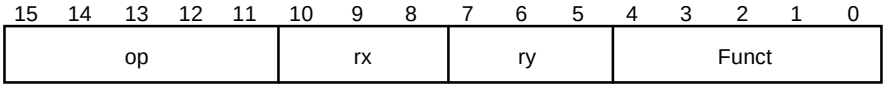
**I-type (immediate) instruction format**



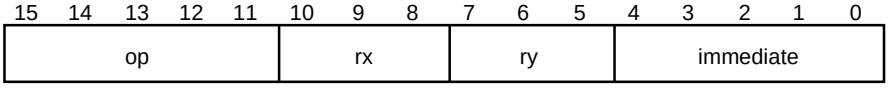

**RI-type instruction format**



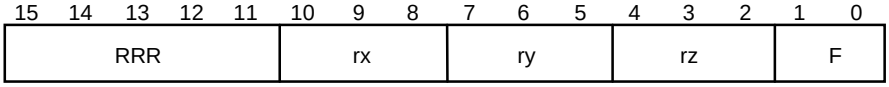

**RR-type instruction format**



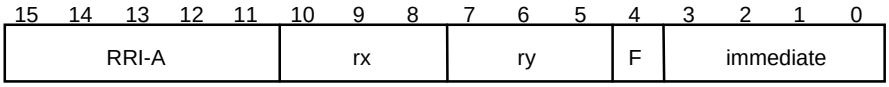

**RRI-type instruction format**



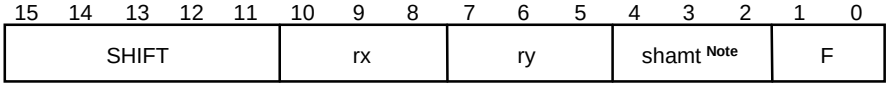

**RRR-type instruction format**




**RRI-A type instruction format**

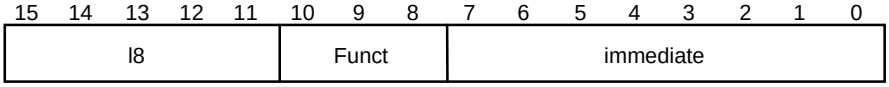



**SHIFT instruction format**

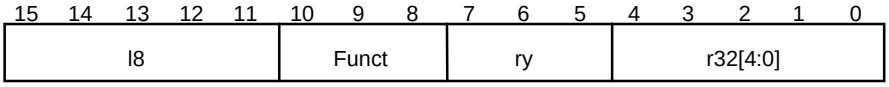



**Note** The 3-bit shamt field can encode shift count numbers from 0 to 7. 0-bit shift (NOP) cannot be executed. 0 is regarded as shift count 8.

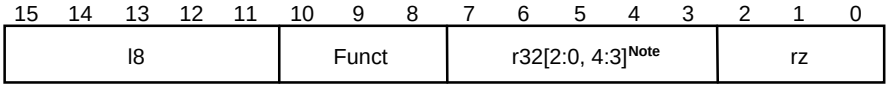
**I8-type instruction format**



**I8_MOVR32 instruction format (used only with MOVR32 instruction)**

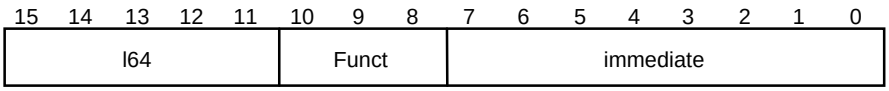



**I8_MOV32R instruction format (used only with MOV32R instruction)**

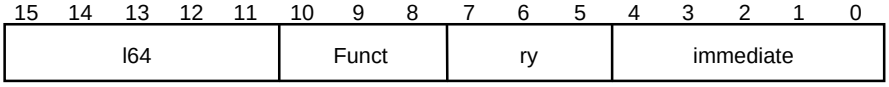



**Note** The r32 field uses special bit encoding. For example, encoding of $7 (00111) is 11100 in the r32 field.

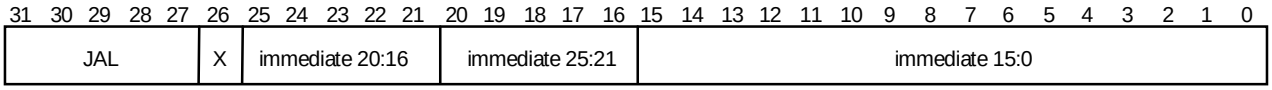
**I64-type instruction format**




**RI64-type instruction format**

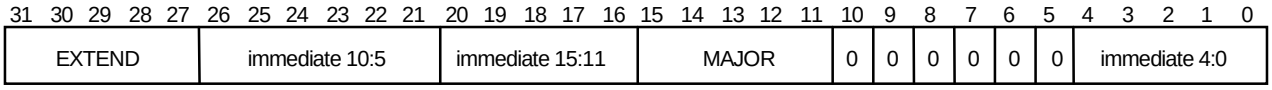



**JAL and JALX instruction format**

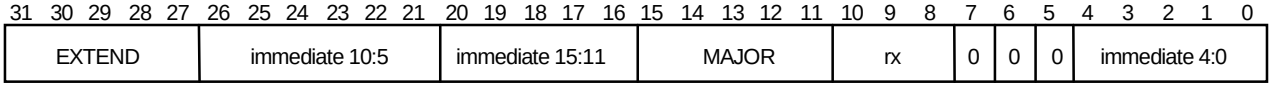



**Remark** JAL in case of X = 0 instruction
JALX in case of X = 1 instruction

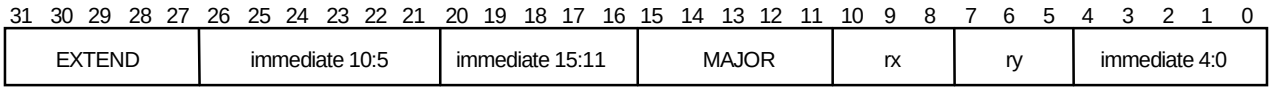
**EXT-I instruction format**



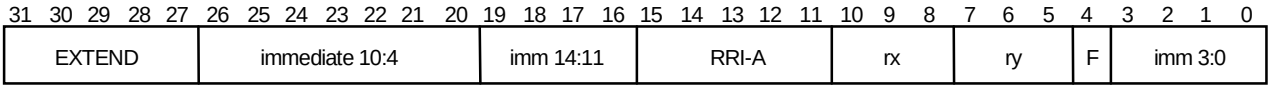

**EXT-RI instruction format**



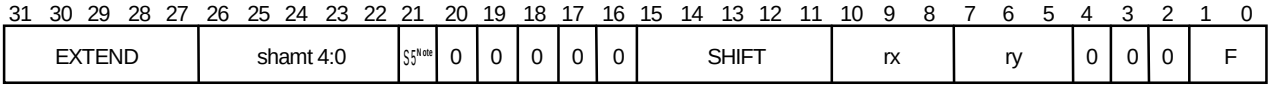

**EXT-RRI instruction format**



**EXT-RRI-A instruction format**

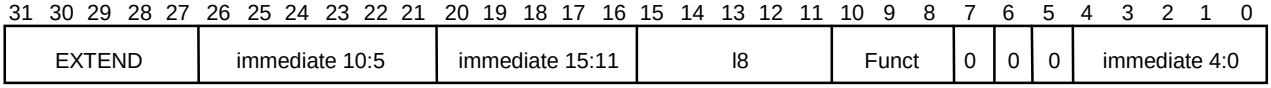



**EXT-SHIFT instruction format**

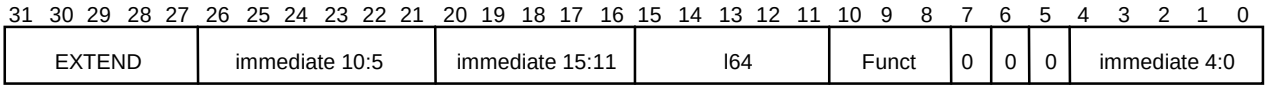



**Note** Only in the case of DSLL, the S5 bit is the most significant bit of the 6-bit shift count field (shamt). In the case of all 32-bit extended shifts, S5 must be 0. For a normal shift instruction, the display of shift count 0 is considered as shift count 8, but the extended shift instruction does not perform such mapping changes. Therefore, 0-bit shift using the extended format is possible.

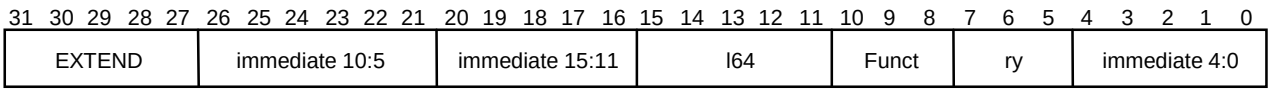
**EXT-I8 instruction format**



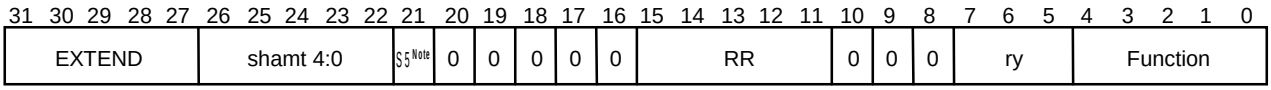

**EXT-I64 instruction format**




**EXT-RI64 instruction format**




**EXT-SHIFT64 instruction format**




**Note** The S5 bit is the most significant bit of the 6-bit shift count field (shamt). In the case of a normal shift instruction, the display of shift count 0 is considered as shift count 8, but the extended shift instruction does not perform such mapping changes.
Therefore, 0-bit shift using the extended format is possible.

### 2.3.6 MIPS16 operation code bit encoding

This section describes encoding for major operation code and minor operation code. Table 2-26 shows bit encoding of the MIPS16 major operation code. Tables 2-27 to 2-32 show bit encoding of the minor operation code. The italic operation codes in the tables are instructions for the extended ISA.

**Table 2-26. Bit Encoding of Major Operation Code (op)**

| Instruction Bits | Instruction Bits [13:11] | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| [15:14] | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| 00 | addiusp Note 1 | addiupc Note 2 | b | jal(x) Note 3 | beqz | bnez | SHIFT | ***ld*** |
| 01 | RRI-A | addiu8 Note 4 | slti | sltiu | I8 | li | cmpi | ***sd*** |
| 10 | lb | lh | lwsp | lw | lbu | lhu | lwpc | ***lwu*** |
| 11 | sb | sh | swsp | sw | RRR | RR | extend | ***I64*** |

**Notes 1.** addiusp : addiu rx, sp, immediate
**2.** addiupc : addiu rx, pc, immediate
**3.** jal(x) : jal instruction and jalx instruction
**4.** addiu8 : aadiu rx, immediate

**Table 2-27. RR Minor Operation Code (RR-Type Instruction)**

| Instruction Bits | Instruction Bits [2:0] | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| [4:3] | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| 00 | j(al)r Note 1 | * | slt | sltu | sllv | break | srlv | srav |
| 01 | ***dsrl*** Note 2 | $\phi$ | cmp | neg | and | or | xor | not |
| 10 | mfhi | * | mflo | ***dsra*** Note 2 | ***dsllv*** | * | ***dsrlv*** | ***dsrav*** |
| 11 | mult | multu | div | divu | ***dmult*** | ***dmultu*** | ***ddiv*** | ***ddivu*** |

**Notes 1.** j(al)r: jr rx instruction (ry = 000)
jr ra instruction (ry = 001, rx = 000)
jalr ra, rx instruction (ry = 010)
**2.** dsrl and dsra use the rx register field to encode the shift count (8-digit shift for 0). In the case of the extended version of these two instructions, the EXT-SHIFT64 format is used. Only these two RR instructions can be extended.

**Remark** The symbols in the figures have the following meaning.
* : Execution of operation code with an asterisk on the current VR4120A causes a reserved instruction exception to be generated. This code is reserved for future extension.
$\phi$ : Operation code with $\phi$ is invalid, but no reserved instruction exception is generated in the VR4120A.

**Table 2-28. RRR Minor Operation Code (RRR-Type Instruction)**

| Instruction Bits [1:0] | | | |
|---|---|---|---|
| 00 | 01 | 10 | 11 |
| ***daddu*** | addu | ***dsubu*** | subu |

**Table 2-29. RRRI-A Minor Operation Code (RRI-Type ADD Instruction)**

| Instruction Bit [4] | |
|---|---|
| 0 | 1 |
| addiu Note 1 | ***daddiu*** Note 2 |

**Notes 1.** addiu : addiu ry, rx, immediate
**2.** daddiu : daddiu ry, rx immediate

**Table 2-30. SHIFT Minor Operation Code (SHIFT-Type Instruction)**

| Instruction Bits [1:0] | | | |
|---|---|---|---|
| 00 | 01 | 10 | 11 |
| sll | ***dsll*** | srl | sra |

**Table 2-31. I8 Minor Operation Code (I8-Type Instruction)**

| Instruction Bits [10:8] | | | | | | | |
|---|---|---|---|---|---|---|---|
| 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| bteqz | btnez | swrasp Note 1 | adjsp Note 2 | * | mov32r Note 3 | * | movr32 Note 4 |

**Notes 1.** swrasp : sw ra, immediate(sp)
**2.** adjsp : addiu sp, immediate
**3.** mov32r : move r32, rz
**4.** movr32 : move ry, r32

**Remark** The symbols used in the figures have the following meaning.
* : Execution of operation code with an asterisk on the current VR4120A causes a reserved instruction exception to be generated. This code is reserved for future extension.

**Table 2-32. I64 Minor Operation Code (64-bit Only, I64-Type Instruction)**

| Instruction Bits [10:8] | | | | | | | |
|---|---|---|---|---|---|---|---|
| 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| ***ldsp*** Note 1 | ***sdsp*** Note 2 | ***sdrasp*** Note 3 | ***dadjsp*** Note 4 | ***ldpc*** Note 5 | ***daddiu5*** Note 6 | ***dadiupc*** Note 7 | ***dadiusp*** Note 8 |

**Notes 1.** ldsp : ld ry, immediate
**2.** sdsp : sd ry, immediate
**3.** sdrasp : sd ra, immediate
**4.** dadjsp : daddiu sp, immediate
**5.** ldpc : ld ry, immediate
**6.** daddiu5 : daddiu ry, immediate
**7.** dadiupc : daddiu ry, pc, immediate
**8.** dadiusp : daddiu ry, sp, immediate

### 2.3.7 Outline of instructions

This section describes the assembler syntax and defines each instruction. Instructions can be divided into the following four types.

- Load and Store instructions
- Computational instructions
- Jump and Branch instructions
- Special instructions

#### 2.3.7.1 PC-relative instructions

PC-relative instructions are the instruction format first defined among the MIPS16 instruction set. MIPS16 supports both extension and non-extension through the Extend instruction for four PC-relative instructions.

| | |
|---|---|
| Load Word | LW rx, offset (pc) |
| Load Doubleword | LD ry, offset (pc) |
| Add Immediate Unsigned | ADDIU rx, pc, immediate |
| Doubleword Add Immediate Unsigned | DADDIU ry, pc, immediate |

All these instructions calculate the PC value of a PC-relative instruction or the PC value of the instruction immediately preceding as the base address. The address calculation base using various function combinations is shown next.

**Table 2-33. Base PC Address Setting**

| Instruction | Base PC value |
|---|---|
| Non-extension PC-relative instructions not located in Jump delay slot | PC of instruction |
| Extension PC-relative instruction | PC of Extend instruction |
| Non-extension PC-relative instruction in Jump delay slot of JR or JALR | PC of JR instruction or JALR instruction |
| Non-extension PC-relative instruction in Jump delay slot of JAL or JALX | PC of initial halfword of JAL or JALX[Note] |

**Note** Because the JAL and JALX instruction length is 32 bits.

The PC value used as the base for address calculation for the PC-relative instruction outlines shown in tables 2-35 and 2-36 is called base PC value. The base PC value is defined so as to be equivalent to the exception program counter (EPC) value related to the PC-relative instruction.

#### 2.3.7.2 Extend instruction

The Extend instruction can extend the immediate fields of MIPS16 instructions, which have fewer immediate fields than equivalent 32-bit MIPS instructions. The Extend instruction must always precede (by one instruction) the instruction whose immediate field you want to extend. Every extended instruction consumes four bytes in program memory instead of two bytes (two bytes for Extend and two bytes for the instruction being extended), and it can cross a word boundary.

For example, the MIPS16 instruction contains a five-bit immediate.

LW ry, offset (rx)

The immediate expands to 16 bits (000000000||offset||00) before execution in the pipeline. This allows 32 different offset values of 0, 4, 8, and up through 124. Once extended, this instruction can hold any of the normal 65,536 values in the range –32,768 through 32,767.

Shift instructions are extended to 5-bit unsigned immediate values. All other immediate instructions expand to either signed or unsigned 16-bit immediate values. The only exceptions are which can be extended only to a 15-bit signed immediate.

ADDIU ry, rx, immediate
DADDIU ry, rx, immediate

Extended instructions should not be placed in jump delay slots. Otherwise, the results are unpredictable because the pipeline would attempt to execute one half the instruction.

Table 2-34 lists the MIPS16 extendable instructions, the size of their immediate, and how much each immediate can be extended when preceded with the Extend instruction.

For the instruction format of the Extend instruction, see **2.3.5 Instruction format**.

**Table 2-34. Extendable MIPS16 Instructions**

| MIPS16 Instruction | MIPS16 Immediate | Instruction Format | Extended Immediate | Instruction Format |
|---|---|---|---|---|
| Load Byte | 5 | RRI | 16 | EXT-RRI |
| Load Byte Unsigned | 5 | RRI | 16 | EXT-RRI |
| Load Halfword | 5 | RRI | 16 | EXT-RRI |
| Load Halfword Unsigned | 5 | RRI | 16 | EXT-RRI |
| Load Word | 5<br>8 | RRI<br>RI | 16<br>16 | EXT-RRI<br>EXT-RI |
| Load Word Unsigned | 5 | RRI | 16 | EXT-RRI |
| Load Doubleword | 5 | RRI | 16 | EXT-RRI |
| Store Byte | 5 | RRI | 16 | EXT-RRI |
| Store Halfword | 5 | RRI | 16 | EXT-RRI |
| Store Word | 5 (Other)<br>8 (SW rx, offset(sp))<br>8 (SW ra, offset(sp)) | RRI<br>RI<br>I8 | 16<br>16<br>16 | EXT-RRI<br>EXT-RI<br>EXT-I8 |
| Store Doubleword | 5 (SD ry, offset(rx))<br>8 (Other) | RRI<br>I64 | 16<br>16 | EXT-RRI<br>EXT-I64 |
| Load Immediate | 8 | RI | 16 | EXT-RI |
| Add Immediate Unsigned | 4 (ADDIU ry, rx, imm)<br>8 (ADDIU sp, imm)<br>8 (Other) | RRI-A<br>I8<br>RI | 15<br>16<br>16 | EXT-RRI-A<br>EXT-I8<br>EXT-RI |
| Doubleword Add Immediate Unsigned | 4 (DADDIU ry, rx, imm)<br>5 (DADDIU ry, pc, imm)<br>8 (Other) | RRI-A<br>RI64<br>I64 | 15<br>16<br>16 | EXT-RRI-A<br>EXT-RI64<br>EXT-I64 |
| Set on Less Than Immediate | 8 | RI | 16 | EXT-RI |
| Set on Less Than Immediate Unsigned | 8 | RI | 16 | EXT-RI |
| Compare Immediate | 8 | RI | 16 | EXT-RI |
| Shift Left Logical | 3 | SHIFT | 5 | EXT-SHIFT |
| Shift Right Logical | 3 | SHIFT | 5 | EXT-SHIFT |
| Shift Right Arithmetic | 3 | SHIFT | 5 | EXT-SHIFT |
| Doubleword Shift Left Logical | 3 | SHIFT | 6 | EXT-SHIFT |
| Doubleword Shift Right Logical | 3 | RR | 6 | EXT- SHIFT64 |
| Doubleword Shift Right Arithmetic | 3 | RR | 6 | EXT- SHIFT64 |
| Branch on Equal to Zero | 8 | RI | 16 | EXT-RI |
| Branch on Not Equal to Zero | 8 | RI | 16 | EXT-RI |
| Branch on T Equal to Zero | 8 | I8 | 16 | EXT-I8 |
| Branch on T Not Equal to Zero | 8 | I8 | 16 | EXT-I8 |
| Branch Unconditional | 11 | I | 16 | EXT-I |

#### 2.3.7.3 Delay slots

MIPS16 instructions normally execute in one cycle. However, some instructions have special requirements that must be met to assure optimum instruction flow. The instructions include all Load, Branch, and Multiply/Divide instructions.

**(1) Load delay slots**

MIPS16 operates with delayed loads. This is similar to the method used by 32-bit MIPS instruction sets. If another instruction references the load destination register before the load operation is completed, one cycle occurs automatically. To assure the best performance, the compiler should always schedule load delay slots as early as possible.

**(2) Branch delay slots not supported**

Unlike for 32-bit MIPS instructions, there are no branch delay slots for branch instructions in MIPS16. If a branch is taken, the instruction that immediately follows the branch (instruction corresponding to 32-bit MIPS delay slot) is cancelled. There are no restrictions on the instruction that follows a branch instruction, and such instruction is executed only when a branch is not taken. Branches, jumps, and extended instructions are permitted in the instruction slot after a branch.

**(3) Jump delay slots**

With MIPS16, there is a delay of one cycle after each jump instruction. The processor executes any instruction in the jump delay slot before it executes the jump target instruction. Two restrictions apply to any instruction placed in the jump delay slot:

1. Do not specify a branch or jump in the delay slot.
2. Do not specify an extended instruction (32-bits) in the delay slot. Doing so will make the results unpredictable.

**(4) Multiply and divide scheduling**

Multiply and divide latency depends on the hardware implementation. If an MFLO or MFHI instruction references the Multiply or Divide result registers before the result is ready, the pipeline stalls until the operation is complete and the result is available. However, to assure the best performance, the compiler should always schedule Multiply and Divide instructions as early as possible.

MIPS16 requires that all MFHI and MFLO instructions be followed by two instructions that do not write to the HI or LO registers. Otherwise, the data read by MFLO or MFHI will be undefined. The Extend instruction is counted singly as one instruction.

### 2.3.7.4 Instruction details

#### (1) Load and store instructions

Load and Store instructions move data between memory and the general registers. The only addressing mode that is supported is the mode for adding immediate offset to the base register.

**Table 2-35. Load and Store Instructions (1/3)**

| Instruction | Format and Description |
|---|---|
| Load Byte | LB ry, offset (rx)<br>The 5-bit immediate is zero extended and then added to the contents of general register rx to form the virtual address. The bytes of the memory location specified by the address are sign extended and loaded into general register ry. |
| Load Byte Unsigned | LBU ry, offset (rx)<br>The 5-bit immediate is zero extended and then added to the contents of general register rx to form the virtual address. The bytes of the memory location specified by the address are zero extended and loaded into general register ry. |
| Load Halfword | LH ry, offset (rx)<br>The 5-bit immediate is shifted left one bit, zero extended, and then added to the contents of general register rx to form the virtual address. The halfword of the memory location specified by the address is sign extended and loaded to general register ry.<br>If the least significant bit of the address is not 0, an address error exception is generated. |
| Load Halfword Unsigned | LHU ry, offset (rx)<br>The 5-bit immediate is shifted left one bit, zero extended, and then added to the contents of general register rx to form the virtual address. The halfword of the memory location specified by the address is zero extended and loaded to general register ry.<br>If the least significant bit of the address is not 0, an address error exception is generated. |
| Load Word | LW ry, offset (rx)<br>The 5-bit immediate is shifted left two bits, zero extended, and then added to the contents of general register rx to form the virtual address. The word of the memory location specified by the address is loaded to general register ry. In the 64-bit mode, it is further sign extended to 64 bits.<br>If either of the lower two bits is not 0, an address error exception is generated. |
| | LW rx, offset (pc)<br>The two lower bits of the BasePC value associated with the instruction are cleared to form the masked BasePC value. The 8-bit immediate is shifted left two bits, zero extended, and then added to the masked BasePC to form the virtual address. The contents of the word at the memory location specified by the address are loaded to general register rx. In the 64-bit mode, it is further sign extended to 64 bits. |
| | LW rx, offset (sp).<br>The 8-bit immediate is shifted left two bits, zero extended, and then added to the contents of general register sp to form the virtual address. The contents of the word at the memory location specified by the address are loaded to general register rx. In the 64-bit mode, it is further sign extended to 64 bits.<br>If either of the two lower bits of the address is 0, an address error exception is generated. |

**Table 2-35. Load and Store Instructions (2/3)**

| Instruction | Format and Description |
|---|---|
| Load Word Unsigned | LWU ry, offset (rx)<br>The 5-bit immediate is shifted left two bits, zero extended to 64 bits, and then added to the contents of general register rx to form the virtual address. The word of the memory location specified by the address is zero extended and loaded to general register ry.<br>If either of the two lower bits of the address is not 0, an address error exception is generated.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| Load Doubleword | LD ry, offset (rx)<br>The 5-bit immediate is shifted left three bits, zero extended to 64 bits, and then added to the contents of general register rx to form the virtual address. The 64-bit doubleword of the memory location specified by the address is loaded to general register ry.<br>If any of the lower three bits of the address is not 0, an address error exception is generated.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| | LD ry, offset (pc)<br>The lower three bits of the base PC value related to the instruction are cleared to form the masked BasePC value. The 5-bit immediate is shifted left three bits, zero extended to 64 bits, and then added to the masked BasePC to form the virtual address. The 64-bit doubleword at the memory location specified by the address is loaded to general register ry.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| | LD ry, offset (sp)<br>The 5-bit immediate is shifted left three bits, zero extended to 64 bits, and added to the contents of general register sp to form the virtual address. The 64-bit doubleword at the memory location specified by the address is loaded to general register ry.<br>If any of the three lower bits of the address is not 0, an address error exception is generated.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| Store Byte | SB ry, offset (rx)<br>The 5-bit immediate is zero extended and then added to the contents of general register rx to form the virtual address. The least significant byte of general register ry is stored to the memory location specified by the address. |
| Store Halfword | SH ry, offset (rx)<br>The 5-bit immediate is shifted left one bit, zero extended, and then added to the contents of general register rx to form the virtual address. The lower halfword of general register ry is stored to the memory location specified by the address.<br>If the least significant bit of the address is not 0, an address error exception is generated. |

**Table 2-35. Load and Store Instructions (3/3)**

| Instruction | Format and Description |
|---|---|
| Store Word | SW ry, offset (rx)<br>The 5-bit immediate is shifted left two bits, zero extended, and then added to the contents of general register rx to form the virtual address. The contents of general register ry are stored to the memory location specified by the address.<br>If either of the two lower bits of the address is not 0, an address error exception is generated. |
| | SW rx, offset (sp)<br>The 8-bit immediate is shifted left two bits, zero extended, and then added to the contents of general register sp to form the virtual address. The contents of general register rx are stored to the memory location specified by the address.<br>If either of the two lower bits of the address is not 0, and address error exception is generated. |
| | SW ra, offset (sp)<br>The 8-bit immediate is shifted left two bits, zero extended, and then added to the contents of general register sp to form the virtual address. The contents of general register ra are stored to the memory location specified by the address.<br>If either of the two lower bits of the address is not 0, an address error exception is generated. |
| Store Doubleword | SD ry, offset (rx)<br>The 5-bit immediate is shifted left three bits, zero extended to 64 bits, and then added to the contents of general register rx to form the virtual address. The 64 bits of general register ry are stored to the memory location specified by the address. If any of the lower three bits of the address is not 0, an address error exception is generated.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| | SD ry, offset (sp)<br>The 5-bit immediate is shifted left three bits, zero extended to 64 bits, and then added to the contents of general register sp to form the virtual address. The 64 bits of general register ry are stored to the memory location specified by the address.<br>If any of the lower three bits of the address is not 0, an address error exception is generated.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| | SD ra, offset (sp).<br>The 8-bit immediate is shifted left three bits, zero extended to 64 bits, and then added to the contents of general register sp to form the virtual address. The 64 bits of general register ra are stored to the memory location specified by the memory.<br>If any of the three lower bits of the address is not 0, an address error exception is generated.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |

**(2) Computational instructions**

Computational instructions perform arithmetic, logical, and shift operations on values in registers. There are four categories of Computational instructions: ALU Immediate, Two/Three-Operand Register-Type, Shift, and Multiply/Divide.

**Table 2-36. ALU Immediate Instructions (1/2)**

| Instruction | Format and Description |
|---|---|
| Load Immediate | LI rx, immediate<br>The 8-bit immediate is zero extended and loaded to general register rx. |
| Add Immediate Unsigned | ADDIU ry, rx, immediate<br>The 4-bit immediate is sign extended and then added to the contents of general register rx to form a 32-bit result. The result is placed into general register ry. No integer overflow exception occurs under any circumstances. In the 64-bit mode, the operand must be a 64-bit value formed by sign-extending a 32-bit value. |
| | ADDIU rx, immediate<br>The 8-bit immediate is sign extended and then added to the contents of general register rx to form a 32-bit result. The result is placed into general register rx. No integer overflow exception occurs under any circumstances. In the 64-bit mode, the operand must be a 64-bit value formed by sign-extending a 32-bit value. |
| | ADDIU sp, immediate<br>The 8-bit immediate is shifted left three bits, sign extended, and then added to the contents of general register sp to form a 32-bit result. The result is placed into general register sp. No integer overflow exception occurs under any circumstances. In the 64-bit mode, the operand must be a 64-bit value formed by sign-extending a 32-bit value. |
| | ADDIU rx, pc, immediate<br>The two lower bits of the BasePC value associated with the instruction are cleared to form the masked BasePC value. The 8-bit immediate is shifted left two bits, zero extended, and then added to the masked BasePC value to form the virtual address. This address is placed into general register rx. No integer overflow exception occurs under any circumstances. |
| | ADDIU rx, sp, immediate<br>The 8-bit immediate is shifted left two bits, zero extended, and then added to the contents of register sp to form a 32-bit result. The result is placed into general register rx. No integer overflow exception occurs under any circumstance. In the 64-bit mode, the operand must be a 64-bit value formed by sign-extending a 32-bit value. |

**Table 2-36. ALU Immediate Instructions (2/2)**

| Instruction | Format and Description |
|---|---|
| Doubleword Add Immediate Unsigned | DADDIU ry, rx, immediate<br>The 4-bit immediate is sign extended to 64 bits, and then added to the contents of register rx to form a 64-bit result. The result is placed into general register ry. No integer overflow exception occurs under any circumstances.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| | DADDIU ry, immediate<br>The 5-bit immediate is sign extended to 64 bits, and then added to the contents of register ry to form a 64-bit result. The result is placed into general register ry. No integer overflow exception occurs under any circumstances.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| | DADDIU sp, immediate<br>The 8-bit immediate is shifted left three bits, sign extended to 64 bits, and then added to the contents of register sp to form a 64-bit result. The result is placed into general register sp. No integer overflow exception occurs under any circumstances.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| | DADDIU ry, pc, immediate<br>The two lower bits of the BasePC value associated with the instruction are cleared to form the masked BasePC value. The 5-bit immediate is shifted left two bits, zero extended, and added to the masked BasePC value to form the virtual address. This address is placed into general register ry. No integer overflow exception occurs under any circumstances.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| | DADDIU ry, sp, immediate<br>The 5-bit immediate is shifted left two bits, zero extended to 64 bits, and then added to the contents of register sp to form a 64-bit result. This result is placed into register ry. No integer overflow exception occurs under any circumstances.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| Set on Less Than Immediate | SLTI rx, immediate<br>The 8-bit immediate is zero extended and subtracted from the contents of general register rx. Considering both quantities as signed integers, if rx is less than the zero-extended immediate, the result is set to 1; otherwise, the result is set to 0. The result is placed into register T ($24). |
| Set on Less Than Immediate Unsigned | SLTIU rx, immediate<br>The 8-bit immediate is zero extended and subtracted from the contents of general register rx. Considering both quantities as signed integers, if rx is less than the zero-extended immediate, the result is set to 1; otherwise, the result is set to 0. The result is placed into register T ($24). |
| Compare Immediate | CMPI rx, immediate<br>The 8-bit immediate is zero extended and exclusive ORed in 1-bit units with the contents of general register rx. The result is placed into register T ($24). |

**Table 2-37. Two-/Three-Operand Register Type (1/2)**

| Instruction | Format and Description |
|---|---|
| Add Unsigned | ADDU rz, rx, ry<br>The contents of general registers rx and ry are added together to form a 32-bit result. The result is placed into general register rz. No integer overflow exception occurs under any circumstances. In the 64-bit mode, the operand must be a 64-bit value formed by sign-extending a 32-bit value. |
| Subtract Unsigned | SUBU rz, rx, ry<br>The contents of general register ry are subtracted from the contents of general register rx. The 32-bit result is placed into general register rz. No integer overflow exception occurs under any circumstances. In the 64-bit mode, the operand must be a 64-bit value formed by sign-extending a 32-bit value. |
| Doubleword Add Unsigned | DADDU rz, rx, ry<br>The contents of general register ry are added to the contents of general register rx. The 64-bit result is placed into register rz. No integer overflow exception occurs under any circumstances.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| Doubleword Subtract Unsigned | DSUBU rz, rx, ry<br>The contents of general register ry are subtracted from the contents of general register rx. The 64-bit result is placed into general register rz. No integer overflow exception occurs under any circumstances.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| Set on Less Than | SLT rx, ry<br>The contents of general register ry are subtracted from the contents of general register rx. Considering both quantities as signed integers, if the contents of rx are less than the contents of ry, the result is set to 1; otherwise, the result is set to 0. The result is placed into register T ($24).<br>No integer overflow exception occurs. The comparison is valid even if the subtraction overflows. |
| Set on Less Than Unsigned | SLTU rx, ry<br>The contents of general register ry are subtracted from the contents of general register rx. Considering both quantities as unsigned integers, if the contents of rx are less than the contents of ry, the result is set to 1; otherwise, the result it set to 0. The result is place in register T ($24).<br>No integer overflow exception occurs. The comparison is valid even if the subtraction overflows. |
| Compare | CMP rx, ry<br>The contents of general register ry are Exclusive-ORed with the contents of general register rx. The result is placed into register T ($24). |
| Negate | NEG rx, ry<br>The contents of general register ry are subtracted from zero to form a 32-bit result. The result is placed in general register rx. |
| AND | AND rx, ry<br>The contents of general register ry are logical ANDed with the contents of general register rx in 1-bit units. The result is placed in general register rx. |
| OR | OR rx, ry<br>The contents of general register ry are logical ORed with the contents of general register ry. The result is placed in general register rx. |

**Table 2-37. Two-/Three-Operand Register Type (2/2)**

| Instruction | Format and Description |
|---|---|
| Exclusive OR | XOR rx, ry<br>The contents of general register ry are Exclusive-ORed with the contents of general register rx in 1-bit units. The result is placed in general register rx. |
| NOT | NOT rx, ry<br>The contents of general register ry are inverted in 1-bit units and placed in general register rx. |
| Move | MOVE ry, r32<br>The contents of general register r32 are moved to general register ry. r32 can specify any one of the 32 general registers. |
| Move | MOVE r32, rz<br>The contents of general register rz are moved to general register r32. r32 can specify any one of the 32 general registers. |

**Table 2-38. Shift Instructions (1/2)**

| Instruction | Format and Description |
|---|---|
| Shift Left Logical | SLL rx, ry, immediate<br>The 32-bit contents of general register ry are shifted left and zeros are inserted into the emptied low-order bits. The 3-bit immediate specifies the shift count. A shift count of 0 is interpreted as a shift count of 8. The result is placed in general register rx. In the 64-bit mode, the value that is formed by sign-extending shifted 32-bit value is stored as the result. |
| Shift Right Logical | SRL rx, ry, immediate<br>The 32-bit contents of general register ry are shifted right, and zeros are inserted into the emptied high-order bits. The 3-bit immediate specifies the shift count. A shift count of 0 is interpreted as a shift count of 8. The result is placed in general register rx. In the 64-bit mode, the value that is formed by sign-extending shifted 32-bit value is stored as the result. |
| Shift Right Arithmetic | SRA rx, ry, immediate<br>The 32-bit contents of general register ry are shifted right and the emptied high-order bits are sign extended. The 3-bit immediate specifies the shift count. A shift count of 0 is interpreted as a shift count of 8. In the 64-bit mode, the value that is formed by sign-extending shifted 32-bit value is stored as the result. |
| Shift Left Logical Variable | SLLV ry, rx<br>The 32-bit contents of general register ry are shifted left, and zeros are inserted into the emptied low-order bits. The five low-order bits of general register rx specify the shift count. The result is placed in general register ry. In the 64-bit mode, the value that is formed by sign-extending shifted 32-bit value is stored as the result. |
| Shift Right Logical Variable | SRLV ry, rx<br>The 32-bit contents of general register ry are shifted right, and the emptied high-order bits are sign extended. The five lower-order bits of general register rx specify the shift count. The register is placed in general register ry. In the 64-bit mode, the value that is formed by sign-extending shifted 32-bit value is stored as the result. |
| Shift Right Arithmetic Variable | SRAV ry, rx<br>The 32-bit contents of general register ry are shifted right, and the emptied high-order bits are sign extended. The five low-order bits of general register rx specify the shift count. The result is placed in general register ry. In the 64-bit mode, the value that is formed by sign-extending shifted 32-bit value is stored as the result. |

**Table 2-38. Shift Instructions (2/2)**

| Instruction | Format and Description |
|---|---|
| Doubleword Shift Left Logical | DSLL rx, ry, immediate<br>The 64-bit doubleword contents of general register ry are shifted left, and zeros are inserted into the emptied low-order bits. The 3-bit immediate specifies the shift count. A shift count of 0 is interpreted as a shift count of 8. The 64-bit result is placed in general register rx.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| Doubleword Shift Right Logical | DSRL ry, immediate<br>The 64-bit doubleword contents of general register ry are shifted right, and zeros are inserted into the emptied high-order bits. The 3-bit immediate specifies the shift count. A shift count of 0 is interpreted as a shift count of 8. The result is placed in general register ry.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| Doubleword Shift Right Arithmetic | DSRA ry, immediate<br>The 64-bit doubleword contents of general register ry are shifted right, and the emptied high-order bits are sign extended. The 3-bit immediate specifies the shift count. A shift count of 0 is interpreted as a shift count of 8. The result is placed in general register ry.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| Doubleword Shift Left Logical Variable | DSLLV ry, rx<br>The 64-bit doubleword contents of general register ry are shifted left, and zeros are inserted into the emptied low-order bits. The six low-order bits of general register rx specify the shift count. The result is placed in general register ry.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| Doubleword Shift Right Logical Variable | DSRLV ry, rx<br>The 64-bit doubleword contents of general register ry are shifted right, and zeros are inserted into the emptied high-order bits. The six low-order bits of general register rx specify the shift count. The result is placed in general register ry.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| Doubleword Shift Right Arithmetic Variable | DSRAV ry, rx<br>The 64-bit doubleword contents of general register ry are shifted right, and the emptied high-order bits are sign extended. The six low-order bits of general register rx specify the shift count. The result is placed in general register ry.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |

**Table 2-39. Multiply/Divide Instructions (1/2)**

| Instruction | Format and Description |
|---|---|
| Multiply | MULT rx, ry<br>The contents of general registers rx and ry are multiplied, treating both operands as 32-bit two's complement values. No integer overflow exception occurs.<br>In the 64-bit mode, the operand must be a 64-bit value formed by sign-extending a 32-bit value.<br>The low-order 32-bit word of the result are placed in special register LO, and the high-order 32-bit word is placed in special register HI. In the 64-bit mode, each result is sign extended and then stored.<br>If either of the two immediately preceding instructions is MFHI or MFLO, their transfer instruction execution result becomes undefined. To obtain the correct result, insert two or more other instructions between the MFHI, MFLO instructions, and the MULT instruction. |
| Multiply Unsigned | MULTU rx, ry<br>The contents of general registers rx and ry are multiplied, treating both operands as 32-bit unsigned values. No integer overflow exception occurs. In the 64-bit mode, the operand must be a 64-bit value formed by sign-extending a 32-bit value. The low-order 32-bit word of the result is placed in special register LO, and the high-order 32-bit word is placed in special register HI. In the 64-bit mode, each result is sign extended and stored.<br>If either of the two immediately preceding instructions is MFHI or MFLO, the result of execution of these transfer instructions is undefined. To obtain the correct result, insert two or more other instructions between the MFHI, MFLO instructions and the MULTU instruction. |
| Divide | DIV rx, ry<br>The contents of general register rx are divided by the contents of general register ry, treating both operands as 32-bit two's complement values. No integer overflow exception occurs. The result when the divisor is 0 is undefined. The 32-bit quotient is placed in special register LO, and the 32-bit remainder is placed in special register HI. In the 64-bit mode, the result is sign extended.<br>Normally, this instruction is executed after instructions checking for division by zero and overflow.<br>If either of the two immediately preceding instructions is MFHI or MFLO, the result of execution of these transfer instructions is undefined. To obtain the correct result, insert two or more other instructions between the MFHI, MFLO instructions and the DIV instruction. |
| Divide Unsigned | DIVU rx, ry<br>The contents of general register rx are divided by the contents of general register ry, treating both operands as 32-bit unsigned values. No integer overflow exception occurs. The result when the divisor is 0 is undefined. The 32-bit quotient is placed in special register LO, and the 32-bit remainder is placed in special register HI. In the 64-bit mode, the result is sign extended.<br>Normally, this instruction is executed after instructions checking for division by zero.<br>If either of the two immediately preceding instructions is MFHI or MFLO, the result of execution of these transfer instructions is undefined. To obtain the correct result, insert two or more other instructions between the MFHI, MFLO instructions and the DIVU instruction. |
| Move From HI | MFHI rx<br>The contents of special register HI are loaded into general register rx.<br>To ensure correct operation when an interrupt occurs, do not use an instruction that changes the HI register (MULT, MULTU, DIV, DIVU, DMULT, DMULTU, DDIV, DDIVU) for the two instructions after the MFHI instruction. |
| Move From LO | MFLO rx<br>The contents of special register LO are loaded into general register rx.<br>To ensure correct operation when an interrupt occurs, do not use an instruction that changes the LO register (MULT, MULTU, DIV, DIVU, DMULT, DMULTU, DDIV, DDIVU) for the two instructions after the MFLO instruction. |

**Table 2-39. Multiply/Divide Instructions (2/2)**

| Instruction | Format and Description |
|---|---|
| Doubleword Multiply | DMULT rx, ry<br>The 64-bit contents of general register rx and ry are multiplied, treating both operands as two's complement values. No integer overflow exception occurs. The low-order 64 bits of the result are placed in special register LO, and the high-order 64 bits are placed in special register HI.<br>If either of the two immediately preceding instructions is MFHI or MFLO, the result of execution of these transfer instructions is undefined. To obtain the correct result, insert two or more other instructions between the MFHI, MFLO instructions and the DMULT instruction.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| Doubleword Multiply Unsigned | DMULTU rx, ry<br>The 64-bit contents of general registers rx and ry are multiplied, treating both operands as unsigned values. No integer overflow exception occurs. The low-order 64 bits of the result are placed in special register LO, and the high-order 64 bits of the result are placed in special register HI.<br>If either of the two immediately preceding instructions is MFHI or MFLO, the result of execution of these transfer instructions is undefined. To obtain the correct result, insert two or more other instructions between the MFHI, MFLO instructions and the DMULTU instruction.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| Doubleword Divide | DDIV rx, ry<br>The 64-bit contents of general registers rx are divided by the contents of general register ry, treating both operands as two's complement values. No integer overflow exception occurs. The result when the divisor is 0 is undefined. The 64-bit quotient is placed in special register LO, and the 64-bit remainder is placed in special register HI. Normally, this instruction is executed after instructions checking for division by zero and overflow.<br>If either of the two immediately preceding instructions is MFHI or MFLO, the result of execution of these transfer instructions is undefined. To obtain the correct result, insert two or more other instructions between the MFHI, MFLO instructions and the DDIV instruction.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |
| Doubleword Divide Unsigned | DDIVU rx, ry<br>The 64-bit contents of general register rx are divided by the contents of general register ry, treating both operands as unsigned values. No integer overflow exception occurs. The result when the divisor is 0 is undefined. The 64-bit quotient is placed in special register LO, and the 64-bit remainder is placed in special register HI. Normally, this instruction is executed after an instruction checking for division by zero.<br>If either of the two immediately preceding instructions is MFHI or MFLO, the result of execution of these transfer instructions is undefined. To obtain the correct result, insert two or more other instructions between the MFHI, MFLO instructions and the DDIVU instruction.<br>This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated. |

**(3) Jump and branch instructions**

Jump and Branch instructions change the control flow of a program.

All Jump instructions occur with a one-instruction delay. That is, the instruction immediately following the jump is always executed.

Branch instructions do not have a delay slot. If a branch is taken, the instruction immediately following the branch is never executed. If the branch is not taken, the instruction immediately following the branch is always executed.

Table 2-40 shows the MIPS16 Jump and Branch instructions.

**Table 2-40. Jump and Branch Instructions (1/2)**

| Instruction | Format and Description |
|---|---|
| Jump and Link | JAL target<br>The 26-bit target address is shifted left two bits and combined with the high-order four bits of the address of the delay slot. The program unconditionally jumps to this calculated address with a delay of one instruction. The address of the instruction immediately following the delay slot is placed in register ra. The ISA Mode bit is left unchanged. The value stored in ra bit 0 will reflect the current ISA Mode bit. |
| Jump and Link Exchange | JALX target<br>The 26-bit target address is shifted left two bits and combined with the high-order four bits of the address of the delay slot. The program unconditionally jumps to this calculated address with a delay of one instruction. The address of the instruction immediately following the delay slot is placed in register ra. The ISA Mode bit is inverted with a delay of one instruction. The value stored in ra bit 0 will reflect the ISA Mode bit before execution of the Jump execution. |
| Jump Register | JR rx<br>The program unconditionally jumps to the address specified in general register rx, with a delay of one instruction. The instruction sets the ISA Mode bit to the value in rx bit 0. If the Jump target address is in the MIPS16 instruction length mode, no address exception occurs when bit 0 of the source register is 1 because bit 0 of the target address is 0 so that the instruction is located at the halfword boundary. If the 32-bit length instruction mode is changed, an address exception occurs when the jump target address is fetched if the two low-order bits of the target address are not 0. |
| | JR ra<br>The program unconditionally jumps to the address specified in register ra, with a delay of one instruction. The instruction sets the ISA Mode bit to the value in ra bit 0. If the Jump target address is in the MIPS16 instruction length mode, no address exception occurs when bit 0 of the source register is 1 because bit 0 of the target address is 0 so that the instruction is located at the halfword boundary. If the 32-bit length instruction mode is changed, an address exception occurs when the jump target address is fetched if the two low-order bits of the target address are not 0. |

**Table 2-40. Jump and Branch Instructions (2/2)**

| Instruction | Format and Description |
|---|---|
| Jump and Link Register | JALR ra, rx<br>The program unconditionally jumps to the address contained in register rx, with a delay of one instruction. This instruction sets the ISA Mode bit to the value in rx bit 0. The address of the instruction immediately following the delay slot is placed in register ra. The value stored in ra bit 0 will reflect the ISA mode bit before the jump execution is executed.<br>If the Jump target address is in the MIPS16 instruction length mode, no address exception occurs when bit 0 of the source register is 1 because bit 0 of the target address is 0 so that the instruction is located at the halfword boundary.<br>If the 32-bit length instruction mode is changed, an address exception occurs when the jump target address is fetched if the two low-order bits of the target address are not 0. |
| Branch on Equal to Zero | BEQZ rx, immediate<br>The 8-bit immediate is shifted left one bit, sign extended, and then added to the address of the instruction after the branch to form the target address. If the contents of general register rx are equal to zero, the program branches to the target address. No delay slot is generated. |
| Branch on Not Equal to Zero | BNEZ rx, immediate<br>The 8-bit immediate is shifted left one bit, sign extended, and then added to the address of the instruction after the branch to form the target address. If the contents of general register rx are not equal to zero, the program branches to the target address. No delay slot is generated. |
| Branch on T Equal to Zero | BTEQZ immediate<br>The 8-bit immediate is shifted left one bit, sign extended, and then added to the address of the instruction after the branch to form the target address. If the contents of special register T ($24) are equal to zero, the program branches to the target address. No delay slot is generated. |
| Branch on T Not Equal to Zero | BTNEZ immediate<br>The 8-bit immediate is shifted left one bit, sign extended, and then added to the address of the instruction after the branch to form the target address. If the contents of special register T ($24) are not equal to zero, the program branches to the target address. No delay slot is generated. |
| Branch Unconditional | B immediate<br>The 11-bit immediate is shifted left one bit, sign extended, and then added to the address of the instruction after the branch to form the target address. The program branches to the target address unconditionally. No delay slot is generated. |

**(4) Special instructions**

Special instructions unconditionally perform branching to general exception vectors. Special instructions are of the R type. Table 2-41 shows two special instructions.

**Table 2-41. Special Instructions**

| Instruction | Format and Description |
|---|---|
| Breakpoint | BREAK immediate<br>A breakpoint trap occurs, unconditionally transferring control to the exception handler.<br>By using a 6-bit code area, parameters can be sent to the exception handler. If the exception handler uses this parameter, the contents of memory including instructions must be loaded as data. |
| Extend | EXTEND immediate<br>The 11-bit immediate is combined with the immediate in the next instruction to form a larger immediate equivalent to 32-bit MIPS. The Extend instruction must always precede (by one instruction) the instruction whose immediate field you want to extend. Every extended instruction consumes four bytes in program memory instead of two bytes (two bytes for Extend and two bytes for the instruction being extended), and it can cross a word boundary. (For details, see **2.3.7.2 Extend instruction**.) |

## 2.4 Pipeline

This section describes the basic operation of the VR4120A Core pipeline, which includes descriptions of the delay slots (instructions that follow a branch or load instruction in the pipeline), interrupts to the pipeline flow caused by interlocks and exceptions, and CP0 hazards.

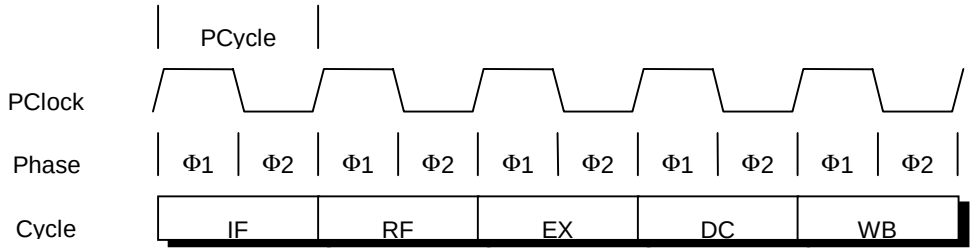
### 2.4.1 Pipeline stages

The pipeline is controlled by PClock(one cycle of PClock which runs at 4-times frequency of MasterClock) and one cycle of this PClock is called PCycle. Each pipeline stage takes one PCycle.

#### 2.4.1.1 Pipeline in MIPS III instruction mode

The VR4120A has a five-stage instruction pipeline; each stage takes one PCycle, and each PCycle has two phases: Φ1 and Φ2, as shown in Figure 2-10. Thus, the execution of each instruction takes at least 5 PCycles. An instruction can take longer - for example, if the required data is not in the cache, the data must be retrieved from main memory.

**Figure 2-10. Pipeline Stages (MIPS III Instruction Mode)**

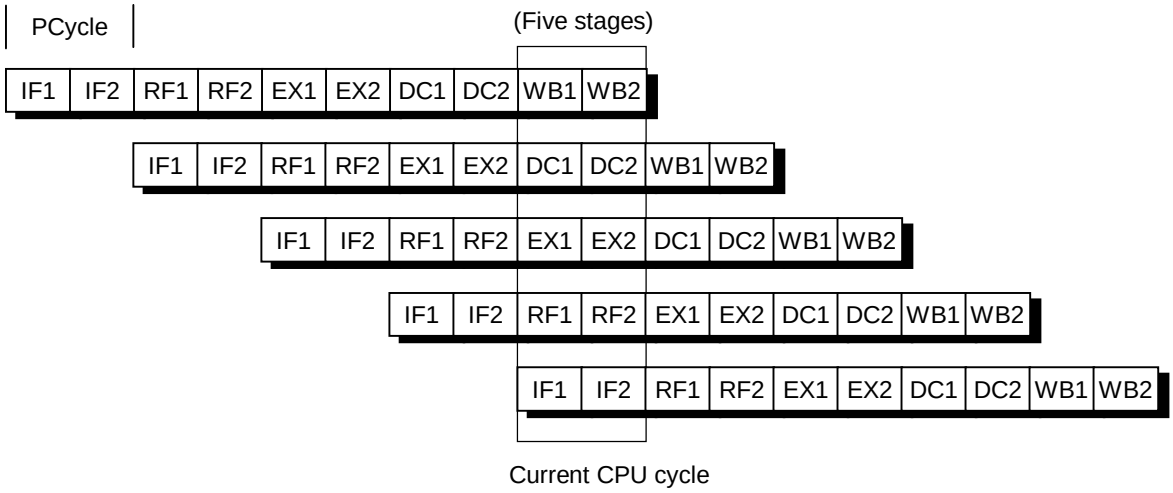



The five pipeline stages are:

- ✧ IF - Instruction cache fetch
- ✧ RF - Register fetch
- ✧ EX - Execution
- ✧ DC - Data cache fetch
- ✧ WB - Write back

Figure 2-11 shows the five stages of the instruction pipeline. In this figure, a row indicates the execution process of each instruction, and a column indicates the processes executed simultaneously.

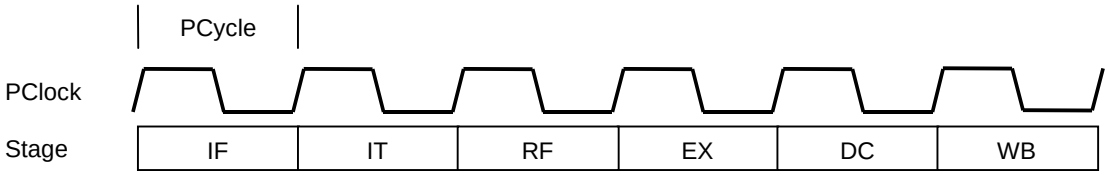
**Figure 2-11. Instruction Execution in the Pipeline**




#### 2.4.1.2 Pipeline in MIPS16 (16-bit length) instruction mode

The VR4120A, which has internal stages for converting MIPS16 instructions into 32-bit instructions, uses a 6-stage pipeline in the MIPS16 mode. As a result, the execution of each instruction requires at least 6 PCycles. If the required data is not cached and must be retrieved from the main memory, a larger number of cycles will be required. If the pipeline flow is smooth, six instructions are executed simultaneously.

**Figure 2-12. Pipeline Stages (MIPS16 Instruction Mode)**

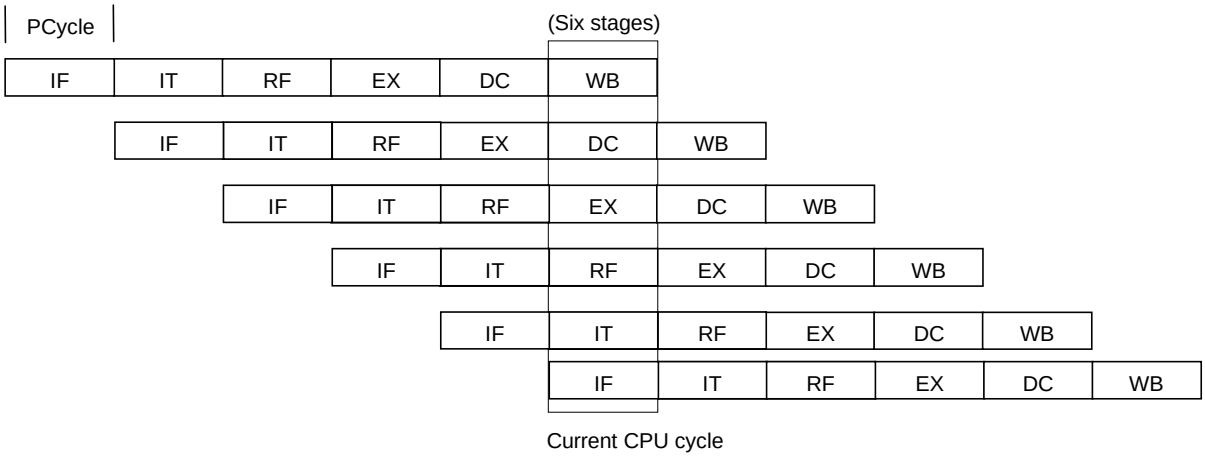



The pipeline stage names and their meaning are described below.

- ✧ IF - Instruction cache fetch
- ✧ IT - Instruction conversion
- ✧ RF - Register fetch
- ✧ EX - Execution
- ✧ DC - Data cache fetch
- ✧ WB - Write back

Figure 2-13 shows the outline of the pipeline. Each row in this figure indicates the execution process of an instruction, and the column indicates the 6 processes that are executed simultaneously.

**Figure 2-13. Instruction Execution in Pipeline (MIPS16 Instruction Mode)**



### 2.4.1.3 Pipeline activities

#### (1) MIPS III instruction

Figure 2-14 shows the activities that can occur during each pipeline stage in MIPS III Instruction mode. Table 2-42 describes these pipeline activities.

**Figure 2-14. Pipeline Activities (MIPS III)**

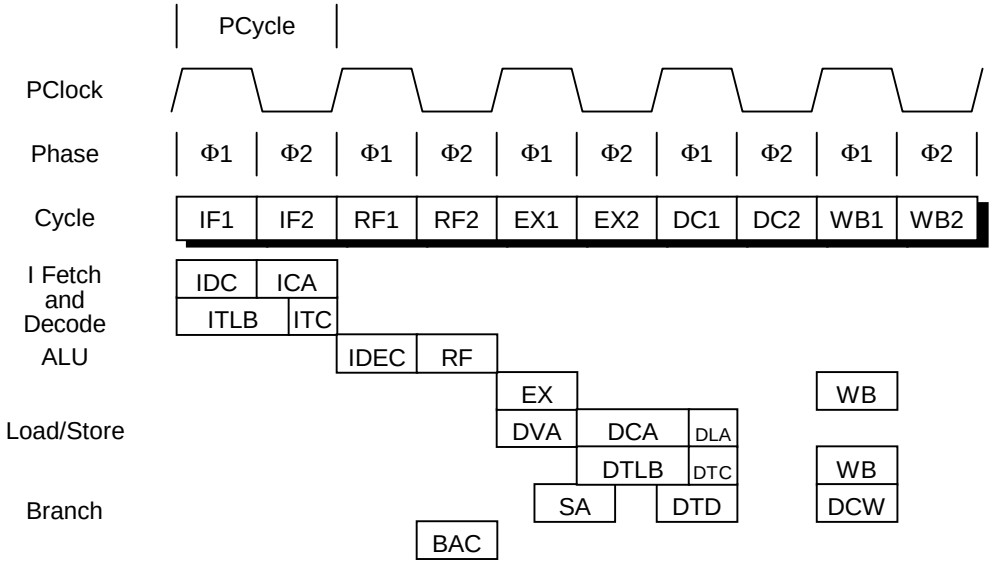



**Table 2-42. Operation in Each Stage of Pipeline (MIPS III)**

| Cycle | Phase | Mnemonic | Description |
|---|---|---|---|
| IF | Φ1 | IDC | Instruction cache address decode |
| | | ITLB | Instruction address translation |
| | Φ2 | ICA | Instruction cache array access |
| | | ITC | Instruction tag check |
| RF | Φ1 | IDEC | Instruction decode |
| | Φ2 | RF | Register operand fetch |
| | | BAC | Branch address calculation |
| EX | Φ1 | EX | Execution stage |
| | | DVA | Data virtual address calculation |
| | | SA | Store align |
| | Φ2 | DCA | Data cache address decode/array access |
| | | DTLB | Data address translation |
| DC | Φ1 | DLA | Data cache load align |
| | | DTC | Data tag check |
| | | DTD | Data transfer to data cache |
| WB | Φ1 | DCW | Data cache write |
| | | WB | Write back to register file |

**(2) MIPS16 instruction**

Figure 2-15 shows the activities that can occur during each pipeline stage; Table 2-43 describes these pipeline activities. Furthermore, IT stage occurs during MIPS16 instruction mode.

**Figure 2-15. Pipeline Activities (MIPS16)**

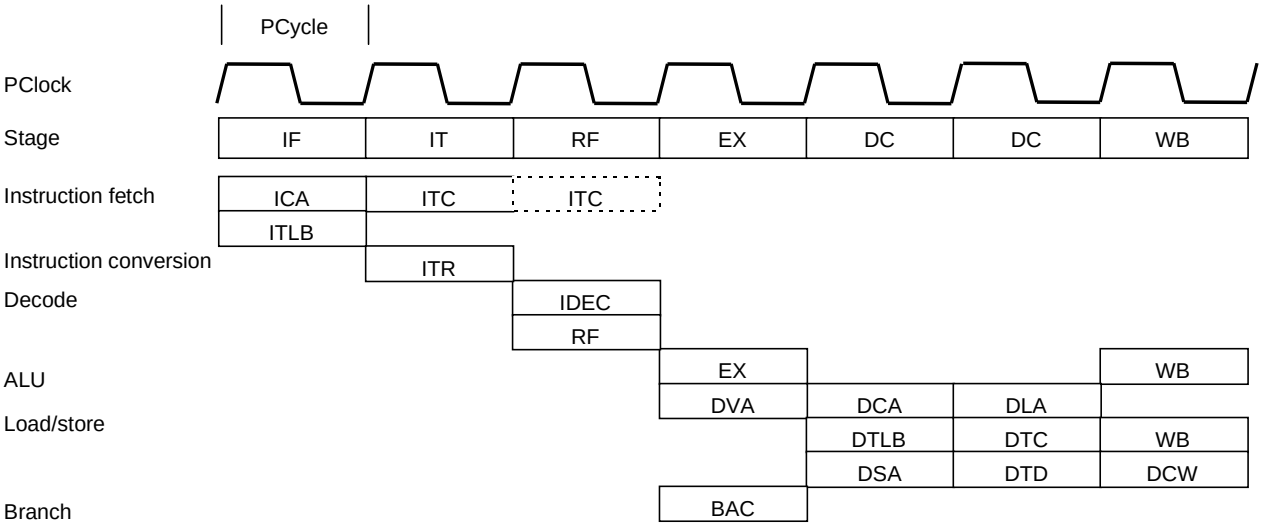



**Table 2-43. Operation in Each Stage of Pipeline (MIPS16)**

| Cycle | Mnemonic | Description |
|---|---|---|
| IF | ITLB | Instruction address conversion |
| | ICA | Instruction cache array access |
| IT | ITR | Instruction conversion |
| | ITC | Instruction tag fetch |
| RF | IDEC | Instruction decode |
| | RF | Register operand fetch |
| | ITC | Instruction tag check (in MIPS III instruction mode) |
| EX | EX | Execution stage |
| | BAC | Branch address calculation |
| | DVA | Data virtual address calculation |
| DC | DLA | Data cache load align |
| | DSA | Store align |
| | DCA | Data cache address decode/array access |
| | DTLB | Data address conversion |
| | DTC | Data tag check |
| | DTD | Data transfer to data cache |
| WB | DCW | Write to data cache |
| | WB | Write back to register file |

### 2.4.2 Branch delay

During a VR4120A's pipeline operation, a one-cycle branch delay occurs when:

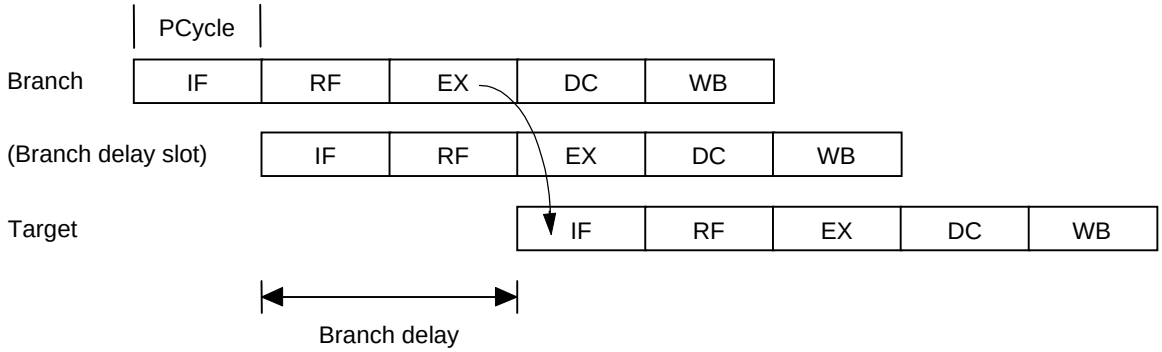
- Target address is calculated by a Jump instruction
- Branch condition of branch instruction is met and then logical operation starts for branch-destination comparison

The instruction location following the Jump/Branch instruction is called a branch delay slot.

The instruction address generated at the EX stage in the Jump/Branch instruction are available in the IF stage, two instructions later. In MIPS III instruction mode, branch delay is two cycles. One instruction in the branch delay slot is executed, except for likely instruction.

Figure 2-16 illustrates the branch delay and the location of the branch delay slot during MIPS III instruction mode.

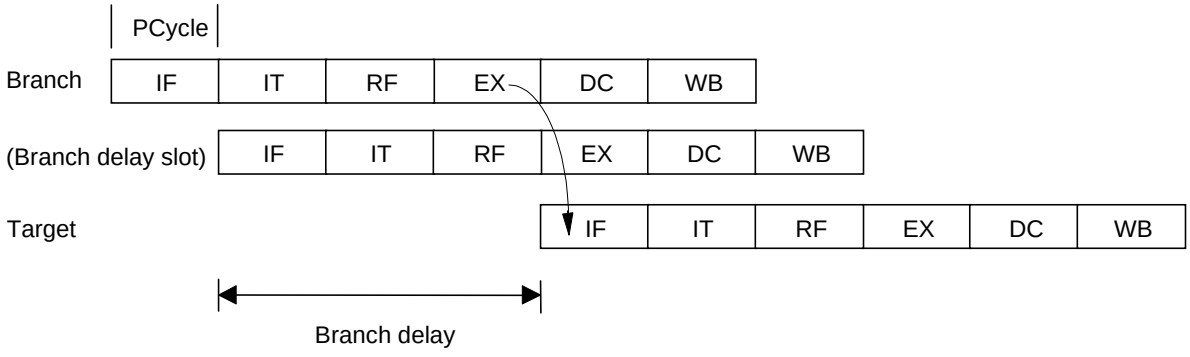
**Figure 2-16. Branch Delay (In MIPS III Instruction Mode)**




In the MIPS16 instruction mode, 3 branch delay cycles occur. If the branch condition of the branch instruction is satisfied, the instruction in the delay slot is discarded.

Figure 2-17 shows the image of branch delay in the MIPS16 instruction mode and the location of the branch delay slot.

**Figure 2-17. Branch Delay (In MIPS16 Instruction Mode)**



### 2.4.3 Load delay

In the case of a load instruction, 2 cycles are required for the DC stage, for reading from the data cache and performing data alignment. In this case, the hardware automatically generates on interlock.

A load instruction that does not allow its result to be used by the instruction immediately following is called a delayed load instruction. The instruction immediately following this delayed load instruction is referred to as the load delay slot.

In the VR4120A, the instruction immediately following a load instruction can use the contents of the loaded register, however in such cases hardware interlocks insert additional delay cycles. Consequently, scheduling load delay slots can be desirable, both for performance and VR-Series processor compatibility.

### 2.4.4 Pipeline operation

The operation of the pipeline is illustrated by the following examples that describe how typical instructions are executed. The instructions described are six: ADD, JALR, BEQ, TLT, LW, and SW. Each instruction is taken through the pipeline and the operations that occur in each relevant stage are described.

#### 2.4.4.1 Add instruction (ADD rd, rs, rt)

IF stage — In Φ1 of the IF stage, the eleven least-significant bits of the virtual access are used to access the instruction cache. In Φ2 of the IF stage, the cache index is compared with the page frame number and the cache data is read out. The virtual PC is incremented by 4 so that the next instruction can be fetched.

IT stage — The MIPS16 instruction is translated into a 32-bit length instruction (during MIPS16 instruction mode only).

RF stage — During Φ2, the 2-port register file is addressed with the rs and rt fields and the register data is valid at the register file output. At the same time, bypass multiplexers select inputs from either the EX- or DC-stage output in addition to the register file output, depending on the need for an operand bypass.

EX stage — The ALU controls are set to do an A + B operation. The operands flow into the ALU inputs, and the ALU operation is started. The result of the ALU operation is latched into the ALU output latch during Φ1.

DC stage — This stage is a NOP for this instruction. The data from the output of the EX stage (the ALU) is moved into the output latch of the DC.

WB stage — During Φ1, the WB latch feeds the data to the inputs of the register file, which is addressed by the rd field. The file write strobe is enabled. By the end of Φ1, the data is written into the file.

**Figure 2-18. ADD Instruction Pipeline Activities (In MIPS III Instruction Mode)**

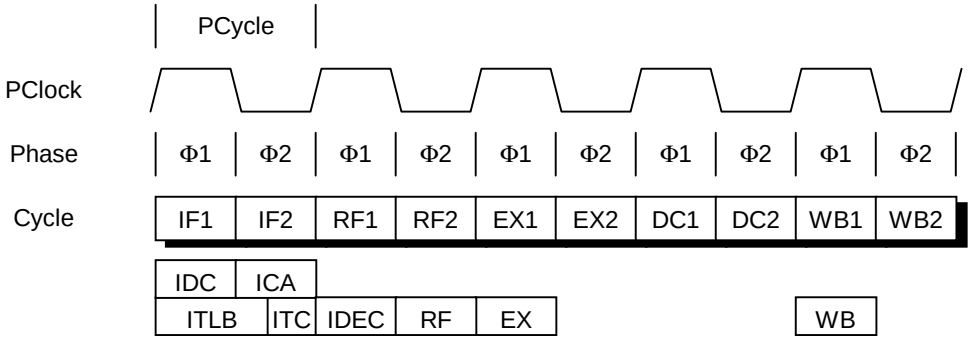
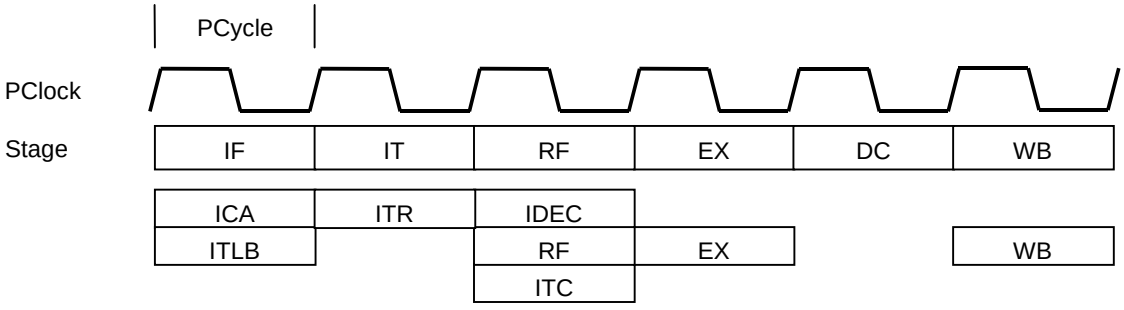



**Figure 2-19. ADD Instruction Pipeline Activities (In MIPS16 Instruction Mode)**



#### 2.4.4.2 Jump and link register instruction (JALR rd, rs)

IF stage — Same as the IF stage for the ADD instruction.

IT stage — Same as the IT stage for the ADD instruction.

RF stage — A register specified in the rs field is read from the file during Φ2 at the RF stage, and the value read from the rs register is input to the virtual PC latch synchronously. This value is used to fetch an instruction at the jump destination. The value of the virtual PC incremented during the IF stage is incremented again to produce the link address PC + 8 (PC + 4 in the case of MIPS16 ISA) where PC is the address of the JALR instruction. The resulting value is the PC to which the program will eventually return. This value is placed in the Link output latch of the Instruction Address unit.

EX stage — The PC + 8 value (PC + 4 value in the case of MIPS16 ISA) is moved from the Link output latch to the output latch of the EX stage.

DC stage — The PC + 8 value (PC + 4 value in the case of MIPS16 ISA) is moved from the output latch of the EX stage to the output latch of the DC stage.

WB stage — Refer to the ADD instruction. Note that if no value is explicitly provided for rd then register 31 is used as the default. If rd is explicitly specified, it cannot be the same register addressed by rs; if it is, the result of executing such an instruction is undefined.

**Figure 2-20. JALR Instruction Pipeline Activities (In MIPS III Instruction Mode)**

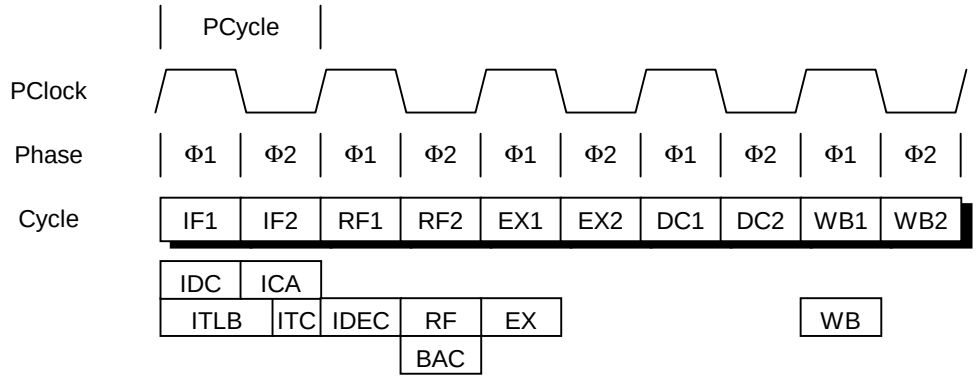
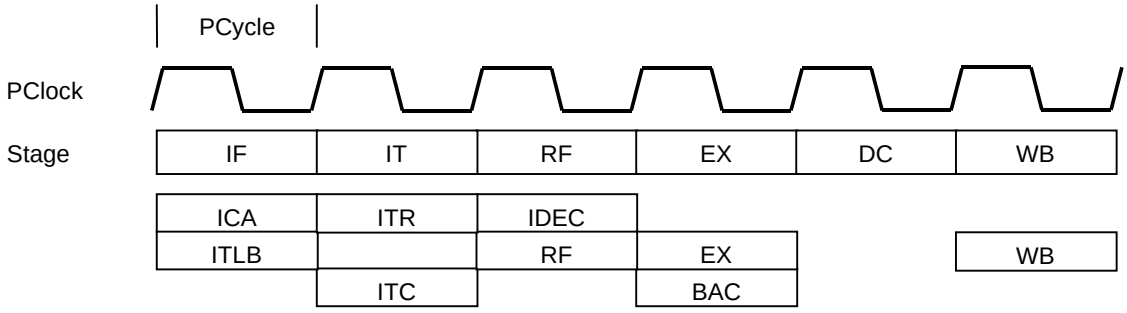



**Figure 2-21. JALR Instruction Pipeline Activities (In MIPS16 Instruction Mode)**



#### 2.4.4.3 Branch on equal instruction (BEQ rs, rt, offset)

| | |
|---|---|
| IF stage | Same as the IF stage for the ADD instruction. |
| IT stage | Same as the IT stage for the ADD instruction. |
| RF stage | During Φ2, the register file is addressed with the rs and rt fields. A check is performed to determine if each corresponding bit position of these two operands has equal values. If they are equal, the PC is set to PC + target, where target is the sign-extended offset field. If they are not equal, the PC is set to PC + 4. |
| EX stage | The next PC resulting from the branch comparison is valid at the beginning of Φ2 for instruction fetch. |
| DC stage | This stage is a NOP for this instruction. |
| WB stage | This stage is a NOP for this instruction. |

**Figure 2-22. BEQ Instruction Pipeline Activities (In MIPS III Instruction Mode)**

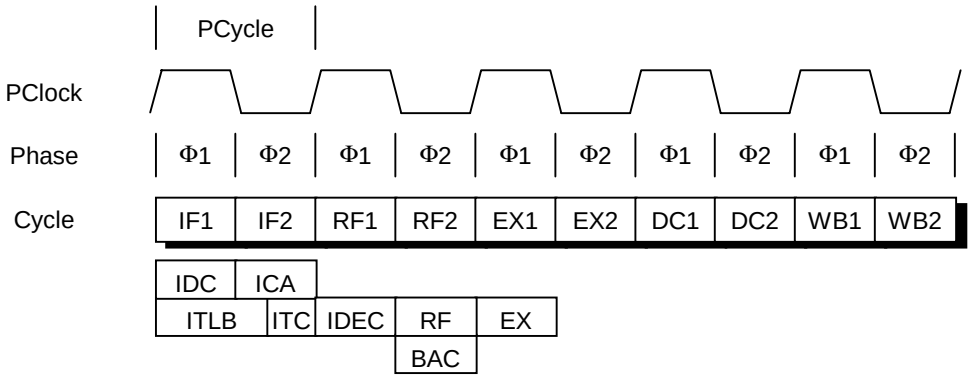
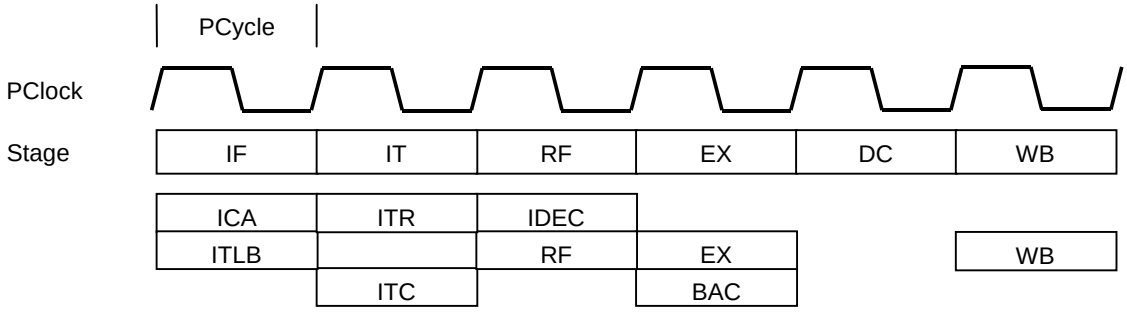


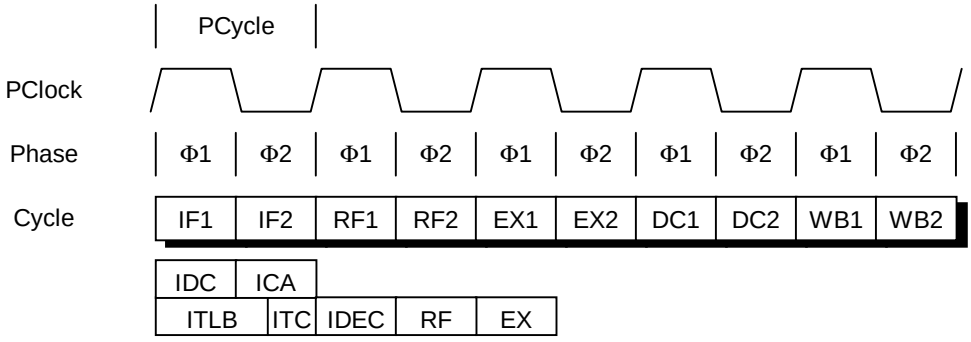

**Figure 2-23. BEQ Instruction Pipeline Activities (In MIPS16 Instruction Mode)**



#### 2.4.4.4 Trap if less than instruction (TLT rs, rt)

| | |
|---|---|
| IF stage | Same as the IF stage for the ADD instruction. |
| IT stage | MIPS16 instruction set does not have TLT instruction, therefore this stage does not occur. |
| RF stage | Same as the RF stage for the ADD instruction. |
| EX stage | ALU controls are set to do an A - B operation. The operands flow into the ALU inputs, and the ALU operation is started. The result of the ALU operation is latched into the ALU output latch during Φ1. The sign bits of operands and of the ALU output latch are checked to determine if a less than condition is true. If this condition is true, a Trap exception occurs. The value in the PC register is used as an exception vector value, and from now on any instruction will be invalid. |
| DC stage | No operation |
| WB stage | The value of the PC is loaded to EPC register if the less than condition was met in the EX stage. The Cause register ExCode field and BD bit are updated appropriately, as is the EXL bit of the Status register. If the less than condition was not met in the EX stage, no activity occurs in the WB stage. |

**Figure 2-24. TLT Instruction Pipeline Activities**



#### 2.4.4.5 Load word instruction (LW rt, offset (base))

IF stage Same as the IF stage for the ADD instruction.

IT stage Same as the IT stage for the ADD instruction.

RF stage Same as the RF stage for the ADD instruction. Note that the base field is in the same position as the rs field.

EX stage Refer to the EX stage for the ADD instruction. For LW, the inputs to the ALU come from GPR[base] through the bypass multiplexer and from the sign-extended offset field. The result of the ALU operation that is latched into the ALU output latch in Φ1 represents the effective virtual address of the operand (DVA).

DC stage The cache tag field is compared with the Page Frame Number (PFN) field of the TLB entry. After passing through the load aligner, aligned data is placed in the DC output latch during Φ2.

WB stage During Φ1, the cache read data is written into the register file addressed by the rt field.

**Figure 2-25. LW Instruction Pipeline Activities (In MIPS III Instruction Mode)**

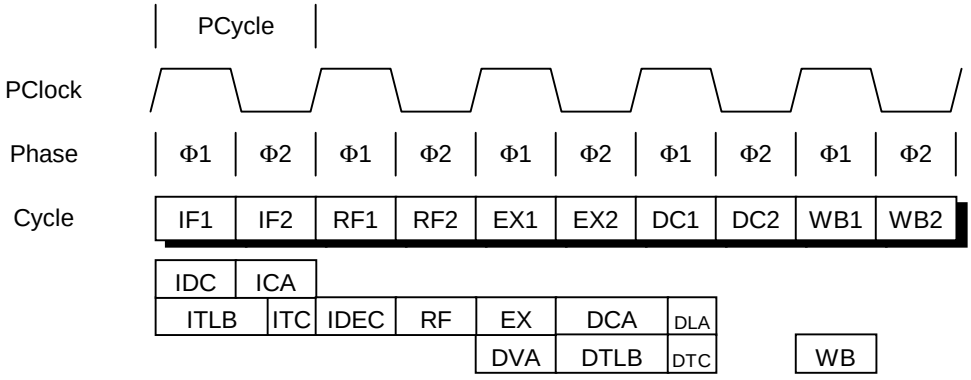
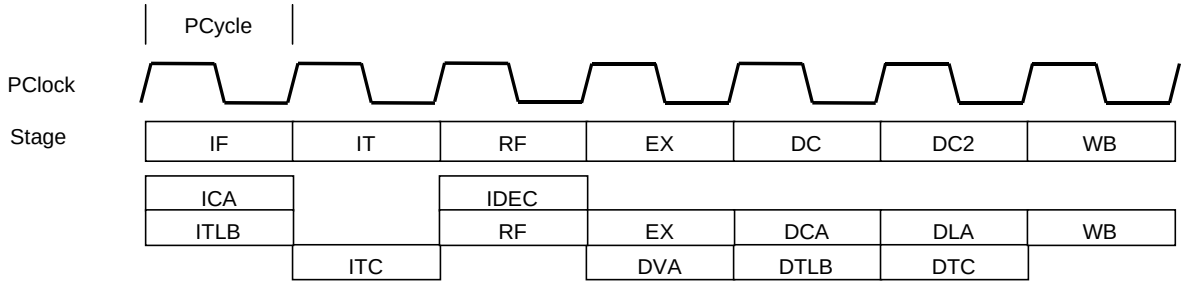



**Figure 2-26. LW Instruction Pipeline Activities (In MIPS16 Instruction Mode)**



#### 2.4.4.6 Store word instruction (SW rt, offset (base))

IF stage — Same as the IF stage for the ADD instruction.

IT stage — Same as the IT stage for the ADD instruction.

RF stage — Same as the RF stage for the LW instruction.

EX stage — Refer to the LW instruction for a calculation of the effective address. From the RF output latch, the GPR[rt] is sent through the bypass multiplexer and into the main shifter, where the shifter performs the byte-alignment operation for the operand. The results of the ALU are latched in the output latches during Φ1. The shift operations are latched in the output latches during Φ2.

DC stage — Refer to the LW instruction for a description of the cache access.

WB stage — If there was a cache hit, the content of the store data output latch is written into the data cache at the appropriate word location.

Note that all store instructions use the data cache for two consecutive PCycles. If the following instruction requires use of the data cache, the pipeline is slipped for one PCycle to complete the writing of an aligned store data.

**Figure 2-27. SW Instruction Pipeline Activities (In MIPS III Instruction Mode)**

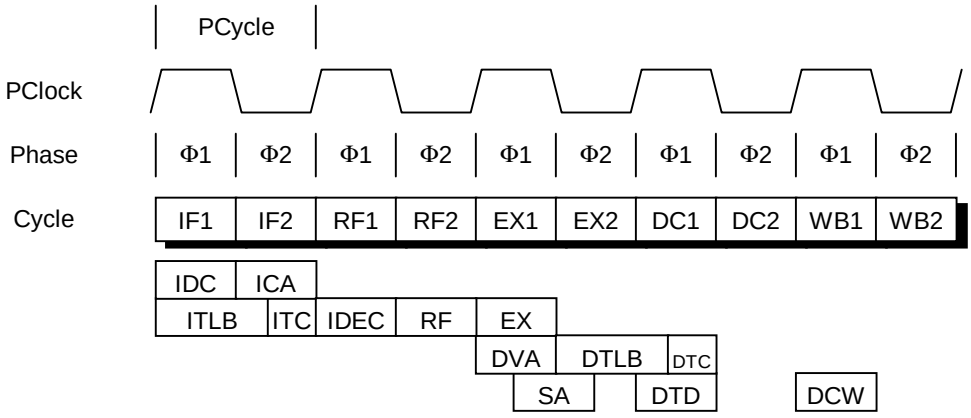
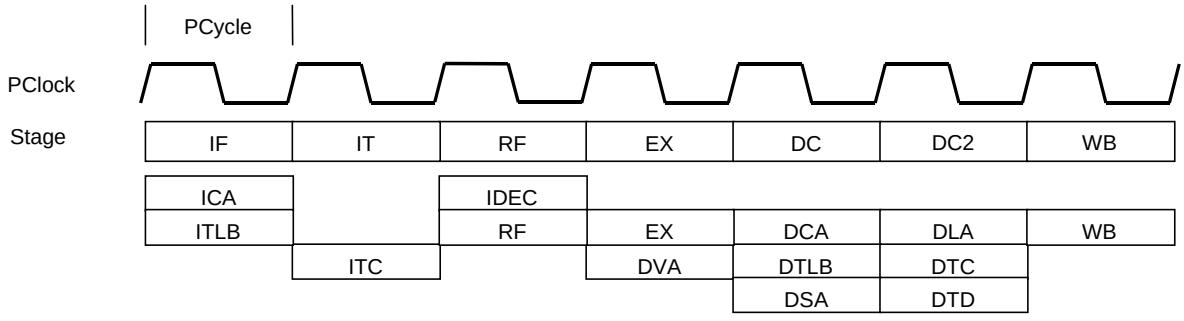


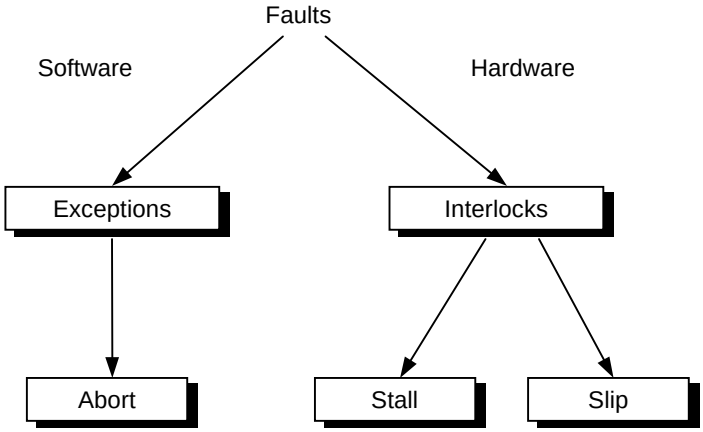

**Figure 2-28. SW Instruction Pipeline Activities (In MIPS16 Instruction Mode)**



### 2.4.5 Interlock and exception handling

Smooth pipeline flow is interrupted when cache misses or exceptions occur, or when data dependencies are detected. Interruptions handled using hardware, such as cache misses, are referred to as interlocks, while those that are handled using software are called exceptions. As shown in Figure 2-22, all interlock and exception conditions are collectively referred to as faults.

**Figure 2-29. Relationship among Interlocks, Exceptions, and Faults**




At each cycle, exception and interlock conditions are checked for all active instructions.

Because each exception or interlock condition corresponds to a particular pipeline stage, a condition can be traced back to the particular instruction in the exception/interlock stage, as shown in Table 2-44. For instance, an LDI Interlock is raised in the Register Fetch (RF) stage.

Tables 2-45 and 2-46 describe the pipeline interlocks and exceptions listed in Table 2-44.

**Table 2-44. Correspondence of Pipeline Stage to Interlock and Exception Conditions**

| Status \ Stage | | IF | RF (IT) | EX | DC | WB |
|---|---|---|---|---|---|---|
| Interlock | Stall | – | ITM<br>ICM | – | DTM<br>DCM<br>DCB | – |
| | Slip | – | LDI<br>MDI<br>SLI<br>CP0 | – | – | – |
| Exception | | IAErr | NMI<br>ITLB<br>IPErr<br>INTr<br>IBE<br>SYSC<br>BP<br>CUn<br>RSVD | Trap<br>OVF<br>DAErr | Reset<br>DTLB<br>TMod<br>DPErr<br>WAT<br>DBE | – |

**Remark** In the above table, exception conditions are listed up in higher priority order.

**Table 2-45. Pipeline Interlock**

| Interlock | Description |
|---|---|
| ITM | Instruction TLB Miss |
| ICM | Instruction Cache Miss |
| LDI | Load Data Interlock |
| MDI | MD Busy Interlock |
| SLI | Store-Load Interlock |
| CP0 | Coprocessor 0 Interlock |
| DTM | Data TLB Miss |
| DCM | Data Cache Miss |
| DCB | Data Cache Busy |

**Table 2-46. Description of Pipeline Exception**

| Exception | Description |
|---|---|
| IAErr | Instruction Address Error exception |
| NMI | Non-maskable Interrupt exception |
| ITLB | ITLB exception |
| IPErr | Instruction Parity Error exception |
| INTr | Interrupt exception |
| IBE | Instruction Bus Error exception |
| SYSC | System Call exception |
| BP | Breakpoint exception |
| CUn | Coprocessor Unusable exception |
| RSVD | Reserved Instruction exception |
| Trap | Trap exception |
| OVF | Integer overflow exception |
| DAErr | Data Address Error exception |
| Reset | Reset exception |
| DTLB | DTLB exception |
| DTMod | DTLB Modified exception |
| DPErr | Data Parity Error exception |
| WAT | Watch exception |
| DBE | Data Bus Error exception |

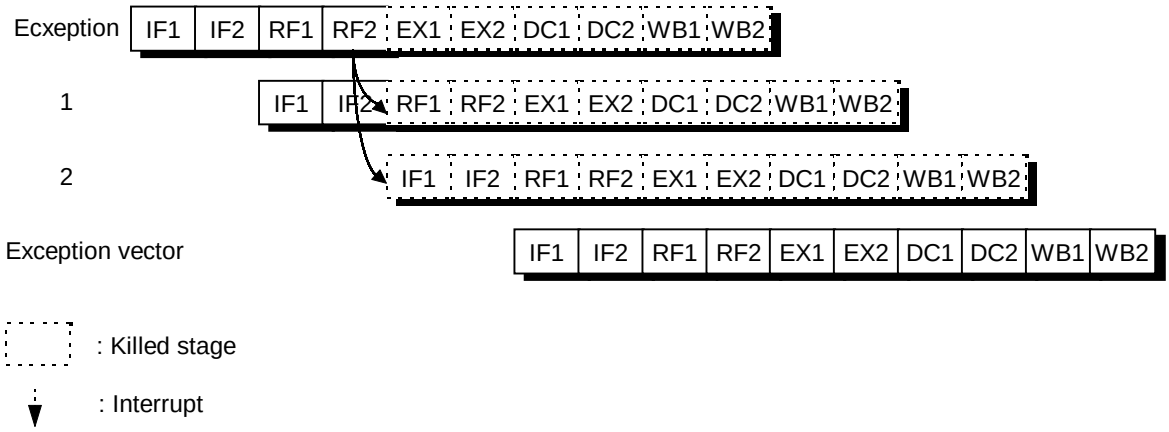
#### 2.4.5.1 Exception conditions

When an exception condition occurs, the relevant instruction and all those that follow it in the pipeline are cancelled. Accordingly, any stall conditions and any later exception conditions that may have referenced this instruction are inhibited; there is no benefit in servicing stalls for a cancelled instruction.

When an exceptional conditions is detected for an instruction, the VR4120A will discard it and all following instructions. When this instruction reaches the WB stage, the exception flag and various information items are written to CP0 registers. The current PC is changed to the appropriate exception vector address and the exception bits of earlier pipeline stages are cleared.

This implementation allows all preceding instructions to complete execution and prevents all subsequent instructions from completing. Thus the value in the EPC is sufficient to restart execution. It also ensures that exceptions are taken in the order of execution; an instruction taking an exception may itself be killed by an instruction further down the pipeline that takes an exception in a later cycle.

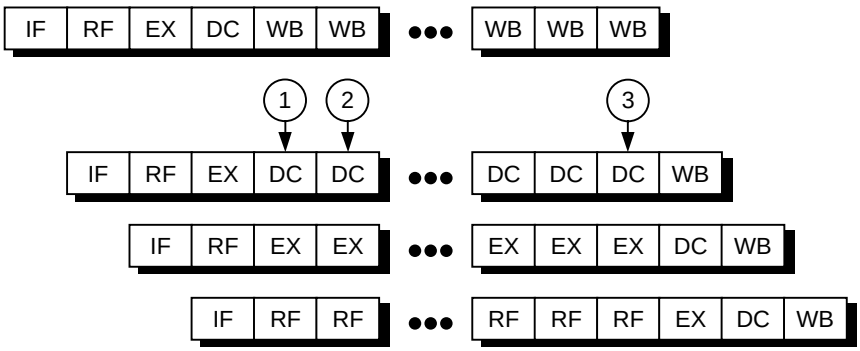
**Figure 2-30. Exception Detection**



### 2.4.5.2 Stall conditions

Stalls are used to stop the pipeline for conditions detected after the RF stage. When a stall occurs, the processor will resolve the condition and then the pipeline will continue. Figure 2-31 shows a data cache miss stall, and Figure 2-32 shows a CACHE instruction stall.

**Figure 2-31. Data Cache Miss Stall**



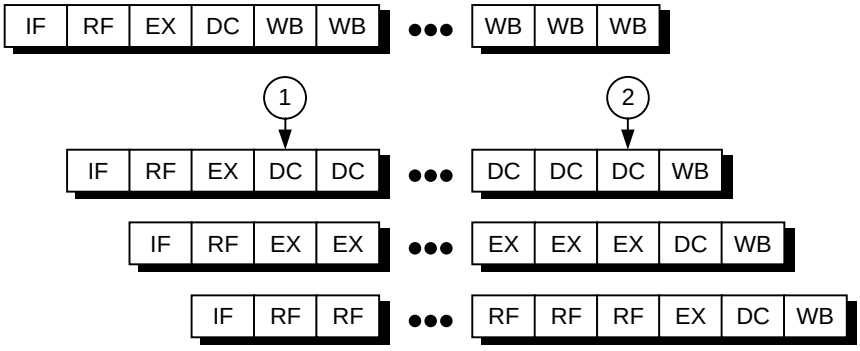

(1) Detect data cache miss

(2) Start moving data cache line to write buffer

(3) Get last word into cache and restart pipeline

If the cache line to be replaced is dirty — the W bit is set — the data is moved to the internal write buffer in the next cycle. The write-back data is returned to memory. The last word in the data is returned to the cache at 3, and pipelining restarts.

**Figure 2-32. CACHE Instruction Stall**




(1) CACHE instruction start

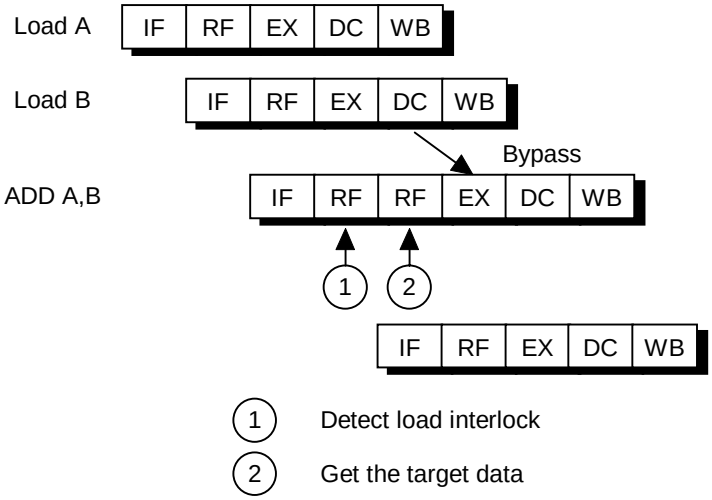
(2) CACHE instruction complete

When the CACHE instruction enters the DC stage, the pipeline stalls while the CACHE instruction is executed. The pipeline begins running again when the CACHE instruction is completed, allowing the instruction fetch to proceed.

#### 2.4.5.3 Slip conditions

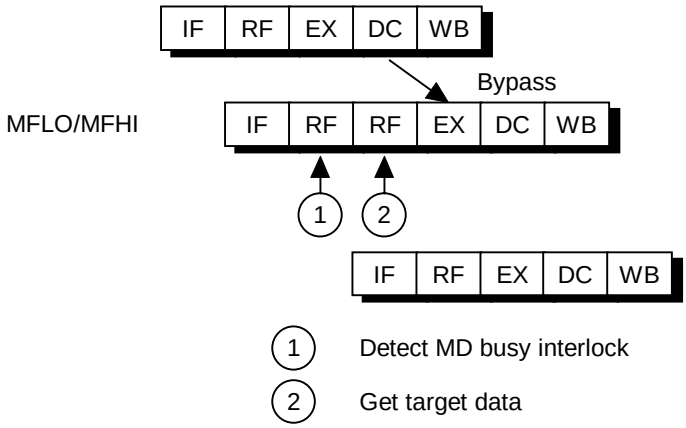
During Φ2 of the RF stage and Φ1 of the EX stage, internal logic will determine whether it is possible to start the current instruction in this cycle. If all of the source operands are available (either from the register file or via the internal bypass logic) and all the hardware resources necessary to complete the instruction will be available whenever required, then the instruction “run”; otherwise, the instruction will “slip”. Slipped instructions are retired on subsequent cycles until they issue. The backend of the pipeline (stages DC and WB) will advance normally during slips in an attempt to resolve the conflict. NOPs will be inserted into the bubble in the pipeline. Instructions killed by branch likely instructions, ERET or exceptions will not cause slips.

**Figure 2-33. Load Data Interlock**




Load Data Interlock is detected in the RF stage shown in as Figure 2-33 and also the pipeline slips in the stage. Load Data Interlock occurs when data fetched by a load instruction and data moved from HI, LO or CP0 registers is required by the next immediate instruction. The pipeline begins running again when the clock after the target of the load is read from the data cache, HI, LO and CP0 registers. The data returned at the end of the DC stage is input into the end of the RF stage, using the bypass multiplexers.

**Figure 2-34. MD Busy Interlock**




MD Busy Interlock is detected in the RF stage as shown in Figure 2-34 and also the pipeline slips in the stage. MD Busy Interlock occurs when HI/LO register is required by MFHI/MFLO instruction before finishing Mult/Div execution.

The pipeline begins running again the clock after finishing Mult/Div execution. The data returned from the HI/LO register at the end of the DC stage is input into the end of the RF stage, using the bypass multiplexers.

Store-Load Interlock is detected in the EX stage and the pipeline slips in the RF stage. Store-Load Interlock occurs when store instruction followed by load instruction is detected. The pipeline begins running again one clock after.

Coprocessor 0 Interlock is detected in the EX stage and the pipeline slips in the RF stage. A coprocessor interlock occurs when an MTC0 instruction for the Configuration or Status register is detected.

The pipeline begins running again one clock after.

#### 2.4.5.4 Bypassing

In some cases, data and conditions produced in the EX, DC and WB stages of the pipeline are made available to the EX stage (only) through the bypass data path.

Operand bypass allows an instruction in the EX stage to continue without having to wait for data or conditions to be written to the register file at the end of the WB stage. Instead, the Bypass Control Unit is responsible for ensuring data and conditions from later pipeline stages are available at the appropriate time for instructions earlier in the pipeline.

The Bypass Control Unit is also responsible for controlling the source and destination register addresses supplied to the register file.

### 2.4.6 Program compatibility

The VR4120A core is designed taking into consideration program compatibility with other VR-Series processors. However, because the VR4120A differs from other processors in its architecture, it may not be able to run some programs that run on other processors. Likewise, programs that run on the VR4120A will not necessarily run on other processors. Matters which should be paid attention to when porting programs between the VR4120A core and other VR-Series processors are listed below.

- The VR4120A core does not support floating-point instructions since it has no Floating-Point Unit (FPU).
- Multiply-add instructions (DMACC, MACC) are added in the VR4120A.
- Instructions for power modes (HIBERNATE, STANDBY, SUSPEND) are added in the VR4120A to support power modes.
- The VR4120A does not have the LL bit to perform synchronization of multiprocessing. Therefore, the CPU core does not support instructions which manipulate the LL bit (LL, LLD, SC, SCD).
- A 16-bit length MIPS16 instruction set is added in the VR4120A.
- The CP0 hazards of the VR4120A are equally or less stringent than those of other processors (for details, see **APPENDIX C VR4120A COPROCESSOR 0 HAZARDS**).
- An instruction for debug has been added for the VR4120A. However, this instruction cannot be used for the VR4120A.

For more information, refer to **Section 2.3 MIPS16 Instruction Set, APPENDIX A MIPS III INSTRUCTION SET DETAILS**, the VR4100, VR4111 User's Manual, or the VR4300™ User's Manual.

The list of instructions supported by VR-Series products is shown below.

**Table 2-47. VR Series Supported Instructions**

| Instruction \ Product | VR4100 VR4102™ | VR4111™ | VR4120A Core | VR4300 VR4305™ VR4310™ | VR5000™ VR10000™ |
|---|---|---|---|---|---|
| MIPS I instruction set | O | O | O | O | O |
| MIPS II instruction set | O | O | O | O | O |
| MIPS III instruction set | O | O | O | O | O |
| LL bit operation | × | × | × | O | O |
| MIPS IV instruction set | × | × | × | × | O |
| MIPS16 instruction set | × | O | O | × | × |
| 16-bit multiply-add operation | O | O | O (Use of 32-bit multiply-add operation | × | × |
| 32-bit multiply-add operation | × | × | O | × | × |
| Floating-point operation | × | × | × | O | O |
| Power mode transfer | O | O | O | × | × |

## 2.5 Memory Management System

The VR4120A Core provides a memory management unit (MMU) which uses a translation lookaside buffer (TLB) to translate virtual addresses into physical addresses. This chapter describes the virtual and physical address spaces, the virtual-to-physical address translation, the operation of the TLB in making these translations, and the CP0 registers that provide the software interface to the TLB.

### 2.5.1 Translation lookaside buffer (TLB)

Virtual addresses are translated into physical addresses using an on-chip TLB[Note]. The on-chip TLB is a fully-associative memory that holds 32 entries, which provide mapping to odd/even page in pairs for one entry. These pages can have five different sizes, 1 K, 4 K, 16 K, 64 K, and 256 K, and can be specified in each entry. If it is supplied with a virtual address, each of the TLB entries is checked simultaneously to see whether they match the virtual addresses that are provided with the ASID field and saved in the EntryHi register.

If there is a virtual address match, or “hit,” in the TLB, the physical page number is extracted from the TLB and concatenated with the offset to form the physical address.

If no match occurs (TLB “miss”), an exception is taken and software refills the TLB from the page table resident in memory. The software writes to an entry selected using the Index register or a random entry indicated in the Random register.

If more than one entry in the TLB matches the virtual address being translated, the operation is undefined and the TLB may be disabled. In this case, the TLB-Shutdown (TS) bit of the status register is set to 1, and the TLB becomes unusable (an attempt to access the TLB results in a TLB Mismatch exception regardless of whether there is an entry that hits). The TS bit can be cleared only by a reset.

**Note** Depending on the address space, virtual addresses may be converted to physical addresses without using a TLB. For example, address translation for the kseg0 or kseg1 address space does not use mapping. The physical addresses of these address spaces are determined by subtracting the base address of the address space from the virtual addresses.

### 2.5.2 Virtual address space

This section describes the virtual/physical address space and the manner in which virtual addresses are converted or “translated” into physical addresses in the TLB. The VR4120A virtual address can be either 32 or 64 bits wide, depending on whether the processor is operating in 32-bit or 64-bit mode.

In 32-bit mode, addresses are 32 bits wide. The maximum user process size is 2 Gbytes ($2^{31}$).
In 64-bit mode, addresses are 64 bits wide. The maximum user process size is 1 Tbyte ($2^{40}$).

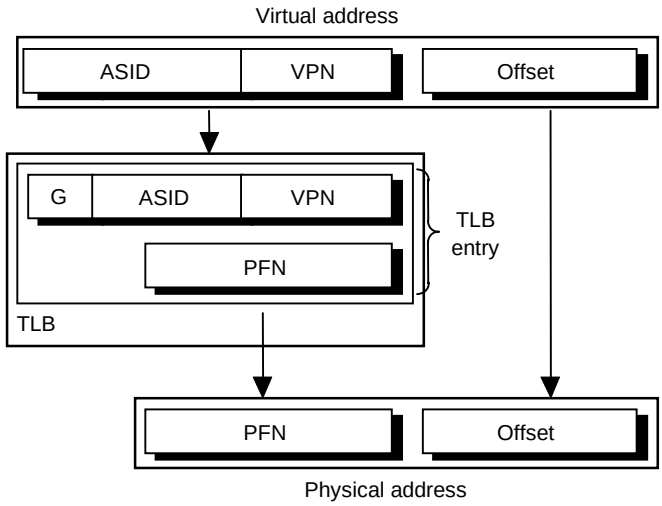
As shown in Figure 2-35, the virtual address is extended with an address space identifier (ASID), which reduces the frequency of TLB flushing when switching contexts. This 8-bit ASID is in the CP0 EntryHi register, described later in this chapter. The Global (G) bit is in the EntryLo0 and EntryLo1 registers, described later in this section.

**Figure 2-35. Virtual-to-Physical Address Translation**

1 The virtual page number (VPN) in the virtual address (VA) is compared with the VPN in the TLB.

2 If there is a match, the page frame number (PFN) representing the high-order bits of the physical address is output from the TLB.

3 The offset is then added to the PFN passing through the TLB.




#### 2.5.2.1 Virtual-to-physical address translation

Converting a virtual address to a physical address begins by comparing the virtual address from the processor with the virtual addresses in the TLB; there is a match when the virtual page number (VPN) of the address is the same as the VPN field of the entry, and either:

- ✧ the Global (G) bit of the TLB entry is set to 1
- ✧ the ASID field of the virtual address is the same as the ASID field of the TLB entry.

This match is referred to as a TLB hit. If there is no match, a TLB Mismatch exception is taken by the processor and software is allowed to refill the TLB from a page table of virtual/physical addresses in memory.

If there is a virtual address match in the TLB, the physical address is output from the TLB and concatenated with the offset, which represents an address within the page frame space. The offset does not pass through the TLB. Instead, the low-order bits of the virtual address are output without being translated. For details about the physical address, see **Section 2.5.5.11 Virtual-to-physical address translation**.

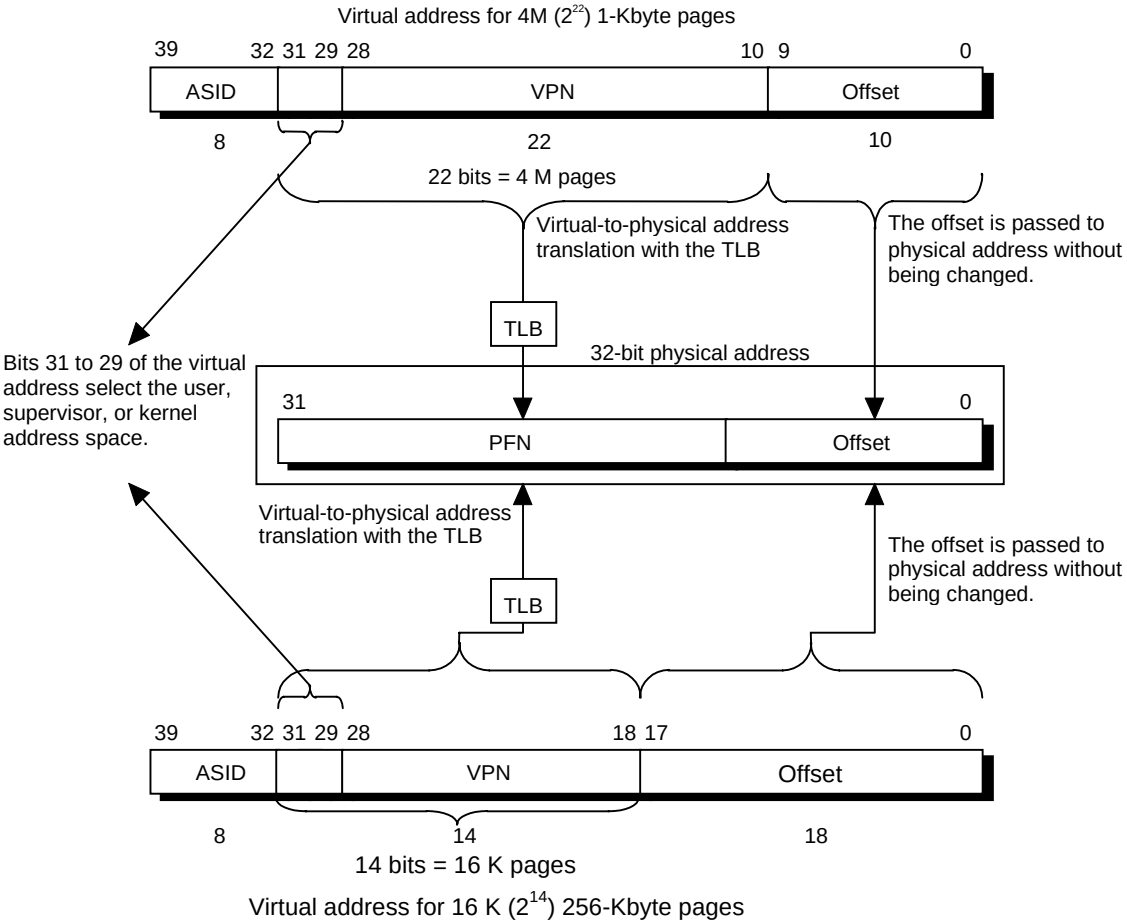
The next two sections describe the 32-bit and 64-bit mode address translations.

#### 2.5.2.2 32-bit mode address translation

Figure 2-36 shows the virtual-to-physical-address translation of a 32-bit mode address. The pages can have five different sizes between 1 Kbyte (10 bits) and 256 Kbytes (18 bits), each being 4 times as large as the preceding one in ascending order, that is 1 K, 4 K, 16 K, 64 K, and 256 K.

- ✧ Shown at the top of Figure 2-36 is the virtual address space in which the page size is 1 Kbyte and the offset is 10 bits. The 22 bits excluding the ASID field represents the virtual page number (VPN), enabling selecting a page table of 4 M entries.
- ✧ Shown at the bottom of Figure 2-36 is the virtual address space in which the page size is 256 Kbytes and the offset is 18 bits. The 14 bits excluding the ASID field represents the VPN, enabling selecting a page table of 16 K entries.

**Figure 2-36. 32-bit Mode Virtual Address Translation**



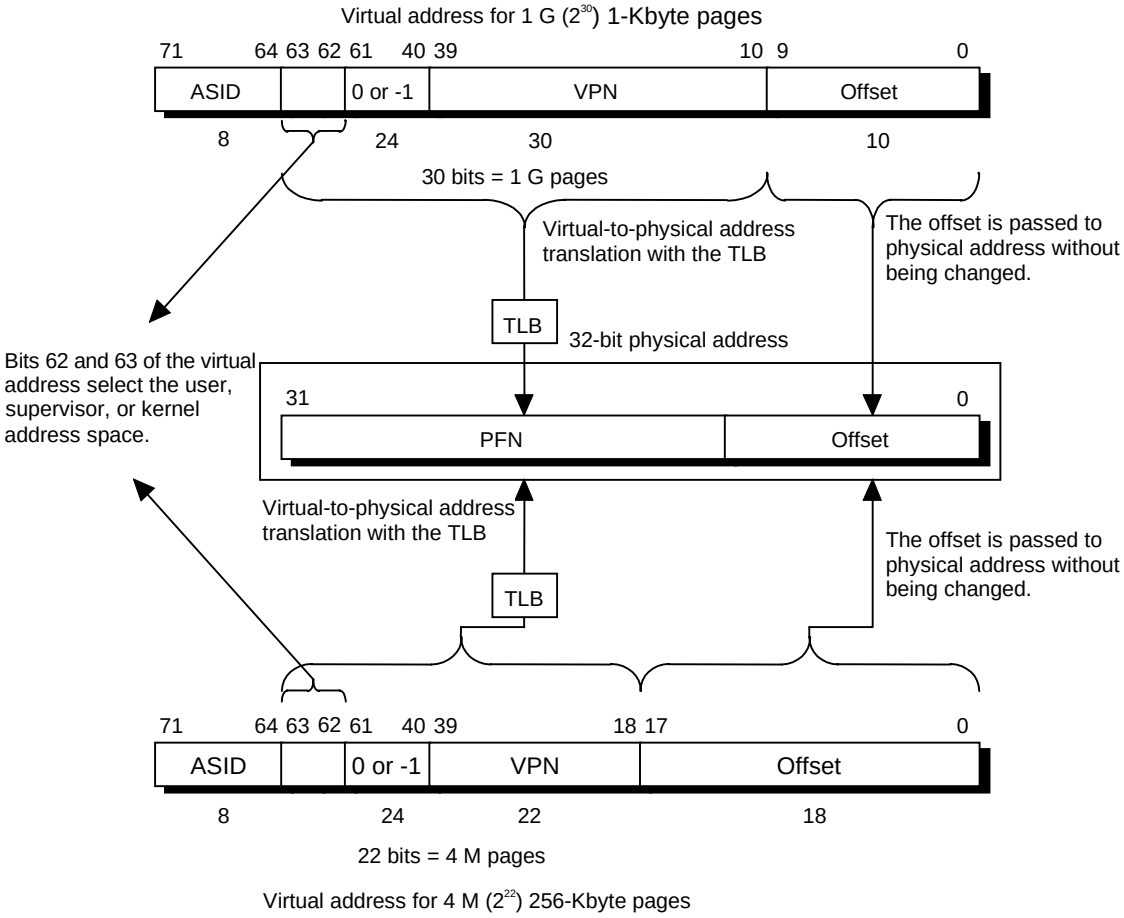

#### 2.5.2.3 64-bit mode address translation

Figure 2-37 shows the virtual-to-physical-address translation of a 64-bit mode address. This figure illustrates the two possible page size; a 1-Kbyte page (10 bits) and a 256-Kbyte page (18 bits).

- ✧ Shown at the top of Figure 2-37 is the virtual address space in which the page size is 1 Kbyte and the offset is 10 bits. The 30 bits excluding the ASID field represents the virtual page number (VPN), enabling selecting a page table of 1 G entry.

✧ Shown at the bottom of Figure 2-37 is the virtual address space in which the page size is 256 Kbytes and the offset is 18 bits. The 22 bits excluding the ASID field represents the VPN, enabling selecting a page table of 4 M entries.

**Figure 2-37. 64-bit Mode Virtual Address Translation**




### 2.5.2.4 Operating modes

The processor has three operating modes that function in both 32- and 64-bit operations:

✧ User mode
✧ Supervisor mode
✧ Kernel mode

User and Kernel modes are common to all VR-Series processors. Generally, Kernel mode is used to execute the operating system, while User mode is used to run application programs. The VR4000 series processors have a third mode, which is called Supervisor mode and categorized in between User and Kernel modes. This mode is used to configure a high-security system.

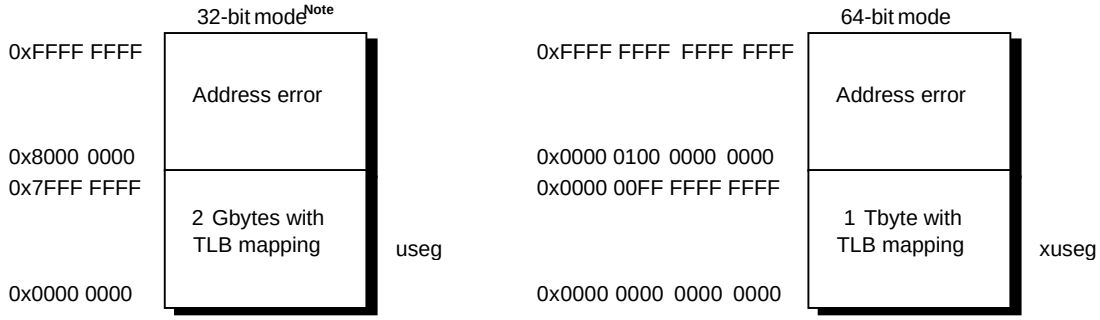
When an exception occurs, the CPU enters Kernel mode, and remains in this mode until an exception return instruction (ERET) is executed. The ERET instruction brings back the processor to the mode in which it was just before the exception occurs.

### 2.5.2.5 User mode virtual addressing

In user mode, a single virtual address space labeled User segment is available ; its size is

- ✧ 2-Gbyte ($2^{31}$ bytes) in 32-bit mode (useg)
- ✧ 1-Tbyte ($2^{40}$ bytes) in 64-bit mode (xuseg)

**Figure 2-38. User Mode Address Space**




**Note** The VR4120A uses 64-bit addresses within it. When the processor is running in Kernel mode, it saves the contents of each register or restores their previous contents to initialize them before switching the context. For 32-bit mode addressing, bit 31 is sign-extended to bits 32 to 63, and the resulting 32 bits are used for addressing. Usually, it is impossible for 32-bit mode programs to generate invalid addresses. If context switching occurs and the processor enters Kernel mode, however, an attempt may be made to save an address other than the sign-extended 32-bit address mentioned above to a 64-bit register. In this case, user-mode programs are likely to generate an invalid address.

The User segment starts at address 0 and the current active user process resides in either useg (in 32-bit mode) or xuseg (in 64-bit mode). The TLB identically maps all references to useg/xuseg from all modes, and controls cache accessibility.

The processor operates in User mode when the Status register contains the following bit-values:

- ✧ KSU = 10
- ✧ EXL = 0
- ✧ ERL = 0

In conjunction with these bits, the UX bit in the Status register selects addressing mode as follows:

- ✧ When UX = 0, 32-bit useg space is selected.
- ✧ When UX = 1, 64-bit xuseg space is selected.

Table 2-48 lists the characteristics of each user segment (useg and xuseg).

**Table 2-48. Comparison of useg and xuseg**

| Address Bit Value | Status Register Bit Value | | | | Segment Name | Address Range | Size |
|---|---|---|---|---|---|---|---|
| | KSU | EXL | ERL | UX | | | |
| 32-bit<br>A31 = 0 | 10 | 0 | 0 | 0 | useg | 0x0000 0000<br>to<br>0x7FFF FFFF | 2 Gbytes<br>($2^{31}$ bytes) |
| 64-bit<br>A(63:40) = 0 | 10 | 0 | 0 | 1 | xuseg | 0x0000 0000 0000 0000<br>to<br>0x0000 00FF FFFF FFFF | 1 Tbyte<br>($2^{40}$ bytes) |

**(1) useg (32-bit mode)**

In User mode, when UX = 0 in the Status register and the most significant bit of the virtual address is 0, this virtual address space is labeled useg.

Any attempt to reference an address with the most-significant bit set while in User mode causes an Address Error exception (see **Section 2.6 Exception Processing**).

The TLB Mismatch exception vector is used for TLB misses.

**(2) xuseg (64-bit mode)**

In User mode, when UX = 1 in the Status register and bits 63 to 40 of the virtual address are all 0, this virtual address space is labeled xuseg.

Any attempt to reference an address with bits 63 to 40 equal to 1 causes an Address Error exception (see **Section 2.6 Exception Processing**).

The XTLB Mismatch exception vector is used for TLB misses.

#### 2.5.2.6 Supervisor-mode virtual addressing

Supervisor mode shown in Figure 2-39 is designed for layered operating systems in which a true kernel runs in Kernel mode, and the rest of the operating system runs in Supervisor mode.

The processor operates in Supervisor mode when the Status register contains the following bit-values:

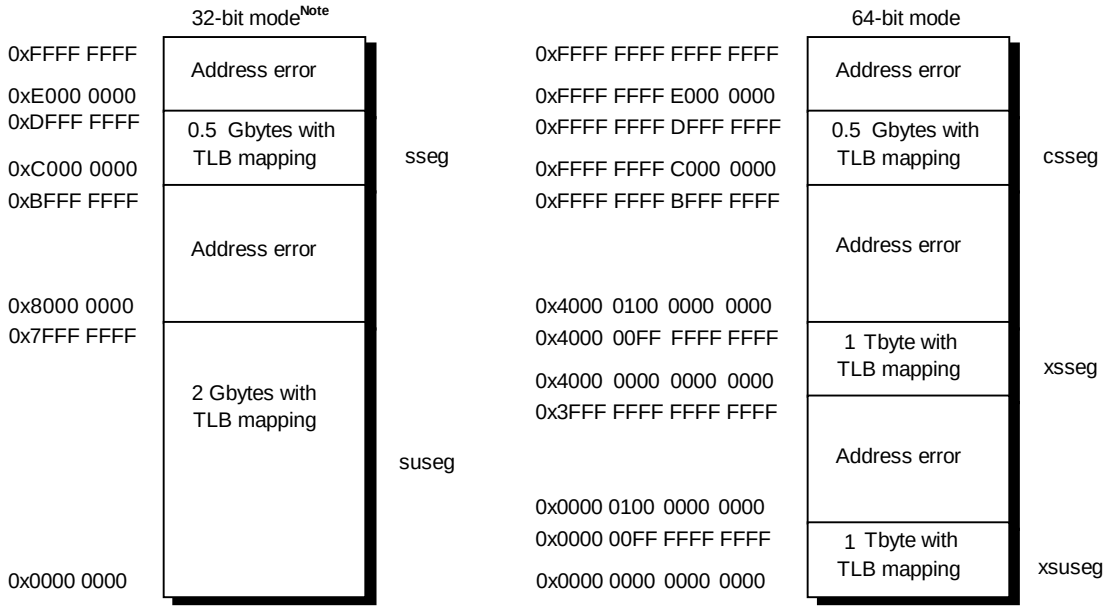
- ✧ KSU = 01
- ✧ EXL = 0
- ✧ ERL = 0

In conjunction with these bits, the SX bit in the Status register selects Supervisor mode addressing:

- ✧ When SX = 0: 32-bit supervisor space is selected.
- ✧ When SX = 1: 64-bit supervisor space is selected.

Figure 2-39 shows the supervisor mode address mapping, and Table 2-49 lists the characteristics of the Supervisor mode segments.

**Figure 2-39. Supervisor Mode Address Space**




**Note** The VR4120A uses 64-bit addresses within it. For 32-bit mode addressing, bit 31 is sign-extended to bits 32 to 63, and the resulting 32 bits are used for addressing. Usually, it is impossible for 32-bit mode programs to generate invalid addresses. In an operation of base register + offset for addressing, however, a two's complement overflow may occur, causing an invalid address. Note that the result becomes undefined. Two factors that can cause a two's complement follow:

- ✧ When offset bit 15 is 0, base register bit 31 is 0, and bit 31 of the operation "base register + offset" is 1
- ✧ When offset bit 15 is 1, base register bit 31 is 1, and bit 31 of the operation "base register + offset" is 0

**Table 2-49. 32-bit and 64-bit Supervisor Mode Segments**

| Address Bit Value | Status Register Bit Value | | | | Segment Name | Address Range | Size |
|---|---|---|---|---|---|---|---|
| | KSU | EXL | ERL | SX | | | |
| 32-bit<br>A31 = 0 | 01 | 0 | 0 | 0 | suseg | 0x0000 0000<br>to<br>0x7FFF FFFF | 2 Gbytes<br>($2^{31}$ bytes) |
| 32-bit<br>A(31:29) = 110 | 01 | 0 | 0 | 0 | sseg | 0xC000 0000<br>to<br>0xDFFF FFFF | 512 Mbytes<br>($2^{29}$ bytes) |
| 64-bit<br>A(63:62) = 00 | 01 | 0 | 0 | 1 | xsuseg | 0x0000 0000 0000 0000<br>to<br>0x0000 00FF FFFF FFFF | 1 Tbyte<br>($2^{40}$ bytes) |
| 64-bit<br>A(63:62) = 01 | 01 | 0 | 0 | 1 | xsseg | 0x4000 0000 0000 0000<br>to<br>0x4000 00FF FFFF FFFF | 1 Tbyte<br>($2^{40}$ bytes) |
| 64-bit<br>A(63:62) = 11 | 01 | 0 | 0 | 1 | csseg | 0xFFFF FFFF C000 0000<br>to<br>0xFFFF FFFF DFFF FFFF | 512 Mbytes<br>($2^{29}$ bytes) |

**(1) suseg (32-bit supervisor mode, user space)**

When SX = 0 in the Status register and the most-significant bit of the virtual address space is set to 0, the suseg virtual address space is selected; it covers 2 Gbytes ($2^{31}$ bytes) of the current user address space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address. This mapped space starts at virtual address 0x0000 0000 and runs through 0x7FFF FFFF.

**(2) sseg (32-bit supervisor mode, supervisor space)**

When SX = 0 in the Status register and the most-significant three bits of the virtual address space are 110, the sseg virtual address space is selected; it covers 512 Mbytes ($2^{29}$ bytes) of the current supervisor virtual address space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address. This mapped space begins at virtual address 0xC000 0000 and runs through 0xDFFF FFFF.

**(3) xsuseg (64-bit supervisor mode, user space)**

When SX = 1 in the Status register and bits 63 and 62 of the virtual address space are set to 00, the xsuseg virtual address space is selected; it covers 1 Tbyte ($2^{40}$ bytes) of the current user address space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address. This mapped space starts at virtual address 0x0000 0000 0000 0000 and runs through 0x0000 00FF FFFF FFFF.

**(4) xsseg (64-bit supervisor mode, current supervisor space)**

When SX = 1 in the Status register and bits 63 and 62 of the virtual address space are set to 01, the xsseg virtual address space is selected; it covers 1 Tbyte ($2^{40}$ bytes) of the current supervisor virtual address space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address. This mapped space begins at virtual address 0x4000 0000 0000 0000 and runs through 0x4000 00FF FFFF FFFF.

**(5) csseg (64-bit supervisor mode, separate supervisor space)**

When SX = 1 in the Status register and bits 63 and 62 of the virtual address space are set to 11, the csseg virtual address space is selected; it covers 512 Mbytes ($2^{29}$ bytes) of the separate supervisor virtual address space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address. This mapped space begins at virtual address 0xFFFF FFFF C000 0000 and runs through 0xFFFF FFFF DFFF FFFF.

#### 2.5.2.7 Kernel-mode virtual addressing

If the Status register satisfies any of the following conditions, the processor runs in Kernel mode.

- ✧ KSU = 00
- ✧ EXL = 1
- ✧ ERL = 1

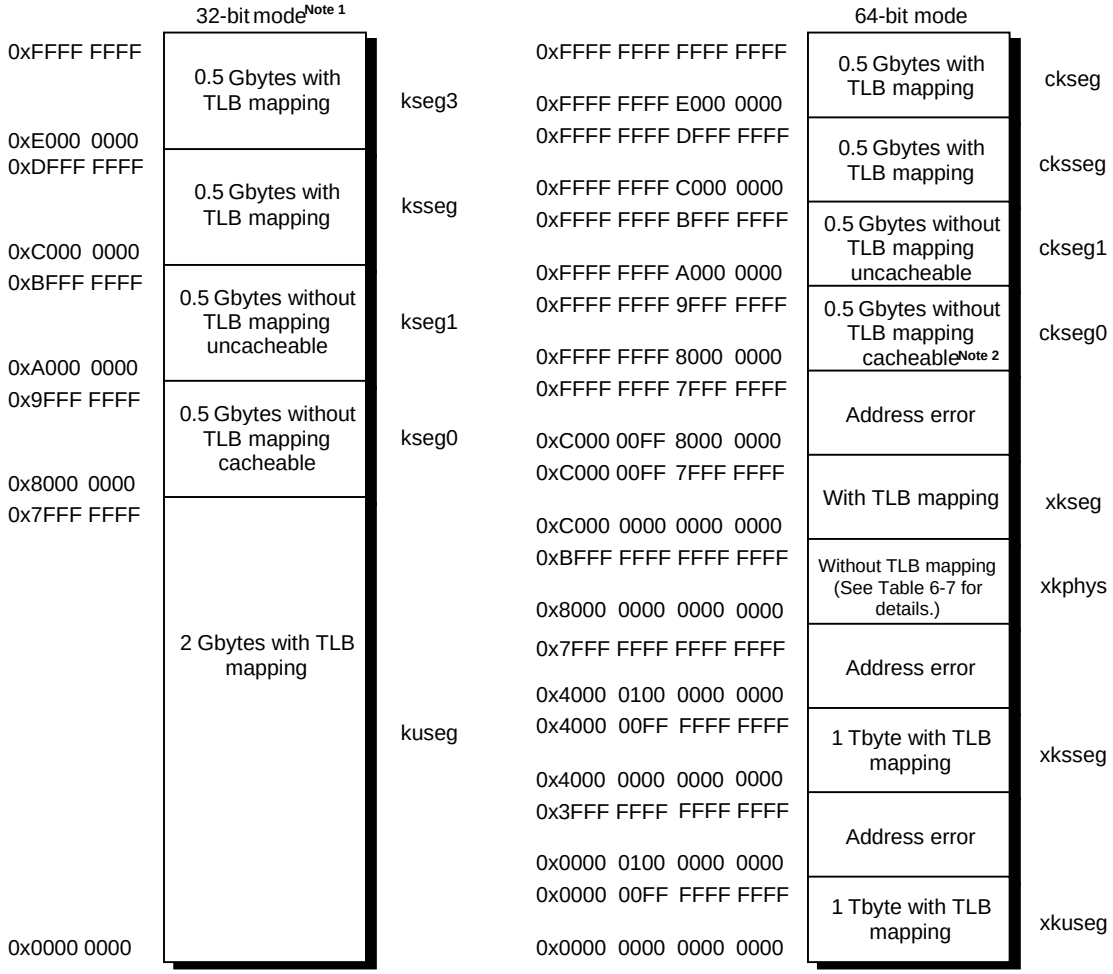
The addressing width in Kernel mode varies according to the state of the KX bit of the Status register, as follows:

- ✧ When KX = 0: 32-bit kernel space is selected.
- ✧ When KX = 1: 64-bit kernel space is selected.

The processor enters Kernel mode whenever an exception is detected and it remains in Kernel mode until an exception return (ERET) instruction is executed and results in ERL and/or EXL = 0. The ERET instruction restores the processor to the mode existing prior to the exception.

Kernel mode virtual address space is divided into regions differentiated by the high-order bits of the virtual address, as shown in Figure 2-40. Table 2-50 lists the characteristics of the 32-bit Kernel mode segments, and Table 2-51 lists the characteristics of the 64-bit Kernel mode segments.

**Figure 2-40. Kernel Mode Address Space**




**Notes 1.** The VR4120A uses 64-bit addresses within it. For 32-bit mode addressing, bit 31 is sign-extended to bits 32 to 63, and the resulting 32 bits are used for addressing. Usually, a 64-bit instruction is used for the program in 32-bit mode.

**2.** The K0 field of the Config register controls cacheability of kseg0 and ckseg0.

**Table 2-50. 32-bit Kernel Mode Segments**

| Address Bit Value | Status Register Bit Value | | | | Segment Name | Virtual Address | Physical Address | Size |
|---|---|---|---|---|---|---|---|---|
| | KSU | EXL | ERL | KX | | | | |
| A31 = 0 | KSU = 00 or EXL = 1 or ERL = 1 | | | 0 | kuseg | 0x0000 0000 to 0x7FFF FFFF | TLB map | 2 Gbytes ($2^{31}$ bytes) |
| A(31:29) = 100 | | | | 0 | kseg0 | 0x8000 0000 to 0x9FFF FFFF | 0x0000 0000 to 0x1FFF FFFF | 512 Mbytes ($2^{29}$ bytes) |
| A(31:29) = 101 | | | | 0 | kseg1 | 0xA000 0000 to 0xBFFF FFFF | 0x0000 0000 to 0x1FFF FFFF | 512 Mbytes ($2^{29}$ bytes) |
| A(31:29) = 110 | | | | 0 | ksseg | 0xC000 0000 to 0xDFFF FFFF | TLB map | 512 Mbytes ($2^{29}$ bytes) |
| A(31:29) = 111 | | | | 0 | kseg3 | 0xE000 0000 to 0xFFFF FFFF | TLB map | 512 Mbytes ($2^{29}$ bytes) |

**(1) kuseg (32-bit kernel mode, user space)**

When KX = 0 in the Status register, and the most-significant bit of the virtual address space is 0, the kuseg virtual address space is selected; it is the current 2-Gbyte ($2^{31}$-byte) user address space.

The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

If the ERL bit of the Status register is 1, the user address space is assigned 2 Gbytes ($2^{31}$ bytes) without TLB mapping and becomes unmapped (with virtual addresses being used as physical addresses) and uncached so that the cache error handler can use it. This allows the Cache Error exception code to operate uncached using r0 as a base register.

**(2) kseg0 (32-bit kernel mode, kernel space 0)**

When KX = 0 in the Status register and the most-significant three bits of the virtual address space are 100, the kseg0 virtual address space is selected; it is the current 512-Mbyte ($2^{29}$-byte) physical space.

References to kseg0 are not mapped through TLB; the physical address selected is defined by subtracting 0x8000 0000 from the virtual address.

The K0 field of the Config register controls cacheability (see **Section 2.6 Exception Processing**).

**(3) kseg1 (32-bit kernel mode, kernel space 1)**

When KX = 0 in the Status register and the most-significant three bits of the virtual address space are 101, the kseg1 virtual address space is selected; it is the current 512-Mbyte ($2^{29}$-byte) physical space.

References to kseg1 are not mapped through TLB; the physical address selected is defined by subtracting 0xA000 0000 from the virtual address.

Caches are disabled for accesses to these addresses, and main memory (or memory-mapped I/O device registers) is accessed directly.

**(4) ksseg (32-bit kernel mode, supervisor space)**

When KX = 0 in the Status register and the most-significant three bits of the virtual address space are 110, the ksseg virtual address space is selected; it is the current 512-Mbyte ($2^{29}$-byte) virtual address space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

**(5) kseg3 (32-bit kernel mode, kernel space 3)**

When KX = 0 in the Status register and the most-significant three bits of the virtual address space are 111, the kseg3 virtual address space is selected; it is the current 512-Mbyte ($2^{29}$-byte) kernel virtual space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

**Table 2-51. 64-bit Kernel Mode Segments**

| Address Bit Value | Status Register Bit Value | | | | Segment Name | Virtual Address | Physical Address | Size |
|---|---|---|---|---|---|---|---|---|
| | KSU | EXL | ERL | KX | | | | |
| A(63:62) = 00 | KSU = 00 or EXL = 1 or ERL = 1 | | | 1 | xkuseg | 0x0000 0000 0000 0000 to 0x0000 00FF FFFF FFFF | TLB map | 1 Tbyte ($2^{40}$ bytes) |
| A(63:62) = 01 | | | | 1 | xksseg | 0x4000 0000 0000 0000 to 0x4000 00FF FFFF FFFF | TLB map | 1 Tbyte ($2^{40}$ bytes) |
| A(63:62) = 10 | | | | 1 | xkphys | 0x8000 0000 0000 0000 to 0xBFFF FFFF FFFF FFFF | 0x0000 0000 to 0xFFFF FFFF | 4 Gbytes ($2^{32}$ bytes) |
| A(63:62) = 11 | | | | 1 | xkseg | 0xC000 0000 0000 0000 to 0xC000 00FF 7FFF FFFF | TLB map | $2^{40}$ to $2^{31}$ bytes |
| A(63:62) = 11 A(63:31) = -1 | | | | 1 | ckseg0 | 0xFFFF FFFF 8000 0000 to 0xFFFF FFFF 9FFF FFFF | 0x0000 0000 to 0x1FFF FFFF | 512 Mbytes ($2^{29}$ bytes) |
| A(63:62) = 11 A(63:31) = -1 | | | | 1 | ckseg1 | 0xFFFF FFFF A000 0000 to 0xFFFF FFFF BFFF FFFF | 0x0000 0000 to 0x1FFF FFFF | 512 Mbytes ($2^{29}$ bytes) |
| A(63:62) = 11 A(63:31) = -1 | | | | 1 | cksseg | 0xFFFF FFFF C000 0000 to 0xFFFF FFFF DFFF FFFF | TLB map | 512 Mbytes ($2^{29}$ bytes) |
| A(63:62) = 11 A(63:31) = -1 | | | | 1 | ckseg3 | 0xFFFF FFFF E000 0000 to 0xFFFF FFFF FFFF FFFF | TLB map | 512 Mbytes ($2^{29}$ bytes) |

**(6) xkuseg (64-bit kernel mode, user space)**

When KX = 1 in the Status register and bits 63 and 62 of the virtual address space are 00, the xkuseg virtual address space is selected; it is the 1-Tbyte ($2^{40}$ bytes) current user address space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

If the ERL bit of the Status register is 1, the user address space is assigned 2 Gbytes ($2^{31}$ bytes) without TLB mapping and becomes unmapped (with virtual addresses being used as physical addresses) and uncached so that the cache error handler can use it. This allows the Cache Error exception code to operate uncached using r0 as a base register.

**(7) xksseg (64-bit kernel mode, current supervisor space)**

When KX = 1 in the Status register and bits 63 and 62 of the virtual address space are 01, the xksseg address space is selected; it is the 1-Tbyte ($2^{40}$ bytes)current supervisor address space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

**(8) xkphys (64-bit kernel mode, physical spaces)**

When the KX = 1 in the Status register and bits 63 and 62 of the virtual address space are 10, the virtual address space is called xkphys and selected as either cached or uncached. If any of bits 58 to 32 of the address is 1, an attempt to access that address results in an address error.

**Table 2-52. Cacheability and xkphys Address Space**

| Bits 61-59 | Cacheability | Start Address |
|---|---|---|
| 0 | Cached | 0x8000 0000 0000 0000<br>to<br>0x8000 0000 FFFF FFFF |
| 1 | Cached | 0x8800 0000 0000 0000<br>to<br>0x8800 0000 FFFF FFFF |
| 2 | Uncached | 0x9000 0000 0000 0000<br>to<br>0x9000 0000 FFFF FFFF |
| 3 | Cached | 0x9800 0000 0000 0000<br>to<br>0x9800 0000 FFFF FFFF |
| 4 | Cached | 0xA000 0000 0000 0000<br>to<br>0xA000 0000 FFFF FFFF |
| 5 | Cached | 0xA800 0000 0000 0000<br>to<br>0xA800 0000 FFFF FFFF |
| 6 | Cached | 0xB000 0000 0000 0000<br>to<br>0xB000 0000 FFFF FFFF |
| 7 | Cached | 0xB800 0000 0000 0000<br>to<br>0xB800 0000 FFFF FFFF |

**(9) xkseg (64-bit kernel mode, physical spaces)**

When the KX = 1 in the Status register and bits 63 and 62 of the virtual address space are 11, the virtual address space is called xkseg and selected as either of the following:

- Kernel virtual space xkseg, the current kernel virtual space; the virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address
  This space is referenced via TLB. Whether cache can be used or not is determined by bit C of each page's TLB entry.
- One of the four 32-bit kernel compatibility spaces, as described in the next section.

**(10) 64-bit kernel mode compatible spaces (ckseg0, ckseg1, cksseg, and ckseg3)**

If the conditions listed below are satisfied in Kernel mode, ckseg0, ckseg1, cksseg, or ckseg3 (each having 512 Mbytes) is selected as a compatible space according to the state of the bits 30 and 29 (two low-order bits) of the address.

✧ The KX bit of the Status register is 1.
✧ Bits 63 and 62 of the 64-bit virtual address are 11.
✧ Bits 61 to 31 of the virtual address are all 1.

**(a) ckseg0**

This space is an unmapped region, compatible with the 32-bit mode kseg0 space. The K0 field of the Config register controls cacheability and coherency.

**(b) ckseg1**

This space is an unmapped and uncached region, compatible with the 32-bit mode kseg1 space.

**(c) cksseg**

This space is the current supervisor virtual space, compatible with the 32-bit mode ksseg space.
References to cksseg are mapped through TLB. Whether cache can be used or not is determined by bit C of each page's TLB entry.

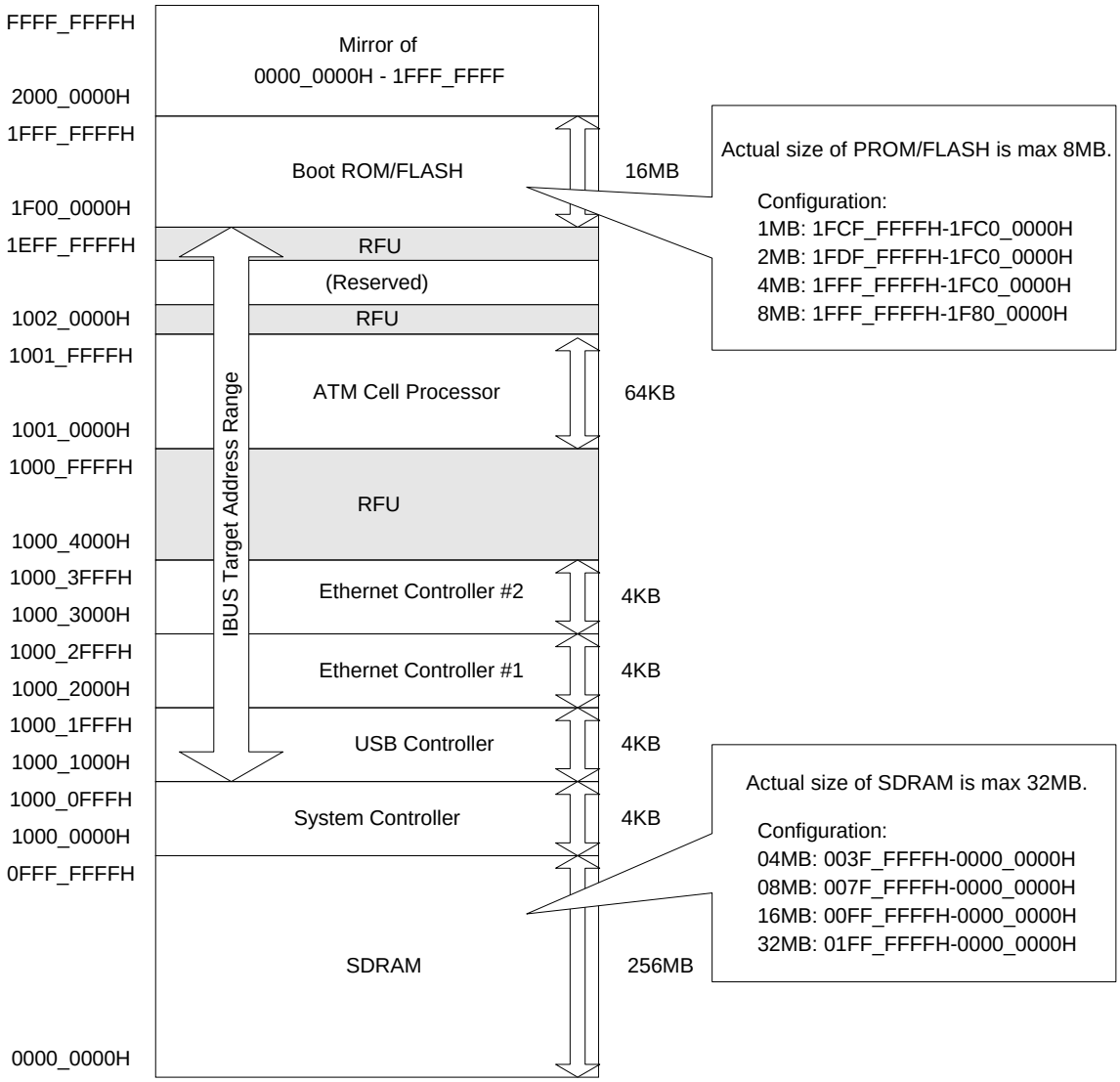
**(d) ckseg3**

This space is the current supervisor virtual space, compatible with the 32-bit mode kseg3 space.
References to ckseg3 are mapped through TLB. Whether cache can be used or not is determined by bit C of each page's TLB entry.

### 2.5.3 Physical address space

So VR4120A core uses a 32-bit address, that the processor physical address space encompasses 4 Gbytes. The VR4120A uses this 4-Gbyte physical address space as shown in Figure 2-41.

**Figure 2-41. μPD98501 Physical Address Space**



#### 2.5.3.1 ROM space

The ROM space differs depending on the data bus' bit width and the capacity of the ROM being used.

- The data bus' bit width is set via the DBUS32 pin.
- The ROM capacity is set via the BCUCNTREG's ROM64 bit and EXT_ROM64 bit.

The physical addresses of the ROM space are listed below.

**Table 2-53. ROM Addresses (When Using 16-bit Data Bus) (1/3)**

| Physical address | ADD[25:0] pin | When using 32-M ROM (DBUS32 = 0, ROM64 = 0) | When using 64-M ROM (DBUS32 = 0, ROM64 = 1) |
|---|---|---|---|
| 0x1FFF FFFF to 0x1FC0 0000 | 0x3FF FFFF to 0x3C0 0000 | Bank 3 (ROMCS_B[3]) | Bank 3 (ROMCS_B[3]) |
| 0x1FBF FFFF to 0x1F80 0000 | 0x3BF FFFF to 0x380 0000 | Bank 2 (ROMCS_B[2]) | |
| 0x1F7F FFFF to 0x1F40 0000 | 0x37F FFFF to 0x340 0000 | Bank 1 (ROMCS_B[1]) | Bank 2 (ROMCS_B[2]) |
| 0x1F3F FFFF to 0x1F00 0000 | 0x33F FFFF to 0x300 0000 | Bank 0 (ROMCS_B[0]) | |
| 0x1EFF FFFF to 0x1E80 0000 | 0x2FF FFFF to 0x280 0000 | ROM space reserved for future use | Bank 1 (ROMCS_B[1]) |
| 0x1E7F FFFF to 0x1E00 0000 | 0x27F FFFF to 0x200 0000 | | Bank 0 (ROMCS_B[0]) |
| 0x1DFF FFFF to 0x1800 0000 | 0x1FF FFFF to 0x000 0000 | | ROM space reserved for future use |

**Table 2-53. ROM Addresses (When Using 32-bit Data Bus) (2/3)**

| Physical address | ADD[25:0] pin | When using 32-M ROM and 32-M extension ROM (DBUS32 = 1, ROM64 = 0, EXT_ROM64 = 0) | When using 64-M ROM and 32-M extension ROM (DBUS32 = 1, ROM64 = 1, EXT_ROM64 = 0) |
|---|---|---|---|
| 0x1FFFFFFF to 0x1F800000 | 0x3FFFFFFF to 0x3800000 | Bank 1 (ROMCS_B[1]) | Bank 1 (ROMCS_B[1]) |
| 0x1F7FFFFF to 0x1F000000 | 0x37FFFFFF to 0x3000000 | Bank 0 (ROMCS_B[0]) | |
| 0x1EFFFFF to 0x1E800000 | 0x2FFFFFF to 0x2800000 | Bank 3 (ROMCS_B[3]) | Bank 0 (ROMCS_B[0]) |
| 0x1E7FFFFF to 0x1E000000 | 0x27FFFFF to 0x2000000 | Bank 2 (ROMCS_B[2]) | |
| 0x1DFFFFFF to 0x1D800000 | 0x1FFFFFF to 0x1800000 | ROM space reserved for future use | Bank 3 (ROMCS_B[3]) Note |
| 0x1D7FFFFF to 0x1D000000 | 0x17FFFFF to 0x1000000 | | Bank 2 (ROMCS_B[2]) Note |
| 0x1CFFFFFF to 0x18000000 | 0x0FFFFFF to 0x0000000 | | ROM space reserved for future use |

**Note** Can be used exclusively from the extension DRAM

**Table 2-53. ROM Addresses (When Using 32-bit Data Bus) (3/3)**

| Physical address | ADD[25:0] pin | When using 32-M ROM and 64-M extension ROM (DBUS32 = 1, ROM64 = 0, EXT_ROM64 = 1) | When using 64-M ROM and 64-M extension ROM (DBUS32 = 1, ROM64 = 1, EXT_ROM64 = 1) |
|---|---|---|---|
| 0x1FFFFFFF to 0x1F800000 | 0x3FFFFFFF to 0x3800000 | Bank 1 (ROMCS_B[1]) | Bank 1 (ROMCS_B[1]) |
| 0x1F7FFFFF to 0x1F000000 | 0x37FFFFFF to 0x3000000 | Bank 0 (ROMCS_B[0]) | |
| 0x1EFFFFFF to 0x1E000000 | 0x2FFFFFF to 0x2000000 | Bank 3 (ROMCS_B[3]) | Bank 0 (ROMCS_B[0]) |
| 0x1DFFFFFF to 0x1D000000 | 0x1FFFFFF to 0x1000000 | Bank 2 (ROMCS_B[2]) | Bank 3 (ROMCS_B[3]) Note |
| 0x1CFFFFFF to 0x1C000000 | 0x0FFFFFF to 0x0000000 | ROM space reserved for future use | Bank 2 (ROMCS_B[2]) Note |
| 0x1BFFFFFF to 0x18000000 | 0x3FFFFFF to 0x0000000 | | ROM space reserved for future use |

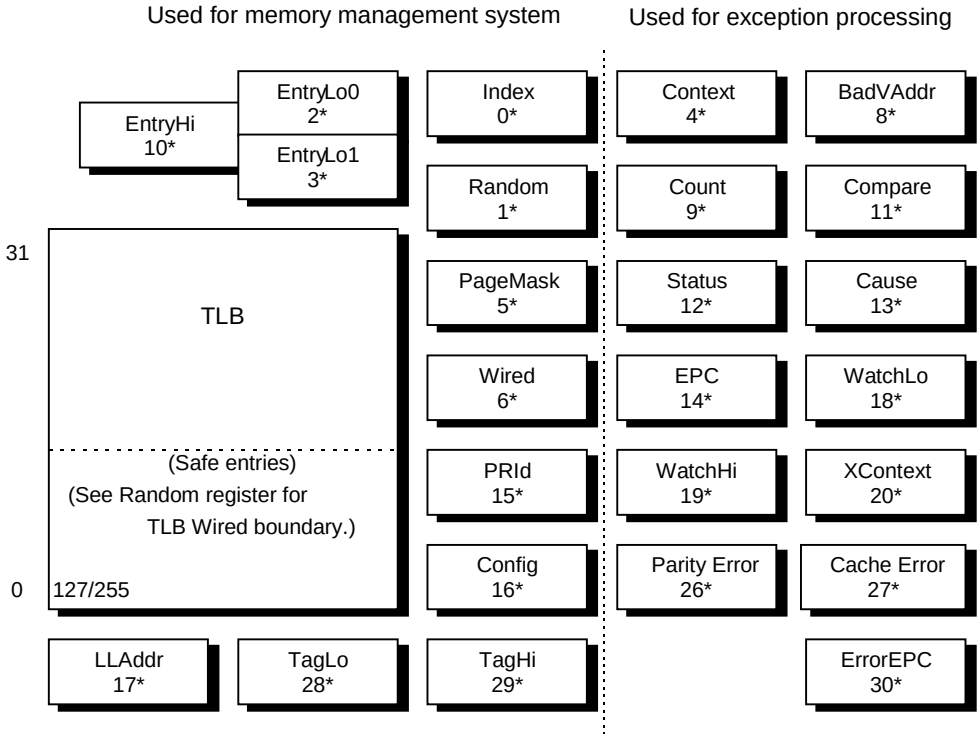
**Note** Can be used exclusively from the extension DRAM

### 2.5.4 System control coprocessor

The System Control Coprocessor (CP0) is implemented as an integral part of the CPU, and supports memory management, address translation, exception processing, and other privileged operations. The CP0 contains the registers and a 32-entry TLB shown in Figure 2-42. The sections that follow describe how the processor uses each of the memory management-related registers.

**Remark** Each CP0 register has a unique number that identifies it; this number is referred to as the register number.

**Figure 2-42. CP0 Registers and TLB**

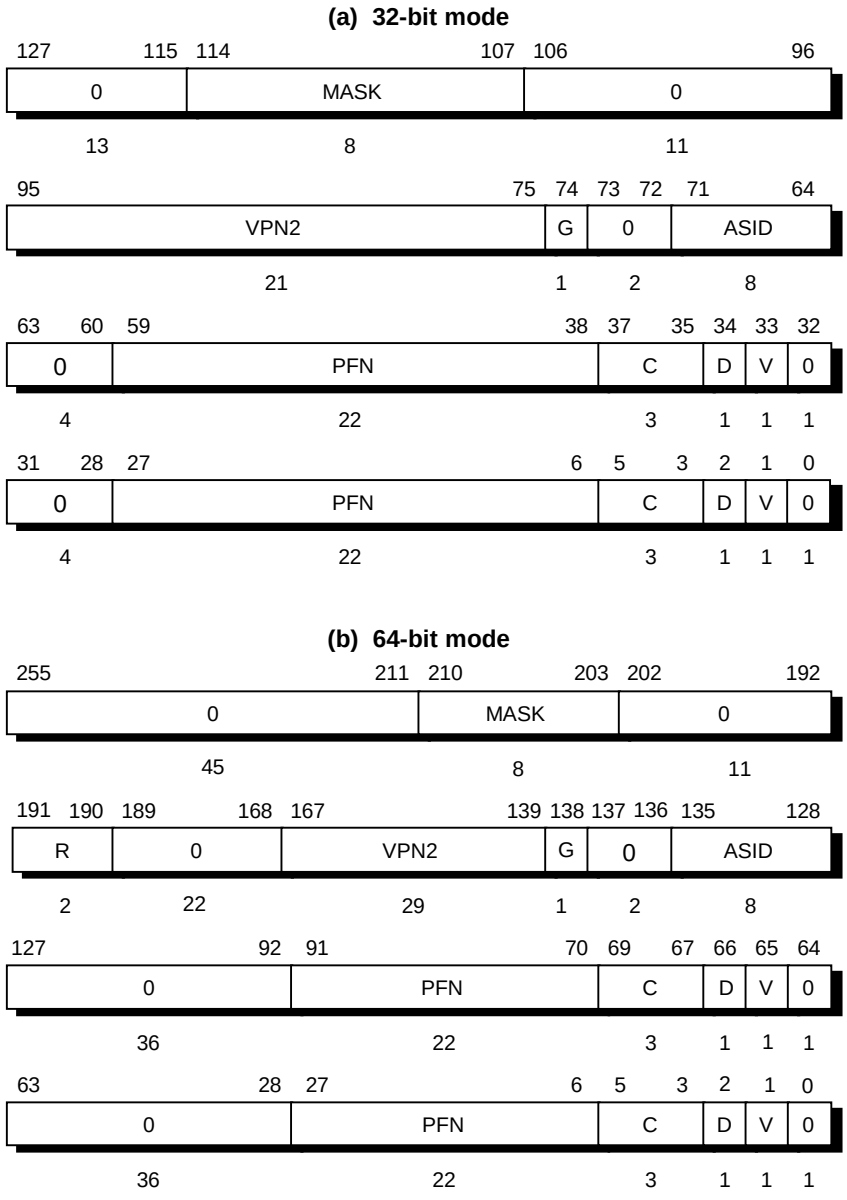



**Remark** *: Register number

### 2.5.4.1 Format of a TLB entry

Figure 2-43 shows the TLB entry formats for both 32- and 64-bit modes. Each field of an entry has a corresponding field in the EntryHi, EntryLo0, EntryLo1, or PageMask registers.

**Figure 2-43. Format of a TLB Entry**




The format of the EntryHi, EntryLo0, EntryLo1, and PageMask registers are nearly the same as the TLB entry. However, it is unknown what bit of the EntryHi register corresponds to the TLB G bit.

### 2.5.5 CP0 registers

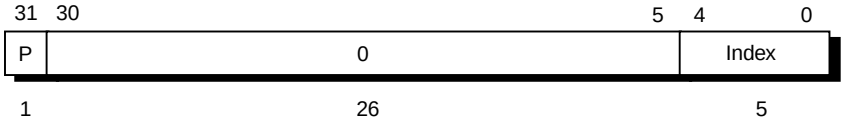
The CP0 registers explained below are accessed by the memory management system and software. The parenthesized number that follows each register name is the register number.

#### 2.5.5.1 Index register (0)

The Index register is a 32-bit, read/write register containing five low-order bits to index an entry in the TLB. The most-significant bit of the register shows the success or failure of a TLB probe (TLBP) instruction.

The Index register also specifies the TLB entry affected by TLB read (TLBR) or TLB write index (TLBWI) instructions.

**Figure 2-44. Index Register**




P : Indicates whether probing is successful or not. It is set to 1 if the latest TLBP instruction fails. It is cleared to 0 when the TLBP instruction is successful.

Index : Specifies an index to a TLB entry that is a target of the TLBR or TLBWI instruction.

0 : RFU. Write 0 in a write operation. When this field is read, 0 is read.

#### 2.5.5.2 Random register (1)

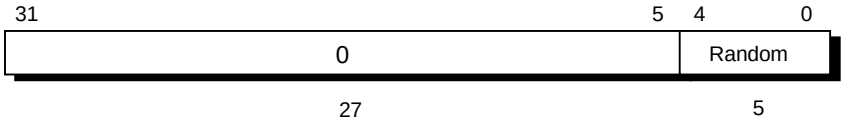
The Random register is a read-only register. The low-order 5 bits are used in referencing a TLB entry. This register is decremented each time an instruction is executed. The values that can be set in the register are as follows:

- ✧ The lower bound is the content of the Wired register.
- ✧ The upper bound is 31.

The Random register specifies the entry in the TLB that is affected by the TLBWR instruction. The register is readable to verify proper operation of the processor.

The Random register is set to the value of the upper bound upon Cold Reset. This register is also set to the upper bound when the Wired register is written. Figure 2-45 shows the format of the Random register.

**Figure 2-45. Random Register**




Random : TLB random index

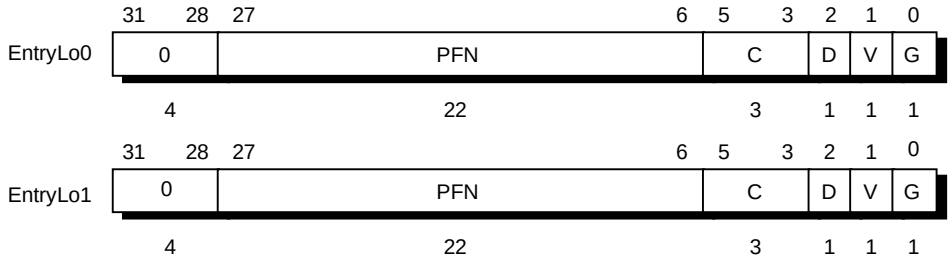
0 : RFU. Write 0 in a write operation. When this field is read, 0 is read.

### 2.5.5.3 EntryLo0 (2) and EntryLo1 (3) registers

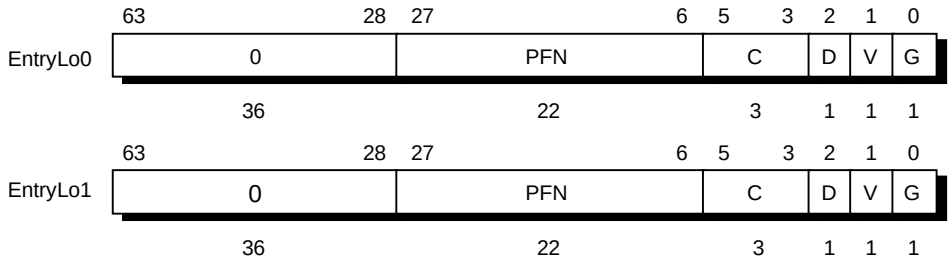
The EntryLo register consists of two registers that have identical formats: EntryLo0, used for even virtual pages and EntryLo1, used for odd virtual pages. The EntryLo0 and EntryLo1 registers are both read-/write-accessible. They are used to access the on-chip TLB. When a TLB read/write operation is carried out, the EntryLo0 and EntryLo1 registers hold the contents of the low-order 32 bits of TLB entries at even and odd addresses, respectively.

**Figure 2-46. EntryLo0 and EntryLo1 Registers**

**(a) 32-bit mode**




**(b) 64-bit mode**




PFN : Page frame number; high-order bits of the physical address.

C : Specifies the TLB page attribute (see Table 2-54).

D : Dirty. If this bit is set to 1, the page is marked as dirty and, therefore, writable. This bit is actually a write-protect bit that software can use to prevent alteration of data.

V : Valid. If this bit is set to 1, it indicates that the TLB entry is valid; otherwise, a TLB Invalid exception (TLBL or TLBS) occurs.

G : Global. If this bit is set in both EntryLo0 and EntryLo1, then the processor ignores the ASID during TLB lookup.

0 : RFU. Write 0 in a write operation. When this field is read, 0 is read.

The coherency attribute (C) bits are used to specify whether to use the cache in referencing a page. When the cache is used, whether the page attribute is “cached” or “uncached” is selected by algorithm.

Table 2-54 lists the page attributes selected according to the value in the C bits.

**Table 2-54. Cache Algorithm**

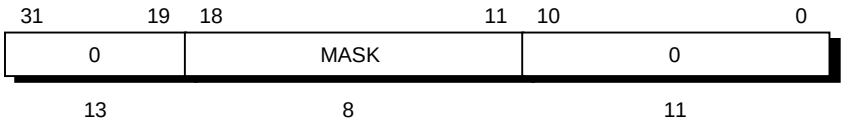
| C Bit Value | Cache Algorithm |
|---|---|
| 0 | Cached |
| 1 | Cached |
| 2 | Uncached |
| 3 | Cached |
| 4 | Cached |
| 5 | Cached |
| 6 | Cached |
| 7 | Cached |

#### 2.5.5.4 PageMask register (5)

The PageMask register is a read/write register used for reading from or writing to the TLB; it holds a comparison mask that sets the page size for each TLB entry, as shown in Table 2-55. Page sizes must be from 1 Kbyte to 256 Kbytes.

TLB read and write instructions use this register as either a source or a destination; Bits 18 to 11 that are targets of comparison are masked during address translation.

**Figure 2-47. Page Mask Register**




MASK : Page comparison mask, which determines the virtual page size for the corresponding entry.

0 : RFU. Write 0 in a write operation. When this field is read, 0 is read.

Table 2-55 lists the mask pattern for each page size. If the mask pattern is one not listed below, the TLB behaves unexpectedly.

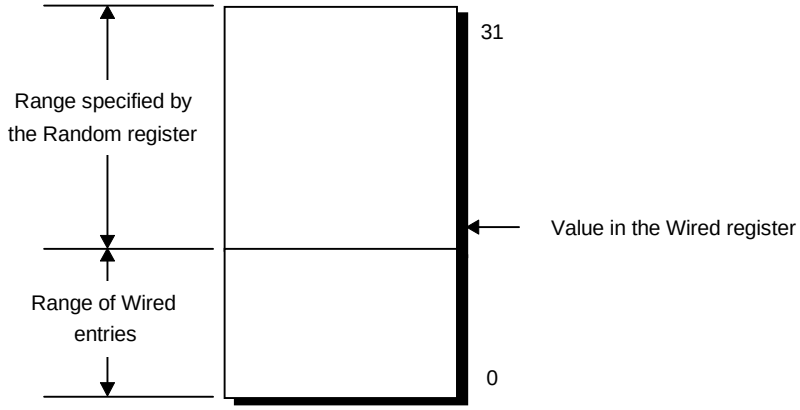
**Table 2-55. Mask Values and Page Sizes**

| Page Size | Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 |
| 1 Kbyte | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 Kbytes | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 16 Kbytes | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 64 Kbytes | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 256 Kbytes | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

### 2.5.5.5 Wired register (6)

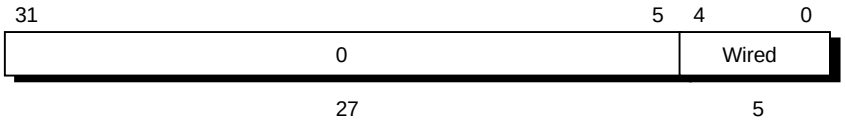
The Wired register is a read/write register that specifies the lower boundary of the random entry of the TLB as shown in Figure 2-48. Wired entries cannot be overwritten by a TLBWR instruction. They can, however, be overwritten by a TLBWI instruction. Random entries can be overwritten by both instructions.

**Figure 2-48. Positions Indicated by Wired Register**




The Wired register is set to 0 upon Cold Reset. Writing this register also sets the Random register to the value of its upper bound (see **Section 2.5.5.2 Random register (1)**). Figure 2-49 shows the format of the Wired register.

**Figure 2-49. Wired Register**




Wired : TLB wired boundary

0 : RFU. Write 0 in a write operation. When this field is read, 0 is read.

### 2.5.5.6 EntryHi register (10)

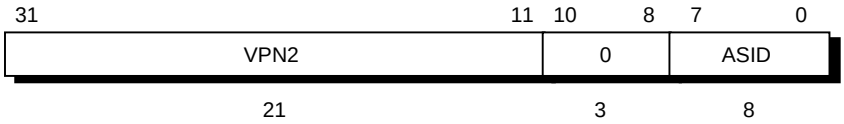
The EntryHi register is write-accessible. It is used to access the on-chip TLB. The EntryHi register holds the high-order bits of a TLB entry for TLB read and write operations. If a TLB Mismatch, TLB Invalid, or TLB Modified exception occurs, the EntryHi register holds the high-order bit of the TLB entry. The EntryHi register is also set with the virtual page number (VPN2) for a virtual address where an exception occurred and the ASID. See **Section 2.6 Exception Processing** for details of the TLB exception.

The ASID is used to read from or write to the ASID field of the TLB entry. It is also checked with the ASID of the TLB entry as the ASID of the virtual address during address translation.

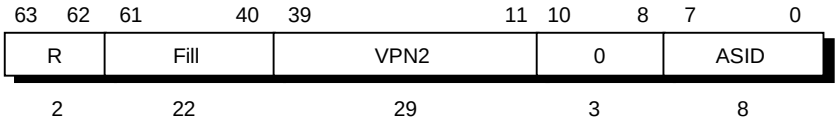
The EntryHi register is accessed by the TLBP, TLBWR, TLBWI, and TLBR instructions.

**Figure 2-50. EntryHi Register**

**(a) 32-bit mode**




**(b) 64-bit mode**




VPN2: Virtual page number divided by two (mapping to two pages)

ASID : Address space ID. An 8-bit ASID field that lets multiple processes share the TLB; each process has a distinct mapping of otherwise identical virtual page numbers.

R : Space type (00 → user, 01 → supervisor, 11 → kernel). Matches bits 63 and 62 of the virtual address.

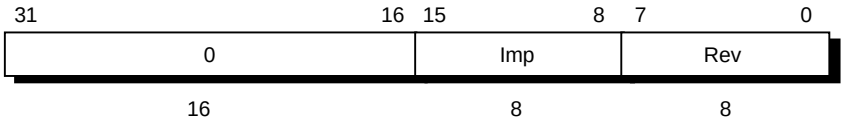
Fill : RFU. Ignored on write. When read, returns zero.

0 : RFU. Write 0 in a write operation. When this field is read, 0 is read.

### 2.5.5.7 Processor revision identifier (PRId) register (15)

The 32-bit, read-only Processor Revision Identifier (PRId) register contains information identifying the implementation and revision level of the CPU and CP0. Figure 2-51 shows the format of the PRId register.

**Figure 2-51. PRId Register**




Imp : CPU core processor ID number (0x0C for the VR4120A)

Rev : CPU core processor revision number

0 : RFU. Write 0 in a write operation. When this field is read, 0 is read.

The processor revision number is stored as a value in the form y.x, where y is a major revision number in bits 7 to 4 and x is a minor revision number in bits 3 to 0.

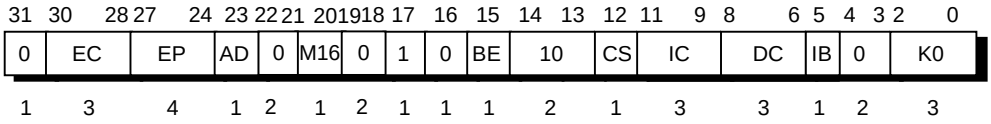
The processor revision number can distinguish some CPU core revisions, however there is no guarantee that changes to the CPU core will necessarily be reflected in the PRId register, or that changes to the revision number necessarily reflect real CPU core changes. Therefore, create a program that does not depend on the processor revision number area.

### 2.5.5.8 Config register (16)

The Config register specifies various configuration options selected on VR4120A processors.

Some configuration options, as defined by the EC and BE fields, are set by the hardware during Cold Reset and are included in the Config register as read-only status bits for the software to access. Other configuration options are read/write (AD, EP, and K0 fields) and controlled by software; on Cold Reset these fields are undefined. Since only a subset of the VR4000 Series options are available in the VR4120A, some bits are set to constants (e.g., bits 14:13) that were variable in the VR4000 Series. The Config register should be initialized by software before caches are used. Figure 2-52 shows the format of the Config register.

**Figure 2-52. Config Register Format**




EC : Frequency ratio of system interface clock (VTCLK) (read only)
0 to 6 → RFU
7 → Pipeline clock (ACLK) frequency /1

EP : Transfer data pattern (cache write-back pattern) setting
0 → DDDD
Others → RFU

AD : Accelerate data mode
0 → VR4000 Series compatible mode
1 → RFU

M16: MIPS16 ISA mode enable/disable indication (read only)
0 → MIPS16 instruction cannot be executed
1 → MIPS16 instruction can be executed.

BE : BigEndianMem. Endian mode of memory and a kernel.
0 → Little endian
1 → Big endian

CS : Cache size mode indication (fixed to 1 in the VR4120A)
0 → IC = $2^{(n+12)}$ Bytes, DC = $2^{(n+12)}$ Bytes
1 → IC = $2^{(n+10)}$ Bytes, DC = $2^{(n+10)}$ Bytes

IC : Instruction cache size indication. In the VR4120A, $2^{(IC+10)}$ bytes.
4 → 16 Kbytes
Others → RFU

DC : Data cache size indication. In the VR4120A, $2^{(DC+10)}$ bytes.
3 → 8 Kbytes
Others → RFU

IB : Select refill size
0 → 4 words (16 bytes)
1 → 8 words (32 bytes)

K0 : kseg0 cache coherency algorithm
010 → Uncached
Others → Cached

1 : 1 is returned when read.

0 : 0 is returned when read.

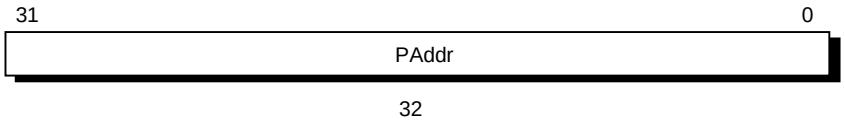
**Caution Be sure to set the EP field, the AD bit and the IB bit to 0. If they are set with any other values, the processor may behave unexpectedly.**

#### 2.5.5.9 Load linked address (LLAddr) register (17)

The read/write Load Linked Address (LLAddr) register is not used with the VR4120A processor except for diagnostic purpose, and serves no function during normal operation.

LLAddr register is implemented just for compatibility between the VR4120A and VR4000/VR4400.

**Figure 2-53. LLAddr Register**




PAddr: 32-bit physical address

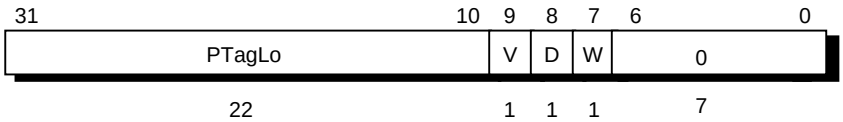
### 2.5.5.10 Cache tag registers (TagLo (28) and TagHi (29))

The TagLo and TagHi registers are 32-bit read/write registers that hold the primary cache tag during cache initialization, cache diagnostics, or cache error processing. The Tag registers are written by the CACHE and MTC0 instructions.

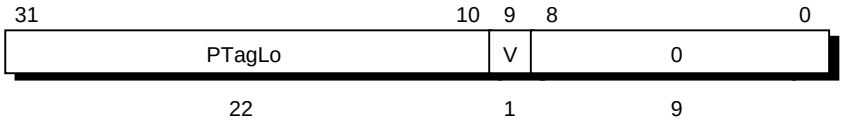
Figures 2-54 and 2-55 show the format of these registers.

**Figure 2-54. TagLo Register**

**(a) When used with data cache**




**(b) When used with instruction cache**




PTagLo: Specifies physical address bits 31 to 10.

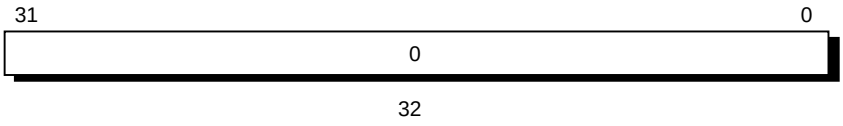
V : Valid bit

D : Dirty bit. However, this bit is defined only for the compatibility with the VR4000 Series processors, and does not indicate the status of cache memory in spite of its readability and writability. This bit cannot change the status of cache memory.

W : Write-back bit (set if cache line has been updated)

0 : RFU. Write 0 in a write operation. When this field is read, 0 is read.

**Figure 2-55. TagHi Register**




0: RFU. Write 0 in a write operation. When this field is read, 0 is read.

#### 2.5.5.11 Virtual-to-physical address translation

During virtual-to-physical address translation, the CPU compares the 8-bit ASID (when the Global bit, G, is not set to 1) of the virtual address to the ASID of the TLB entry to see if there is a match. One of the following comparisons are also made:

- ✧ In 32-bit mode, the high-order bits[Note] of the 32-bit virtual address are compared to the contents of the VPN2 (virtual page number divided by two) of each TLB entry.
- ✧ In 64-bit mode, the high-order bits[Note] of the 64-bit virtual address are compared to the contents of the VPN2 (virtual page number divided by two) of each TLB entry.

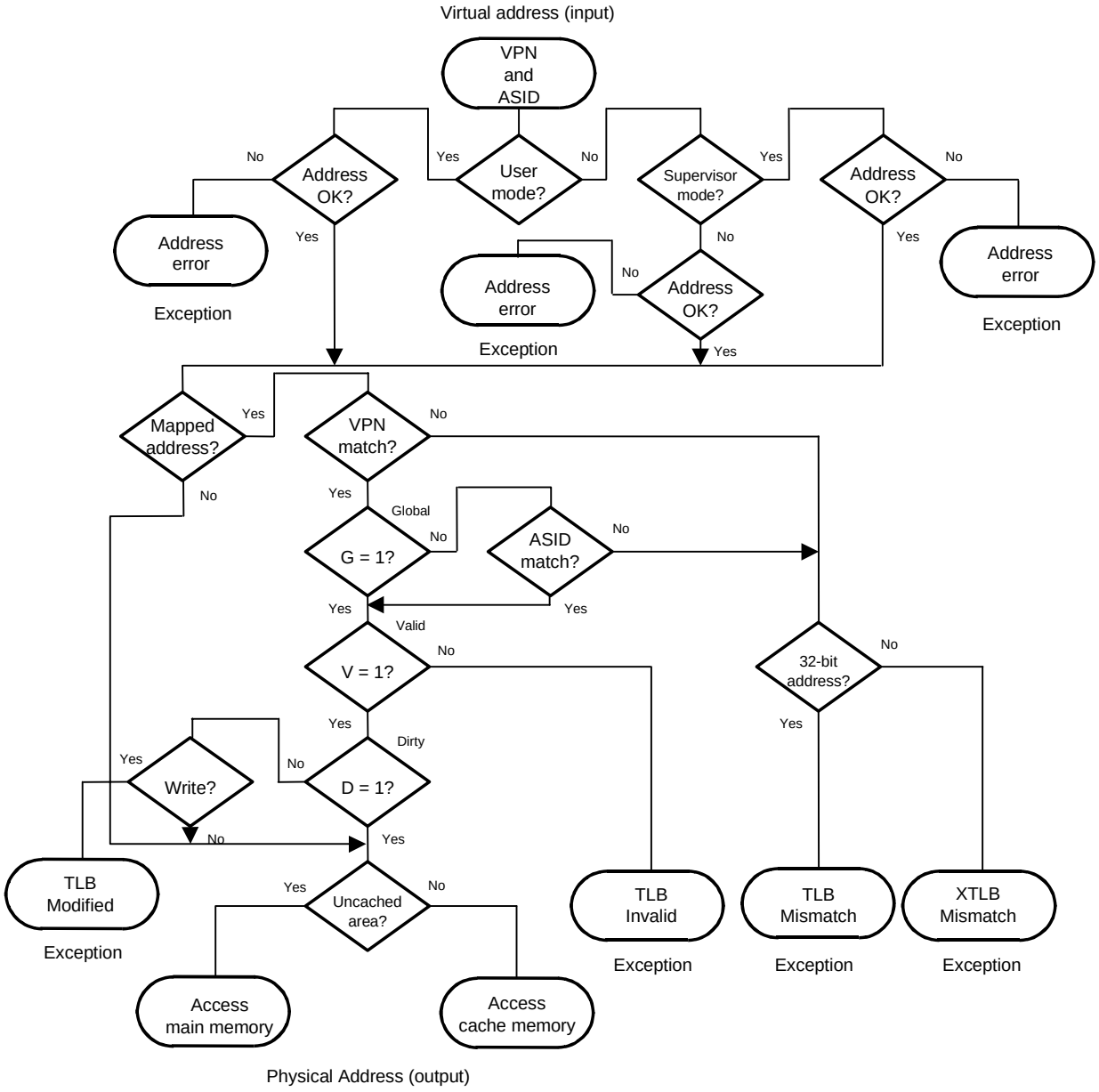
If a TLB entry matches, the physical address and access control bits (C, D, and V) are retrieved from the matching TLB entry. While the V bit of the entry must be set to 1 for a valid address translation to take place, it is not involved in the determination of a matching TLB entry.

Figure 2-56 illustrates the TLB address translation flow.

**Notes 1.** Up to bit 28. Number of bits depends on the TBL page size

**2.** Up to bit 29. Number of bits depends on the TBL page size

**Figure 2-56. TLB Address Translation**




### 2.5.5.12 TLB misses

If there is no TLB entry that matches the virtual address, a TLB Refill (miss) exception occurs[Note]. If the access control bits (D and V) indicate that the access is not valid, a TLB Modified or TLB Invalid exception occurs. If the C bit is 010, the retrieved physical address directly accesses main memory, bypassing the cache.

**Note** See **Section 2.6 Exception Processing** for details of the TLB Miss exception.

#### 2.5.5.13 TLB instructions

The instructions used for TLB control are described below.

**(1) Translation lookaside buffer probe (TLBP)**

The translation lookaside buffer probe (TLBP) instruction loads the Index register with a TLB number that matches the content of the EntryHi register. If there is no TLB number that matches the TLB entry, the highest-order bit of the Index register is set.

**(2) Translation lookaside buffer read (TLBR)**

The translation lookaside buffer read (TLBR) instruction loads the EntryHi, EntryLo0, EntryLo1, and PageMask registers with the content of the TLB entry indicated by the content of the Index register.

**(3) Translation lookaside buffer write index (TLBWI)**

The translation lookaside buffer write index (TLBWI) instruction writes the contents of the EntryHi, EntryLo0, EntryLo1, and PageMask registers to the TLB entry indicated by the content of the Index register.

**(4) Translation lookaside buffer write random (TLBWR)**

The translation lookaside buffer write random (TLBWR) instruction writes the contents of the EntryHi, EntryLo0, EntryLo1, and PageMask registers to the TLB entry indicated by the content of the Random register.

## 2.6 Exception Processing

This chapter describes VR4120A CPU exception processing, including an explanation of hardware that processes exceptions.

### 2.6.1 Exception processing operation

The processor receives exceptions from a number of sources, including translation lookaside buffer (TLB) misses, arithmetic overflows, I/O interrupts, and system calls. When the CPU detects an exception, the normal sequence of instruction execution is suspended and the processor enters Kernel mode (see **Section 2.5 Memory Management System** for a description of system operating modes). If an exception occurs while executing a MIPS16 instruction, the processor stops the MIPS16 instruction execution, and shifts to the 32-bit instruction execution mode.

The processor then disables interrupts and transfers control for execution to the exception handler (located at a specific address as an exception handling routine implemented by software). The exception handler saves the context of the processor, including the contents of the program counter, the current operating mode (User or Supervisor), statuses, and interrupt enabling. This context is saved so it can be restored when the exception has been serviced.

When an exception occurs, the CPU loads the Exception Program Counter (EPC) register with a location where execution can restart after the exception has been serviced. The restart location in the EPC register is the address of the instruction that caused the exception or, if the instruction was executing in a branch delay slot, the address of the branch instruction immediately preceding the delay slot. Note that no branch delay slot generated by executing a branch instruction exists when the processor operates in the MIPS16 mode.

When MIPS16 instructions are enabled to be executed, bit 0 of the EPC register indicates the operating mode in which an exception occurred. It indicates 1 when in the MIPS16 instruction mode, and indicates 0 when in the MIPS III instruction mode.

The VR4120A processor supports a Supervisor mode and high-speed TLB refill for all address spaces. The VR4120A CPU also provides the following functions:

- ✧ Interrupt enable (IE) bit
- ✧ Operating mode (User, Supervisor, or Kernel)
- ✧ Exception level (normal or exception is indicated by the EXL bit in the Status register)
- ✧ Error level (normal or error is indicated by the ERL bit in the Status register).

Interrupts are enabled when the following conditions are satisfied:

#### 2.6.1.1 Interrupt enable

An interrupt is enabled when the following conditions are satisfied.

- Interrupt enable bit (IE) = 1
- EXL bit = 0, ERL bit = 0
- Corresponding IM field bits in the Status register = 1

#### 2.6.1.2 Operating mode

The operating mode is specified by KSU bit in the Status register when both the exception level and error level are normal (0). The operation enters Kernel mode when either EXL bit or ERL bit in the Status register is set to 1.

#### 2.6.1.3 Exception/error levels

Returning from an exception resets the exception level to normal (0) (for details, see **APPENDIX A MIPS III INSTRUCTION SET DETAILS**).

The registers that retain address, cause, and status information during exception processing are described in **Section 2.6.3 Exception processing registers**. For a description of the exception process, see **Section 2.6.4 Details of exceptions**.

### 2.6.2 Precision of exceptions

VR4120A CPU exceptions are logically precise; the instruction that causes an exception and all those that follow it are aborted and can be re-executed after servicing the exception. When succeeding instructions are discarded, exceptions associated with those instructions are also discarded. Exceptions are not taken in the order detected, but in instruction fetch order.

The exception handler can determine the cause of an exception and the address. The program can be restarted by rewriting the destination register - not automatically, however, as in the case of all the other precise exceptions where no status change occurs.

### 2.6.3 Exception processing registers

This section describes the CP0 registers that are used in exception processing. Table 2-56 lists these registers, along with their number-each register has a unique identification number that is referred to as its register number. The CP0 registers not listed in the table are used in memory management (for details, see **Section 2.5 Memory Management System**).

The exception handler examines the CP0 registers during exception processing to determine the cause of the exception and the state of the CPU at the time the exception occurred.

The registers in Table 2-56 are used in exception processing, and are described in the sections that follow.

**Table 2-56. CP0 Exception Processing Registers**

| Register Name | Register Number |
|---|---|
| Context register | 4 |
| BadVAddr register | 8 |
| Count register | 9 |
| Compare register | 11 |
| Status register | 12 |
| Cause register | 13 |
| EPC register | 14 |
| WatchLo register | 18 |
| WatchHi register | 19 |
| XContext register | 20 |
| Parity Error register[Note] | 26 |
| Cache Error register[Note] | 27 |
| Error EPC register | 30 |

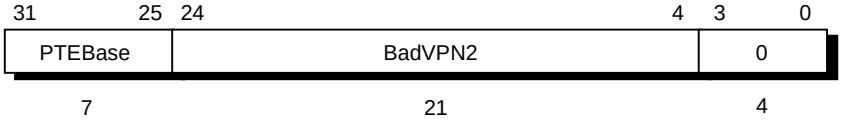
**Note** This register is prepared to maintain compatibility with the VR4100. This register is not used in the μPD98501 hardware.

### 2.6.3.1 Context register (4)

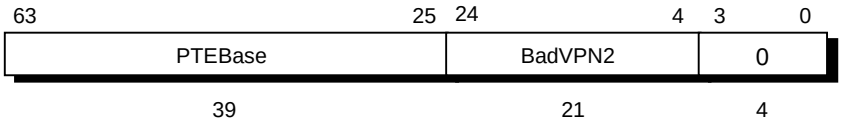
The Context register is a read/write register containing the pointer to an entry in the page table entry (PTE) array on the memory; this array is a table that stores virtual-to-physical address translations. When there is a TLB miss, the operating system loads the unsuccessfully translated entry from the PTE array to the TLB. The Context register is used by the TLB Refill exception handler for loading TLB entries. The Context register duplicates some of the information provided in the BadVAddr register, but the information is arranged in a form that is more useful for a software TLB exception handler. Figure 2-57 shows the format of the Context register.

**Figure 2-57. Context Register Format**

**(a) 32-bit mode**




**(b) 64-bit mode**




PTEBase : The PTEBase field is a base address of the PTE entry table.

BadVPN2 : This field holds the value (VPN2) obtained by halving the virtual page number of the most recent virtual address for which translation failed.

0 : RFU. Write 0 in a write operation. When this field is read, 0 is read.

The PTEBase field is used by software as the pointer to the base address of the PTE table in the current user address space.

The 21-bit BadVPN2 field contains bits 31 to 11 of the virtual address that caused the TLB miss; bit 10 is excluded because a single TLB entry maps to an even-odd page pair. For a 1-Kbyte page size, this format can directly address the pair-table of 8-byte PTEs. When the page size is 4 Kbytes or more, shifting or masking this value produces the correct PTE reference address.

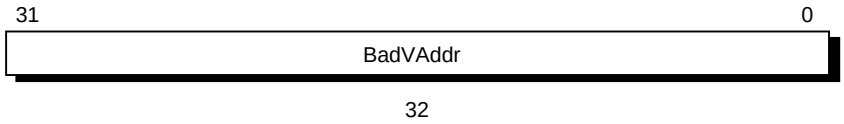
#### 2.6.3.2 BadVAddr register (8)

The Bad Virtual Address (BadVAddr) register is a read-only register that saves the most recent virtual address that failed to have a valid translation, or that had an addressing error. Figure 2-58 shows the format of the BadVAddr register.

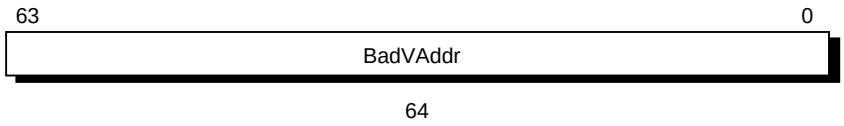
**Caution This register saves no information after a bus error exception, because it is not an address error exception.**

**Figure 2-58. BadVAddr Register Format**

**(a) 32-bit mode**




**(b) 64-bit mode**




BadVAddr: Most recent virtual address for which an addressing error occurred, or for which address translation failed.

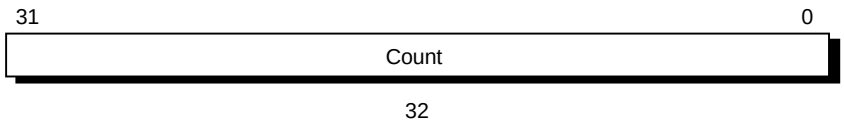
#### 2.6.3.3 Count register (9)

The read/write Count register acts as a timer. It is incremented in synchronization with the frequencies of MasterOut clock, regardless of whether instructions are being executed, retired, or any forward progress is actually made through the pipeline.

This register is a free-running type. When the register reaches all ones, it rolls over to zero and continues counting. This register is used for self-diagnostic test, system initialization, or the establishment of inter-process synchronization.

Figure 2-59 shows the format of the Count register.

**Figure 2-59. Count Register Format**




Count: 32-bit up-date count value that is compared with the value of the Compare register.

#### 2.6.3.4 Compare register (11)

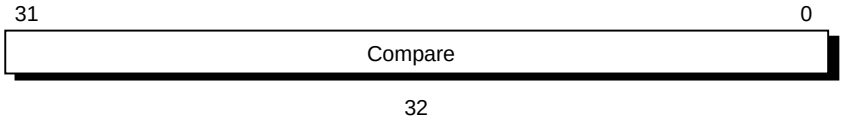
The Compare register causes a timer interrupt; it maintains a stable value that does not change on its own.

When the value of the Count register (see **Section 2.6.3.3 Count register (9)**) equals the value of the Compare register, the IP7 bit in the Cause register is set. This causes an interrupt as soon as the interrupt is enabled.

Writing a value to the Compare register, as a side effect, clears the timer interrupt request.

For diagnostic purposes, the Compare register is a read/write register. Normally, this register should be only used for a write. Figure 2-60 shows the format of the Compare register.

**Figure 2-60. Compare Register Format**

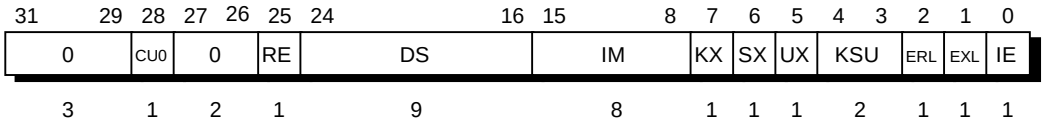



Compare: Value that is compared with the count value of the Count register.

#### 2.6.3.5 Status register (12)

The Status register is a read/write register that contains the operating mode, interrupt enabling, and the diagnostic states of the processor. Figure 2-61 shows the format of the Status register.

**Figure 2-61. Status Register Format**




CU0 : Enables/disables the use of the coprocessor (1 → Enabled, 0 → Disabled).
CP0 can be used by the kernel at all times.

RE : Enables/disables reversing of the endian setting in User mode (0 → Disabled, 1 → Enabled).

**Caution This bit must be set to 0 since the VR4120A CPU supports the little-endian order only.**

DS : Diagnostic Status field (see Figure 2-62).

IM : Interrupt Mask field used to enable/disable external/internal and software interrupts (0 → Disabled, 1 → Enabled). This field consists of 8 bits that are used to control eight interrupts. The bits are assigned to interrupts as follows:

IM7 : Masks a timer interrupt.

IM(6:2) :Mask ordinary interrupts (Int(4:0)[Note]). However, Int4[Note] never occur in the VR4120A CPU.

IM(1:0) : Software interrupts.

**Note** Int(4:0) are internal signals of the CPU core. For details about connection to the on-chip peripheral units.

KX : Enables 64-bit addressing in Kernel mode (0 → 32-bit, 1 → 64-bit). If this bit is set, an XTLB Refill exception occurs if a TLB miss occurs in the Kernel mode address space.
In addition, 64-bit operations are always valid in kernel mode.

SX : Enables 64-bit addressing and operation in Supervisor mode (0 → 32-bit, 1 → 64-bit). If this bit is set, an XTLB Refill exception occurs if a TLB miss occurs in the Supervisor mode address space.

UX: : Enables 64-bit addressing and operation in User mode (0 → 32-bit, 1 → 64-bit). If this bit is set, an XTLB Refill exception occurs if a TLB miss occurs in the User mode address space.

KSU : Sets and indicates the operating mode (10 → User, 01 → Supervisor, 00 → Kernel).

ERL : Sets and indicates the error level (0 → Normal, 1 → Error).

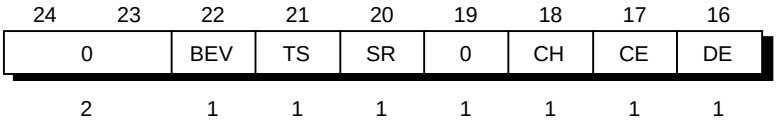
EXL : Sets and indicates the exception level (0 → Normal, 1 → Exception).

IE : Sets and indicates interrupt enabling/disabling (0 → Disabled, 1 → Enabled).

0 : RFU. Write 0 in a write operation. When this bit is read, 0 is read.

Figure 2-62 shows the details of the Diagnostic Status (DS) field. All DS field bits other than the TS bit are writable.

**Figure 2-62. Status Register Diagnostic Status Field**




BEV : Specifies the base address of a TLB Refill exception vector and common exception vector (0 → Normal, 1 → Bootstrap).

TS : Occurs the TLB to be shut down (read-only) (0 → Not shut down, 1 → Shut down). This bit is used to avoid any problems that may occur when multiple TLB entries match the same virtual address. After the TLB has been shut down, reset the processor to enable restart. Note that the TLB is shut down even if a TLB entry matching a virtual address is marked as being invalid (with the V bit cleared).

SR : Occurs a Soft Reset or NMI exception (0 → Not occurred, 1 → Occurred).

CH : CP0 condition bit (0 → False, 1 → True). This bit can be read and written by software only; it cannot be accessed by hardware.

CE, DE: These are prepared to maintain compatibility with the VR4100, and are not used in the VR4120A Core hardware.

0 : RFU. Write 0 in a write operation. When this field is read, 0 is read.

The status register has the following fields where the modes and access status are set.

**(1) Interrupt enable**

Interrupts are enabled when all of the following conditions are true:

- ✧ IE is set to 1.
- ✧ EXL is cleared to 0.
- ✧ ERL is cleared to 0.
- ✧ The appropriate bit of the IM is set to 1.

**(2) Operating modes**

The following Status register bit settings are required for User, Kernel, and Supervisor modes.

- ✧ The processor is in User mode when KSU = 10, EXL = 0, and ERL = 0.
- ✧ The processor is in Supervisor mode when KSU = 01, EXL = 0, and ERL = 0.
- ✧ The processor is in Kernel mode when KSU = 00, EXL = 1, or ERL = 1.

**(3) 32- and 64-bit modes**

The following Status register bit settings select 32- or 64-bit operation for User, Kernel, and Supervisor operating modes. Enabling 64-bit operation permits the execution of 64-bit opcodes and translation of 64-bit addresses. 64-bit operation for User, Kernel and Supervisor modes can be set independently.

- ✧ 64-bit addressing for Kernel mode is enabled when KX bit = 1. 64-bit operations are always valid in Kernel mode.
- ✧ 64-bit addressing and operations are enabled for Supervisor mode when SX bit = 1.
- ✧ 64-bit addressing and operations are enabled for User mode when UX bit = 1.

**(4) Kernel address space accesses**

Access to the kernel address space is allowed when the processor is in Kernel mode.

**(5) Supervisor address space accesses**

Access to the supervisor address space is allowed when the processor is in Supervisor or Kernel mode.

**(6) User address space accesses**

Access to the user address space is allowed in any of the three operating modes.

**(7) Status after reset**

The contents of the Status register are undefined after Cold resets, except for the following bits in the diagnostic status field.

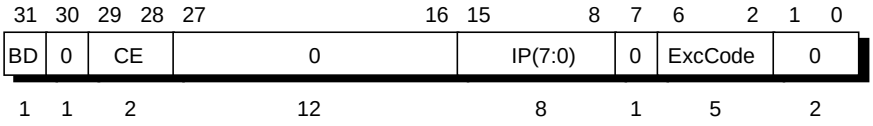
- TS and SR are cleared to 0.
- ERL and BEV are set to 1.
- SR is 0 after Cold reset, and is 1 after Soft reset or NMI interrupt.

**Remark** Cold reset and Soft reset are CPU core reset (see **Section 2.7 Reset of the CPU Core**).

#### 2.6.3.6 Cause register (13)

The 32-bit read/write Cause register holds the cause of the most recent exception. A 5-bit exception code indicates one of the causes (see Table 2-57). Other bits hold the detailed information of the specific exception. All bits in the Cause register, with the exception of the IP1 and IP0 bits, are read-only; IP1 and IP0 are used for software interrupts. Figure 2-63 shows the fields of this register; Table 2-57 describes the Cause register codes.

**Figure 2-63. Cause Register Format**




BD : Indicates whether the most recent exception occurred in the branch delay slot (1 → In delay slot, 0 → Normal).

CE : Indicates the coprocessor number in which a Coprocessor Unusable exception occurred. This field will remain undefined for as long as no exception occurs.

IP : Indicates whether an interrupt is pending (1 → Interrupt pending, 0 → No interrupt pending).

- IP7 : A timer interrupt.
- IP(6:2) : Ordinary interrupts (Int(4:0)[Note]). However, Int4[Note] never occurs in the VR4120A CPU.
- IP(1:0) : Software interrupts. Only these bits cause an interrupt exception, when they are set to 1 by means of software.

**Note** Int(4:0) are internal signals of the CPU core. For details about connection to the on-chip peripheral units.

ExcCode: Exception code field (refer to Table 2-57 for details).

0 : RFU. Write 0 in a write operation. When this field is read, 0 is read.

**Table 2-57. Cause Register Exception Code Field**

| Exception Code | Mnemonic | Description |
|---|---|---|
| 0 | Int | Interrupt exception |
| 1 | Mod | TLB Modified exception |
| 2 | TLBL | TLB Refill exception (load or fetch) |
| 3 | TLBS | TLB Refill exception (store) |
| 4 | AdEL | Address Error exception (load or fetch) |
| 5 | AdES | Address Error exception (store) |
| 6 | IBE | Bus Error exception (instruction fetch) |
| 7 | DBE | Bus Error exception (data load or store) |
| 8 | Sys | System Call exception |
| 9 | Bp | Breakpoint exception |
| 10 | RI | Reserved Instruction exception |
| 11 | CpU | Coprocessor Unusable exception |
| 12 | Ov | Integer Overflow exception |
| 13 | Tr | Trap exception |
| 14 to 22 | — | RFU |
| 23 | WATCH | Watch exception |
| 24 to 31 | — | RFU |

The VR4120A CPU has eight interrupt request sources, IP7 to IP0. For the detailed description of interrupts, refer to **Section 2.9 CPU Core Interrupts**.

**(1) IP7**

This bit indicates whether there is a timer interrupt request.

It is set when the values of Count register and Compare register match.

**(2) IP6 to IP2**

IP6 to IP2 reflect the state of the interrupt request signal of the CPU core.

**(3) IP1 and IP0**

These bits are used to set/clear a software interrupt request.

#### 2.6.3.7 Exception program counter (EPC) register (14)

The Exception Program Counter (EPC) is a read/write register that contains the address at which processing resumes after an exception has been serviced. The contents of this register change depending on whether execution of MIPS16 instructions is enabled or disabled. Setting the MIPS16EN pin after RTC reset specifies whether execution of the MIPS16 instructions is enabled or disabled.

When the MIPS16 instruction execution is disabled, the EPC register contains either:

- Virtual address of the instruction that caused the exception.
- Virtual address of the immediately preceding branch or jump instruction (when the instruction associated with the exception is in a branch delay slot, and the BD bit in the Cause register is set to 1).

When the MIPS16 instruction execution is enabled, the EPC register contains either:

- Virtual address of the instruction that caused the exception and ISA mode at which an exception occurs.
- Virtual address of the immediately preceding branch or jump instruction and ISA mode at which an exception occurs (when the instruction associated with the exception is in a branch delay slot of the jump instruction, and the BD bit in the Cause register is set to 1).

When the 16-bit instruction is executed, the EPC register contains either:

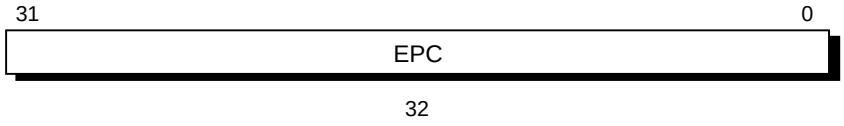
- Virtual address of the instruction that caused the exception and ISA mode at which an exception occurs.
- Virtual address of the immediately preceding Extend or jump instruction and ISA mode at which an exception occurs (when the instruction associated with the exception is in a branch delay slot of the jump instruction or in the instruction following the Extend instruction, and the BD bit in the Cause register is set to 1).

The EXL bit in the Status register is set to 1 to keep the processor from overwriting the address of the exception-causing instruction contained in the EPC register in the event of another exception.

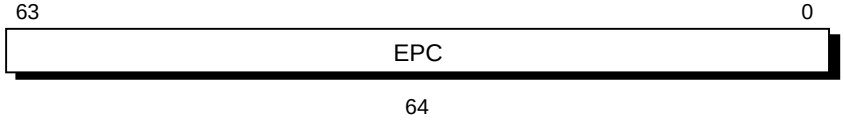
Figure 2-64 shows the EPC register format when MIPS16 ISA is disabled, and Figure 2-65 shows the EPC register format when MIPS16 ISA is enabled.

**Figure 2-64. EPC Register Format (When MIPS16 ISA Is Disabled)**

**(a) 32-bit mode**



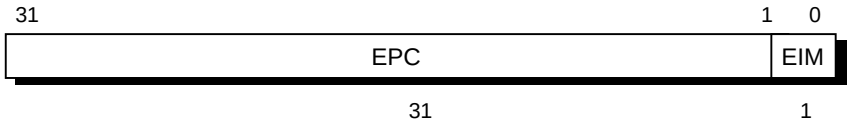

**(b) 64-bit mode**




EPC: Restart address after exception processing.

**Figure 2-65. EPC Register Format (When MIPS16 ISA Is Enabled)**

**(a) 32-bit mode**

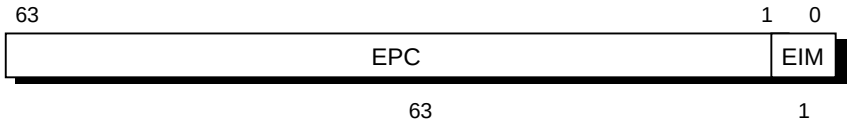



EPC: Virtual address that caused the exception (31:1).

EIM: ISA mode at which an exception occurs.

(1: when MIPS16 SIA instruction is executed, 0: when MIPS III ISA instruction is executed.)

**(b) 64-bit mode**




EPC: Virtual address that caused the exception (63:1).

EIM: ISA mode at which an exception occurs.

(1: when MIPS16 ISA instruction is executed, 0: when MIPS III ISA instruction is executed.)

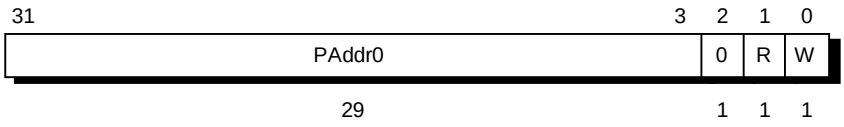
### 2.6.3.8 WatchLo (18) and WatchHi (19) registers

The VR4120A processor provides a debugging feature to detect references to a selected physical address; load and store instructions to the location specified by the WatchLo and WatchHi registers cause a Watch exception.

Figures 2-66 and 2-67 show the format of the WatchLo and WatchHi registers.

**Figure 2-66. WatchLo Register Format**

**WatchLo Register**




PAddr0 : Specifies physical address bits 31 to 3.

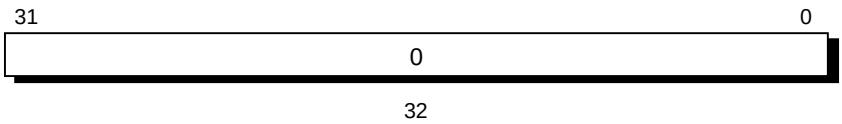
R : If this bit is set to 1, an exception will occur when a load instruction is executed.

W : If this bit is set to 1, an exception will occur when a store instruction is executed.

0 : RFU. Write 0 in a write operation. When this field is read, 0 is read.

**Figure 2-67. WatchHi Register Format**

**WatchHi Register**




0 : RFU. Write 0 in a write operation. When this field is read, 0 is read.

#### 2.6.3.9 XContext register (20)

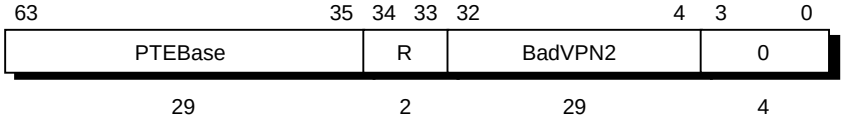
The read/write XContext register contains a pointer to an entry in the page table entry (PTE) array, an operating system data structure that stores virtual-to-physical address translations. If a TLB miss occurs, the operating system loads the untranslated data from the PTE into the TLB to handle the software error.

The XContext register is used by the XTLB Refill exception handler to load TLB entries in 64-bit addressing mode.

The XContext register duplicates some of the information provided in the BadVAddr register, and puts it in a form useful for the XTLB exception handler.

This register is included solely for operating system use. The operating system sets the PTEBase field in the register, as needed. Figure 2-68 shows the format of the XContext register.

**Figure 2-68. XContext Register Format**




PTEBase : The PTEBase field is a base address of the PTE entry table.

R : Space type (00 → User, 01→ Supervisor, 11 → Kernel). The setting of this field matches virtual address bits 63 and 62.

BadVPN2 : This field holds the value (VPN2) obtained by halving the virtual page number of the most recent virtual address for which translation failed.

0 : RFU. Write 0 in a write operation. When this field is read, 0 is read.

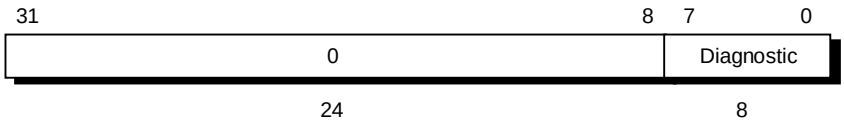
The 29-bit BadVPN2 field has bits 39 to 11 of the virtual address that caused the TLB miss; bit 10 is excluded because a single TLB entry maps to an even-odd page pair. For a 1-Kbyte page size, this format may be used directly to address the pair-table of 8-byte PTEs. For 4-Kbyte or more page and PTE sizes, shifting or masking this value produces the appropriate address.

#### 2.6.3.10 Parity error register (26)

The Parity Error (PErr) register is a readable/writable register. This register is defined to maintain software-compatibility with the VR4100, and is not used in hardware because the VR4120A CPU has no parity.

Figure 2-69 shows the format of the PErr register.

**Figure 2-69. Parity Error Register Format**




Diagnostic : 8-bit self diagnostic field.

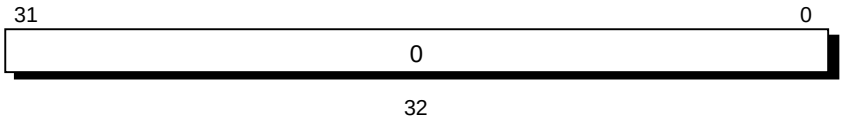
0 : RFU. Write 0 in a write operation. When this field is read, 0 is read.

#### 2.6.3.11 Cache error register (27)

The Cache Error register is a readable/writable register. This register is defined to maintain software-compatibility with the VR4100, and is not used in hardware because the VR4120A CPU has no parity.

Figure 2-70 shows the format of the Cache Error register.

**Figure 2-70. Cache Error Register Format**




0 : RFU. Write 0 in a write operation. When this field is read, 0 is read.

### 2.6.3.12 ErrorEPC register (30)

The Error Exception Program Counter (ErrorEPC) register is similar to the EPC register. It is used to store the Program Counter value at which the Cache Error, Cold Reset, Soft Reset, or NMI exception has been serviced.

The read/write ErrorEPC register contains the virtual address at which instruction processing can resume after servicing an error. The contents of this register change depending on whether execution of MIPS16 instructions is enabled or disabled. Setting the MIPS16EN pin after RTC reset specifies whether the execution of MIPS16 instructions is enabled or disabled.

When the MIPS16 instruction execution is disabled, this address can be:

- Virtual address of the instruction that caused the exception.
- Virtual address of the immediately preceding branch or jump instruction, when the instruction associated with the error exception is in a branch delay slot.

When the MIPS16 instruction execution is enabled during a 32-bit instruction execution, this address can be:

- Virtual address of the instruction that caused the exception and ISA mode at which an exception occurs.
- Virtual address of the immediately preceding branch or jump instruction and ISA mode at which an exception occurs when the instruction associated with the exception is in a branch delay slot.

When the MIPS16 instruction execution is enabled during a 16-bit instruction execution, this address can be:

- Virtual address of the instruction that caused the exception and ISA mode at which an exception occurs.
- Virtual address of the immediately preceding jump instruction or Extend instruction and ISA mode at which an exception occurs when the instruction associated with the exception is in a branch delay slot of the jump instruction or is the instruction following the Extend instruction.

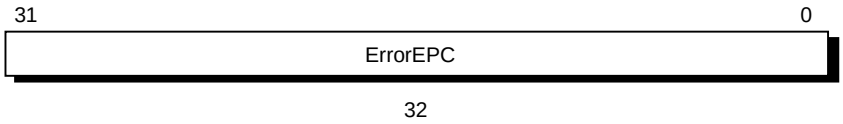
The contents of the ErrorEPC register do not change when the ERL bit of the Status register is set to 1. This prevents the processor when other exceptions occur from overwriting the address of the instruction in this register which causes an error exception.

There is no branch delay slot indication for the ErrorEPC register.

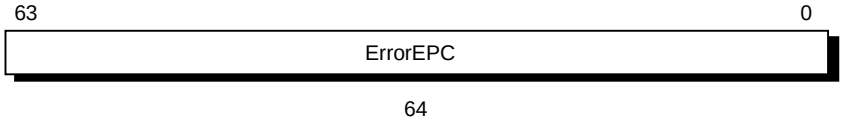
Figure 2-71 shows the format of the ErrorEPC register when the MIPS16 ISA is disabled. Figure 2-72 shows the format of the ErrorEPC register when the MIPS16 ISA is enabled.

**Figure 2-71. ErrorEPC Register Format (When MIPS16 ISA Is Disabled)**

**(a) 32-bit mode**



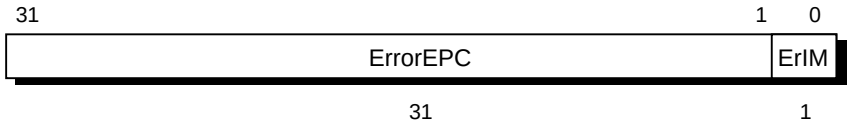

**(b) 64-bit mode**




ErrorEPC: Program counter that indicates the restart address after Cold reset, Soft reset, or NMI exception.

**Figure 2-72. ErrorEPC Register Format (When MIPS16 ISA Is Enabled)**

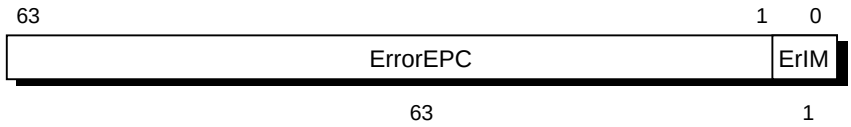
**(a) 32-bit mode**




ErrorEPC: Virtual restart address (31:1) after Cold reset, Soft reset, or NMI exception.

ErIM: ISA mode at which an error exception occurs (1: MIPS16 ISA, 0: MIPS III ISA).

**(b) 64-bit mode**




ErrorEPC: Virtual restart address (63:1) after Cold reset, Soft reset, or NMI exception.

ErIM: ISA mode at which an error exception occurs (1: MIPS16 ISA, 0: MIPS III ISA).

### 2.6.4 Details of exceptions

This section describes causes, processes, and services of the VR4120A's exceptions.

#### 2.6.4.1 Exception types

This section gives sample exception handler operations for the following exception types:

- ✧ Cold Reset
- ✧ Soft Reset
- ✧ NMI
- ✧ Remaining processor exceptions

When the EXL and ERL bits in the Status register are 0, either User, Supervisor, or Kernel operating mode is specified by the KSU bits in the Status register. When either the EXL or ERL bit is set to 1, the processor is in Kernel mode.

When the processor takes an exception, the EXL bit is set to 1, meaning the system is in Kernel mode. After saving the appropriate state, the exception handler typically resets the EXL bit back to 0. The exception handler sets the EXL bit to 1 so that the saved state is not lost upon the occurrence of another exception while the saved state is being restored.

Returning from an exception also resets the EXL bit to 0. For details, see **APPENDIX A MIPS III INSTRUCTION SET DETAILS**.

#### 2.6.4.2 Exception vector address

The Cold Reset, Soft Reset, and NMI exceptions are always branched to the following reset exception vector address. This address is in an uncached, unmapped space.

- ✧ 0xBFC0 0000 in 32-bit mode (virtual address)
- ✧ 0xFFFF FFFF BFC0 0000 in 64-bit mode (virtual address)

Vector addresses for the remaining exceptions are a combination of a vector offset and a base address.

64-/32-bit mode exception vectors and their offsets are shown below.

**Table 2-58. 64-Bit Mode Exception Vector Base Addresses**

| | Vector Base Address (Virtual) | Vector Offset |
|---|---|---|
| Cold Reset<br>Soft Reset<br>NMI | 0xFFFF FFFF BFC0 0000<br>(BEV bit is automatically set to 1) | 0x0000 |
| TLB Refill (EXL = 0) | 0xFFFF FFFF 8000 0000 (BEV = 0)<br>0xFFFF FFFF BFC0 0200 (BEV = 1) | 0x0000 |
| XTLB Refill (EXL = 0) | | 0x0080 |
| Other exceptions | | 0x0180 |

**Table 2-59. 32-Bit Mode Exception Vector Base Addresses**

| | Vector Base Address (Virtual) | Vector Offset |
|---|---|---|
| Cold Reset<br>Soft Reset<br>NMI | 0xBFC0 0000<br>(BEV bit is automatically set to 1) | 0x0000 |
| TLB Refill (EXL = 0) | 0x8000 0000 (BEV = 0)<br>0xBFC0 0200 (BEV = 1) | 0x0000 |
| XTLB Refill (EXL = 0) | | 0x0080 |
| Other exceptions | | 0x0180 |

**(1) TLB refill exception vector**

When BEV bit = 0, the vector base address (virtual) for the TLB Refill exception is in kseg0 (unmapped) space.

- ✧ 0x8000 0000 in 32-bit mode
- ✧ 0xFFFF FFFF 8000 0000 in 64-bit mode

When BEV bit = 1, the vector base address (virtual) for the TLB Refill exception is in kseg1 (uncached, unmapped) space.

- ✧ 0xBFC0 0200 in 32-bit mode
- ✧ 0xFFFF FFFF BFC0 0200 in 64-bit mode

This is an uncached, non-TLB-mapped space, allowing the exception handler to bypass the cache and TLB.

### 2.6.4.3 Priority of exceptions

While more than one exception can occur for a single instruction, only the exception with the highest priority is reported. Table 2-60 lists the priorities.

**Table 2-60. Exception Priority Order**

| | |
|---|---|
| High | Cold Reset |
| ↑ | Soft Reset |
| \| | NMI |
| \| | Address Error (instruction fetch) |
| \| | TLB/XTLB Refill (instruction fetch) |
| \| | TLB Invalid (instruction fetch) |
| \| | Bus Error (instruction fetch) |
| \| | System Call |
| \| | Breakpoint |
| \| | Coprocessor Unusable |
| \| | Reserved Instruction |
| \| | Trap |
| \| | Integer Overflow |
| \| | Address Error (data access) |
| \| | TLB/XTLB Refill (data access) |
| \| | TLB Invalid (data access) |
| \| | TLB Modified (data write) |
| \| | Watch |
| ↓ | Bus Error (data access) |
| Low | Interrupt (other than NMI) |

Hereafter, handling exceptions by hardware is referred to as "process", and handling exception by software is referred to as "service".

#### 2.6.4.4 Cold reset exception

**(1) Cause**

The Cold Reset exception occurs when the ColdReset_B signal (internal) is asserted and then deasserted. This exception is not maskable. The Reset_B signal (internal) must be asserted along with the ColdReset_B signal (for details, see **Section 2.7 Initialization Interface**).

**(2) Processing**

The CPU provides a special interrupt vector for this exception:

- ✧ 0xBFC0 0000 (virtual address) in 32-bit mode
- ✧ 0xFFFF FFFF BFC0 0000 (virtual address) in 64-bit mode

The Cold Reset vector resides in unmapped and uncached CPU address space, so the hardware need not initialize the TLB or the cache to process this exception. It also means the processor can fetch and execute instructions while the caches and virtual memory are in an undefined state.

The contents of all registers in the CPU are undefined when this exception occurs, except for the following register fields:

- When the MIPS16 instruction execution is disabled while the ERL of Status register is 0, the PC value at which an exception occurs is set to the ErrorEPC register.
  When the MIPS16 instruction execution is enabled while the ERL of Status register is 0, the PC value at which an exception occurs is set to the ErrorEPC register and the ISA mode in which an exception occurs is set to the least significant bit of the ErrorEPC register.
- TS and SR bits of the Status register are cleared to 0.
- ERL and BEV bits of the Status register are set to 1.
- The Random register is initialized to the value of its upper bound (31).
- The Wired register is initialized to 0.
- Bits 31 to 28 and bits 22 to 3 of the Config register are set to fixed values.
- All other bits are undefined.

**(3) Servicing**

The Cold Reset exception is serviced by:

- Initializing all processor registers, coprocessor registers, TLB, caches, and the memory system
- Performing diagnostic tests
- Bootstrapping the operating system

#### 2.6.4.5 Soft reset exception

**(1) Cause**

A Soft Reset (sometimes called Warm Reset) occurs when the ColdReset_B signal (internal) remains deasserted while the Reset_B signal (internal) goes from assertion to deassertion (for details, see **Section 2.7 Initialization Interface**).

A Soft Reset immediately resets all state machines, and sets the SR bit of the Status register. Execution begins at the reset vector when the reset is deasserted.

This exception is not maskable.

**Caution In the μPD98501, a soft reset never occurs.**

**(2) Processing**

The CPU provides a special interrupt vector for this exception (same location as Cold Reset):

- ✧ 0xBFC0 0000 (virtual) in 32-bit mode
- ✧ 0xFFFF FFFF BFC0 0000 (virtual) in 64-bit mode

This vector is located within unmapped and uncached address space, so that the cache and TLB need not be initialized to process this exception. The SR bit of the Status register is set to 1 to distinguish this exception from a Cold Reset exception.

When this exception occurs, the contents of all registers are preserved except for the following registers:

- When the MIPS16 instruction execution is disabled, the PC value at which an exception occurs is set to the ErrorEPC register.
  When the MIPS16 instruction execution is enabled, the PC value at which an exception occurs is set to the ErrorEPC register and the ISA mode in which an exception occurs is set to the least significant bit of the ErrorEPC register.
- TS bit of the Status register is cleared to 0.
- ERL, SR, and BEV bits of the Status register are set to 1.

During a soft reset, access to the operating cache or system interface may be aborted. This means that the contents of the cache and memory will be undefined if a Soft Reset occurs.

**(3) Servicing**

The Soft Reset exception is serviced by:

- Preserving the current processor states for diagnostic tests
- Reinitializing the system in the same way as for a Cold Reset exception

#### 2.6.4.6 NMI exception

**(1) Cause**

The Nonmaskable Interrupt (NMI) exception occurs when the NMI signal (internal) becomes active. This interrupt is not maskable; it occurs regardless of the settings of the EXL, ERL, and IE bits in the Status register (for details, see **Section 2.9 CPU Core Interrupts**).

**(2) Processing**

The CPU provides a special interrupt vector for this exception:

- ✧ 0xBFC0 0000 (virtual) in 32-bit mode
- ✧ 0xFFFF FFFF BFC0 0000 (virtual) in 64-bit mode

This vector is located within unmapped and uncached address space so that the cache and TLB need not be initialized to process an NMI exception. The SR bit of the Status register is set to 1 to distinguish this exception from a Cold Reset exception.

Unlike Cold Reset and Soft Reset, but like other exceptions, NMI is taken only at instruction boundaries. The states of the caches and memory system are preserved by this exception.

When this exception occurs, the contents of all registers are preserved except for the following registers:

- When the MIPS16 instruction execution is disabled, the PC value at which an exception occurs is set to the ErrorEPC register.
  When the MIPS16 instruction execution is enabled, the PC value at which an exception occurs is set to the ErrorEPC register and the ISA mode in which an exception occurs is set to the least significant bit of the ErrorEPC register.
- The TS bit of the Status register is cleared to 0.
- The ERL, SR, and BEV bits of the Status register are set to 1.

**(3) Servicing**

The NMI exception is serviced by:

- Preserving the current processor states for diagnostic tests
- Reinitializing the system in the same way as for a Cold Reset exception

#### 2.6.4.7 Address error exception

**(1) Cause**

The Address Error exception occurs when an attempt is made to execute one of the following. This exception is not maskable.

- Execution of the LW, LWU, SW, or CACHE instruction for word data that is not located on a word boundary
- Execution of the LH, LHU, or SH instruction for half-word data that is not located on a half-word boundary
- Execution of the LD or SD instruction for double-word data that is not located on a double-word boundary
- Referencing the kernel address space in User or Supervisor mode
- Referencing the supervisor space in User mode
- Referencing an address that does not exist in the kernel, user, or supervisor address space in 64-bit Kernel, User, or Supervisor mode
- Branching to an address that was not located on a word boundary when the MIPS16 instruction is disabled
- Branching to address whose least-significant 2 bits are 10 when the MIPS16 instruction is enabled

**(2) Processing**

The common exception vector is used for this exception. The AdEL or AdES code in the Cause register is set. If this exception has been caused by an instruction reference or load operation, AdEL is set. If it has been caused by a store operation, AdES is set.

When this exception occurs, the BadVAddr register stores the virtual address that was not properly aligned or was referenced in protected address space. The contents of the VPN field of the Context and EntryHi registers are undefined, as are the contents of the EntryLo register.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

**(3) Servicing**

The kernel reports the UNIX® SIGSEGV (segmentation violation) signal to the current process, and this exception is usually fatal.

#### 2.6.4.8 TLB exceptions

Three types of TLB exceptions can occur:

- TLB Refill exception occurs when there is no TLB entry that matches a referenced address.
- A TLB Invalid exception occurs when a TLB entry that matches a referenced virtual address is marked as being invalid (with the V bit set to 0).
- The TLB Modified exception occurs when a TLB entry that matches a virtual address referenced by the store instruction is marked as being valid (with the V bit set to 1).

The following three sections describe these TLB exceptions.

**(1) TLB refill exception (32-bit space mode)/XTLB refill exception (64-bit space mode)**

**(a) Cause**

The TLB Refill exception occurs when there is no TLB entry to match a reference to a mapped address space. This exception is not maskable.

**(b) Processing**

There are two special exception vectors for this exception; one for references to 32-bit address spaces, and one for references to 64-bit address spaces. The UX, SX, and KX bits of the Status register determine whether the user, supervisor or kernel address spaces referenced are 32-bit or 64-bit spaces. When the EXL bit of the Status register is set to 0, either of these two special vectors is referenced. When the EXL bit is set to 1, the common exception vector is referenced.

This exception sets the TLBL or TLBS code in the ExcCode field of the Cause register. If this exception has been caused by an instruction reference or load operation, TLBL is set. If it has been caused by a store operation, TLBS is set.

When this exception occurs, the BadVAddr, Context, XContext and EntryHi registers hold the virtual address that failed address translation. The EntryHi register also contains the ASID from which the translation fault occurred. The Random register normally contains a valid location in which to place the replacement TLB entry. The contents of the EntryLo register are undefined.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

**(c) Servicing**

To service this exception, the contents of the Context or XContext register are used as a virtual address to fetch memory words containing the physical page frame and access control bits for a pair of TLB entries. The memory word is written into the TLB entry by using the EntryLo0, EntryLo1, or EntryHi register.

It is possible that the physical page frame and access control bits are placed in a page where the virtual address is not resident in the TLB. This condition is processed by allowing a TLB Refill exception in the TLB Refill exception handler. In this case, the common exception vector is used because the EXL bit of the Status register is set to 1.

### (2) TLB invalid exception

#### (a) Cause

The TLB Invalid exception occurs when the TLB entry that matches with the virtual address to be referenced is invalid (the V bit is set to 0). This exception is not maskable.

#### (b) Processing

The common exception vector is used for this exception. The TLBL or TLBS code in the ExcCode field of the Cause register is set. If this exception has been caused by an instruction reference or load operation, TLBL is set. If it has been caused by a store operation, TLBS is set.

When this exception occurs, the BadVAddr, Context, Xcontext, and EntryHi registers contain the virtual address that failed address translation. The EntryHi register also contains the ASID from which the translation fault occurred. The Random register normally stores a valid location in which to place the replacement TLB entry. The contents of the EntryLo register are undefined.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

#### (c) Servicing

Usually, the V bit of a TLB entry is cleared in the following cases:

- ✧ When a virtual address does not exist
- ✧ When the virtual address exists, but is not in main memory (a page fault)
- ✧ When a trap is required on any reference to the page (for example, to maintain a reference bit)

After servicing the cause of a TLB Invalid exception, the TLB entry is located with a TLBP (TLB Probe) instruction, and replaced by an entry with its V bit set to 1.

### (3) TLB modified exception

#### (a) Cause

The TLB Modified exception occurs when the TLB entry that matches with the virtual address referenced by the store instruction is valid (V bit is 1) but is not writable (D bit is 0). This exception is not maskable.

#### (b) Processing

The common exception vector is used for this exception, and the Mod code in the ExcCode field of the Cause register is set.

When this exception occurs, the BadVAddr, Context, Xcontext, and EntryHi registers contain the virtual address that failed address translation. The EntryHi register also contains the ASID from which the translation fault occurred. The contents of the EntryLo register are undefined.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

**(c) Servicing**

The kernel uses the failed virtual address or virtual page number to identify the corresponding access control bits. The page identified may or may not permit write accesses; if writes are not permitted, a write protection violation occurs.

If write accesses are permitted, the page frame is marked dirty (/writable) by the kernel in its own data structures.

The TLBP instruction places the index of the TLB entry that must be altered into the Index register. The word data containing the physical page frame and access control bits (with the D bit set to 1) is loaded to the EntryLo register, and the contents of the EntryHi and EntryLo registers are written into the TLB.

#### 2.6.4.9 Bus error exception

**(1) Cause**

A Bus Error exception is raised by board-level circuitry for events such as bus time-out, local bus parity errors, and invalid physical memory addresses or access types. This exception is not maskable.

A Bus Error exception occurs only when a cache miss refill, uncached reference, or unbuffered write occurs simultaneously. In other words, it occurs when an illegal access is detected during BCU read.

For details of illegal accesses.

**(2) Processing**

The common interrupt vector is used for a Bus Error exception. The IBE or DBE code in the ExcCode field of the Cause register is set, signifying whether the instruction caused the exception by an instruction reference, load operation, or store operation.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

**(3) Servicing**

The physical address at which the fault occurred can be computed from information available in the System Control Coprocessor (CP0) registers.

- If the IBE code in the Cause register is set (indicating an instruction fetch), the virtual address is contained in the EPC register.
- If the DBE code is set (indicating a load or store), the virtual address of the instruction that caused the exception is saved to the EPC register.

The virtual address of the load and store instruction can then be obtained by interpreting the instruction. The physical address can be obtained by using the TLBP instruction and reading the EntryLo register to compute the physical page number.

At the time of this exception, the kernel reports the UNIX SIGBUS (bus error) signal to the current process, but the exception is usually fatal.

#### 2.6.4.10 System call exception

**(1) Cause**

A System Call exception occurs during an attempt to execute the SYSCALL instruction. This exception is not maskable.

**(2) Processing**

The common exception vector is used for this exception, and the Sys code in the ExcCode field of the Cause register is set.

The EPC register contains the address of the SYSCALL instruction unless it is in a branch delay slot, in which case the EPC register contains the address of the preceding branch instruction.

If the SYSCALL instruction is in a branch delay slot, the BD bit of the Status register is set to 1; otherwise this bit is cleared.

**(3) Servicing**

When this exception occurs, control is transferred to the applicable system routine.

To resume execution, the EPC register must be altered so that the SYSCALL instruction does not re-execute; this is accomplished by adding a value of 4 to the EPC register before returning.

If a SYSCALL instruction is in a branch delay slot, interpretation (decoding) of the branch instruction is required to resume execution.

#### 2.6.4.11 Breakpoint exception

**(1) Cause**

A Breakpoint exception occurs when an attempt is made to execute the BREAK instruction. This exception is not maskable.

**(2) Processing**

The common exception vector is used for this exception, and the Bp code in the ExcCode field of the Cause register is set.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1; otherwise this bit is cleared.

**(3) Servicing**

When the Breakpoint exception occurs, control is transferred to the applicable system routine. Additional distinctions can be made by analyzing the unused bits of the BREAK instruction (bits 25 to 6), and loading the contents of the instruction whose address the EPC register contains.

To resume execution, the EPC register must be altered so that the BREAK instruction does not re-execute; this is accomplished by adding a value of 4 to the EPC register before returning.

When a Breakpoint exception occurs while executing the MIPS16 instruction, a value of 2 should be added to the EPC register before returning.

If a BREAK instruction is in a branch delay slot, interpretation (decoding) of the branch instruction is required to resume execution.

#### 2.6.4.12 Coprocessor unusable exception

**(1) Cause**

The Coprocessor Unusable exception occurs when an attempt is made to execute a coprocessor instruction for either:

- ✧ a corresponding coprocessor unit that has not been marked usable (Status register bit, CU0 = 0), or
- ✧ CP0 instructions, when the unit has not been marked usable (Status register bit, CU0 = 0) and the process executes in User or Supervisor mode.

This exception is not maskable.

**(2) Processing**

The common exception vector is used for this exception, and the CPU code in the ExcCode field of the Cause register is set. The CE bit of the Cause register indicates which of the four coprocessors was referenced.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

**(3) Servicing**

The coprocessor unit to which an attempted reference was made is identified by the CE bit of the Cause register. One of the following processing is performed by the handler:

- ✧ If the process is entitled access to the coprocessor, the coprocessor is marked usable and the corresponding state is restored to the coprocessor.
- ✧ If the process is entitled access to the coprocessor, but the coprocessor does not exist or has failed, interpretation of the coprocessor instruction is possible.
- ✧ If the BD bit in the Cause register is set to 1, the branch instruction must be interpreted; then the coprocessor instruction can be emulated and execution resumed with the EPC register advanced past the coprocessor instruction.
- ✧ If the process is not entitled access to the coprocessor, the kernel reports UNIX SIGILL/ILL_PRIVIN_FAULT (illegal instruction/privileged instruction fault) signal to the current process, and this exception is fatal.

#### 2.6.4.13 Reserved instruction exception

**(1) Cause**

The Reserved Instruction exception occurs when an attempt is made to execute one of the following instructions:

- Instruction with an undefined major opcode (bits 31 to 26)
- SPECIAL instruction with an undefined minor opcode (bits 5 to 0)
- REGIMM instruction with an undefined minor opcode (bits 20 to 16)
- 64-bit instructions in 32-bit User or Supervisor mode
- RR instruction with an undefined minor op code (bits 4 to 0) when executing the MIPS16 instruction
- I8 instruction with an undefined minor op code (bits 10 to 8) when executing the MIPS16 instruction

64-bit operations are always valid in Kernel mode regardless of the value of the KX bit in the Status register. This exception is not maskable.

**(2) Processing**

The common exception vector is used for this exception, and the RI code in the ExcCode field of the Cause register is set.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

**(3) Servicing**

All currently defined MIPS ISA instructions can be executed. The process executing at the time of this exception is handled by a UNIX SIGILL/ILL_RESOP_FAULT (illegal instruction/reserved operand fault) signal. This error is usually fatal.

#### 2.6.4.14 Trap exception

**(1) Cause**

The Trap exception occurs when a TGE, TGEU, TLT, TLTU, TEQ, TNE, TGEI, TGEUI, TLTI, TLTUI, TEQI, or TNEI instruction results in a TRUE condition. This exception is not maskable.

**(2) Processing**

The common exception vector is used for this exception, and the Tr code in the ExcCode field of the Cause register is set.

The EPC register contains the address of the trap instruction causing the exception unless the instruction is in a branch delay slot, in which case the EPC register contains the address of the preceding branch instruction and the BD bit of the Cause register is set to 1.

**(3) Servicing**

At the time of a Trap exception, the kernel reports the UNIX SIGFPE/FPE_INTOVF_TRAP (floating-point exception/integer overflow) signal to the current process, but the exception is usually fatal.

#### 2.6.4.15 Integer overflow exception

**(1) Cause**

An Integer Overflow exception occurs when an ADD, ADDI, SUB, DADD, DADDI, or DSUB instruction results in a 2's complement overflow. This exception is not maskable.

**(2) Processing**

The common exception vector is used for this exception, and the Ov code in the ExcCode field of the Cause register is set.

The EPC register contains the address of the instruction that caused the exception unless the instruction is in a branch delay slot, in which case the EPC register contains the address of the preceding branch instruction and the BD bit of the Cause register is set to 1.

**(3) Servicing**

At the time of the exception, the kernel reports the UNIX SIGFPE/FPE_INTOVF_TRAP (floating-point exception/integer overflow) signal to the current process, and this exception is usually fatal.

#### 2.6.4.16 Watch exception

**(1) Cause**

A Watch exception occurs when a load or store instruction references the physical address specified by the WatchLo/WatchHi registers. The WatchLo/WatchHi registers specify whether a load or store or both could have initiated this exception.

- When the R bit of the WatchLo register is set to 1: Load instruction
- When the W bit of the WatchLo register is set to 1: Store instruction
- When both the R bit and W bit of the WatchLo register are set to 1: Load instruction or store instruction

The CACHE instruction never causes a Watch exception.

The Watch exception is postponed while the EXL bit in the Status register is set to 1, and Watch exception is only maskable by setting the EXL bit in the Status register to 1.

**(2) Processing**

The common exception vector is used for this exception, and the Watch code in the ExcCode field of the Cause register is set.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

**(3) Servicing**

The Watch exception is a debugging aid; typically the exception handler transfers control to a debugger, allowing the user to examine the situation. To continue, once the Watch exception must be disabled to execute the faulting instruction. The Watch exception must then be reenabled. The faulting instruction can be executed either by the debugger or by setting breakpoints.

#### 2.6.4.17 Interrupt exception

**(1) Cause**

The Interrupt exception occurs when one of the eight interrupt conditions[Note] is asserted. In the VR4120A CPU, interrupt requests from internal peripheral units first enter the ICU and are then notified to the CPU core via one of four interrupt sources (Int(3:0)) or NMI.

Each of the eight interrupts can be masked by clearing the corresponding bit in the IM field of the Status register, and all of the eight interrupts can be masked at once by clearing the IE bit of the Status register or setting the EXL/ERL bit.

**Note** They are 1 timer interrupt, 5 ordinary interrupts, and 2 software interrupts.

Of the five ordinary interrupts, Int4 is never asserted active in the VR4120A CPU.

**(2) Processing**

The common exception vector is used for this exception, and the Int code in the ExcCode field of the Cause register is set.

The IP field of the Cause register indicates current interrupt requests. It is possible that more than one of the bits can be simultaneously set (or cleared) if the interrupt request signal is asserted and then deasserted before this register is read.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

**(3) Servicing**

If the interrupt is caused by one of the two software-generated exceptions (SW0 or SW1), the interrupt condition is cleared by setting the corresponding Cause register bit to 0.

If the interrupt is caused by hardware, the interrupt condition is cleared by deactivating the corresponding interrupt request signal.

### 2.6.5 Exception processing and servicing flowcharts

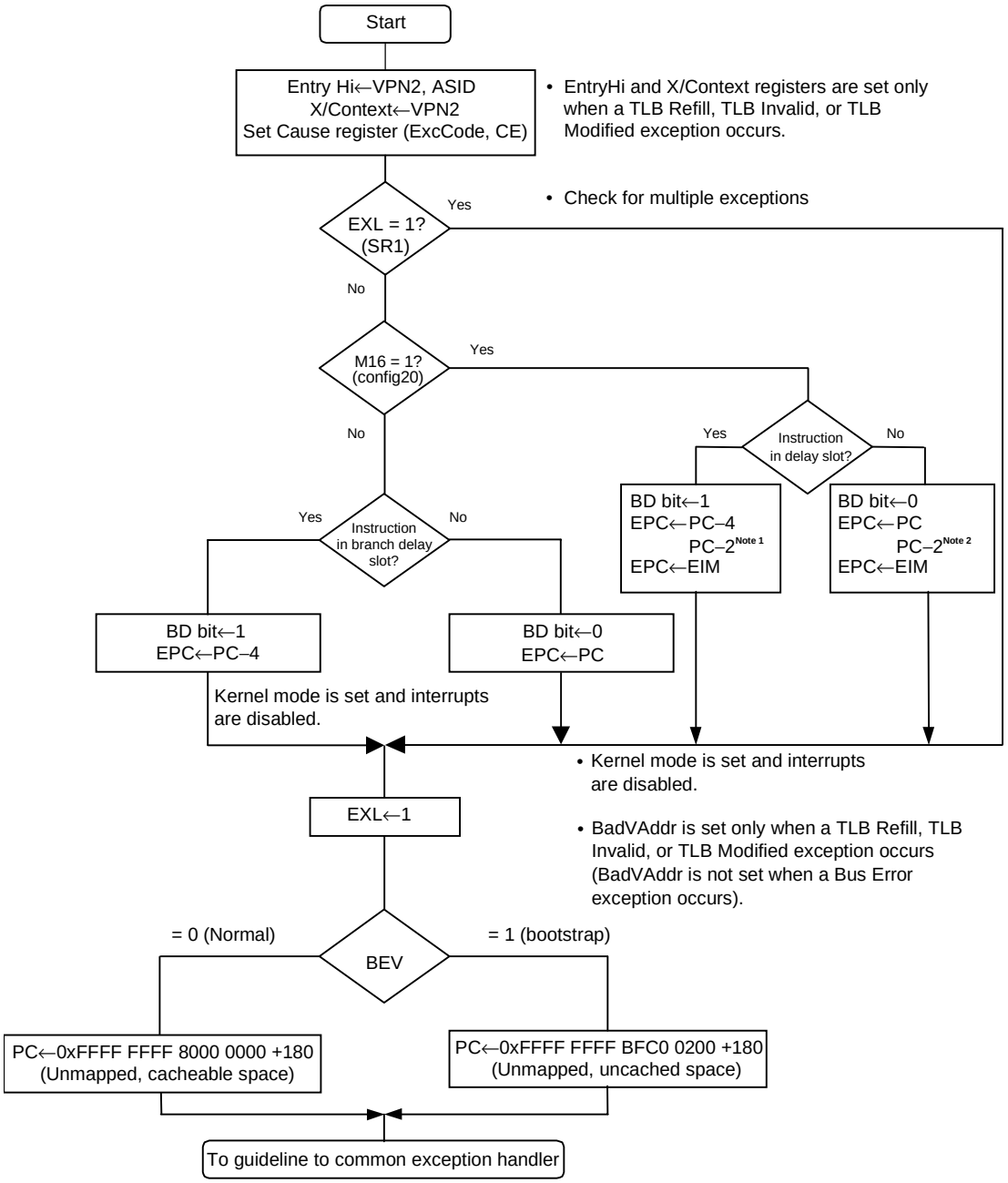
The remainder of this chapter contains flowcharts for the following exceptions and guidelines for their handlers:

- ✧ Common exceptions and a guideline to their exception handler
- ✧ TLB/XTLB Refill exception and a guideline to their exception handler
- ✧ Cold Reset, Soft Reset and NMI exceptions, and a guideline to their handler.

Generally speaking, the exceptions are "processed" by hardware (HW); the exceptions are then "serviced" by software (SW).

**Figure 2-73. Common Exception Handling (1/2)**

**(a) Handling exceptions other than Cold reset, Soft reset, NMI, and TLB/XTLB Refill (hardware)**




**Notes 1.** When the JR or JALR instruction of MIPS16 instructions

**2.** When the Extend instruction of MIPS16 instructions

**Remark** The interrupts can be masked by setting the IE or IM bit.
The Watch exception can be set to pending state by setting the EXL bit to 1.

**Figure 2-73. Common Exception Handling (2/2)**

**(b) Servicing common exceptions (software)**

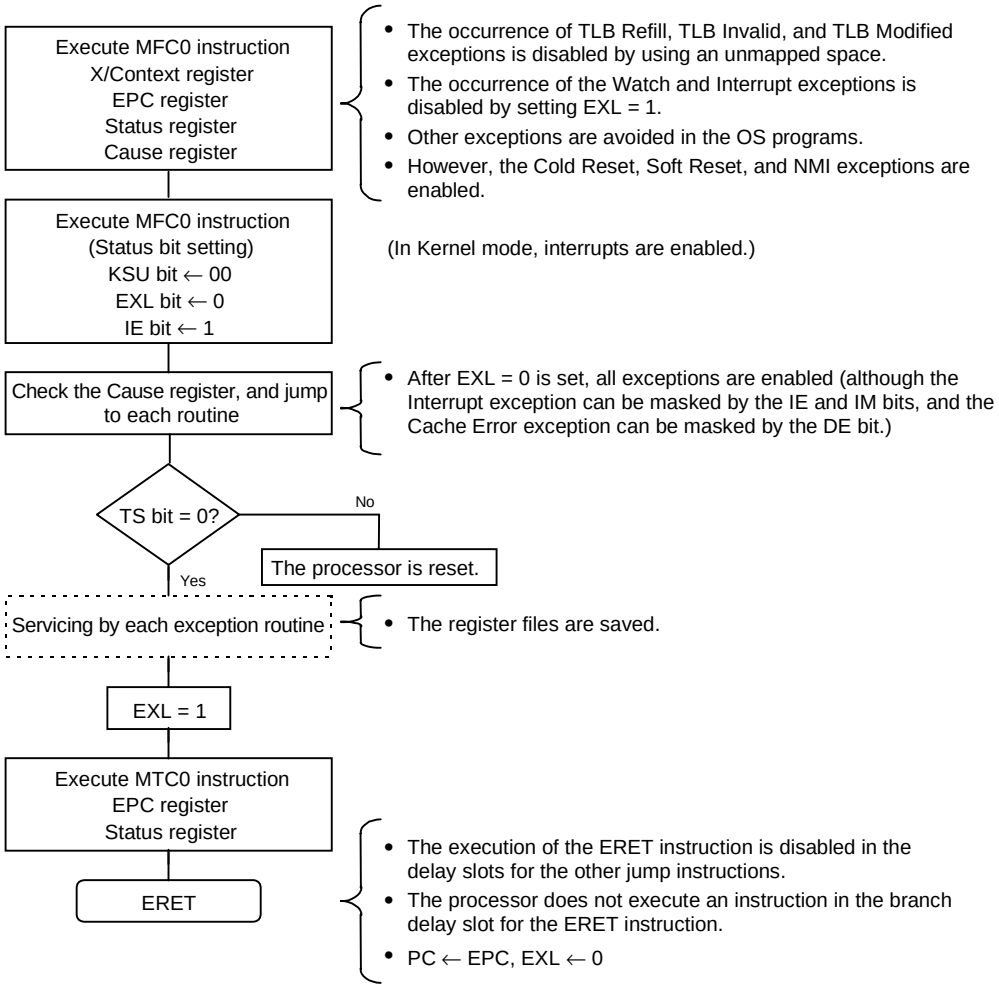
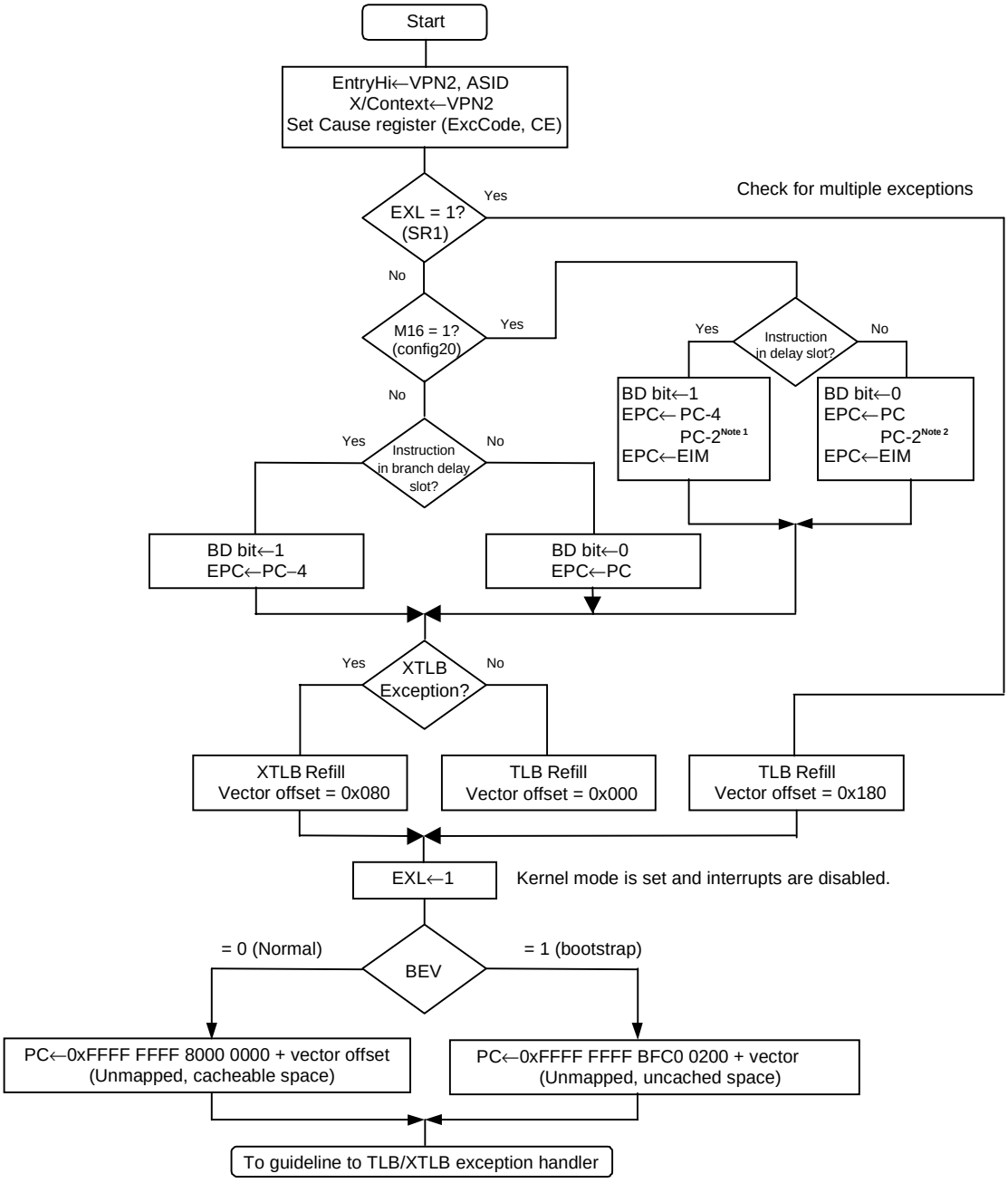


**Figure 2-74. TLB/XTLB Refill Exception Handling (1/2)**

**(a) Handling TLB/XTLB Refill exceptions (hardware)**




**Notes 1.** When the JR or JALR instruction of MIPS16 instructions

**2.** When the Extend instruction of MIPS16 instructions

**Figure 2-74. TLB/XTLB Refill Exception Handling (2/2)**

**(b) Servicing TLB/XTLB Refill Exceptions (Software)**

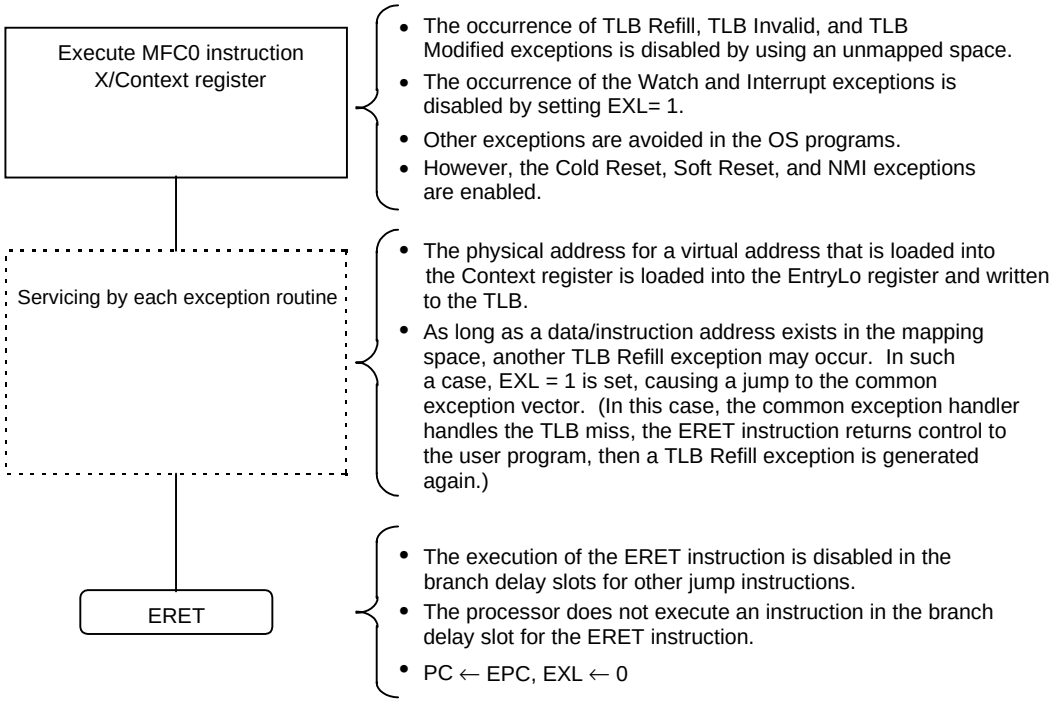
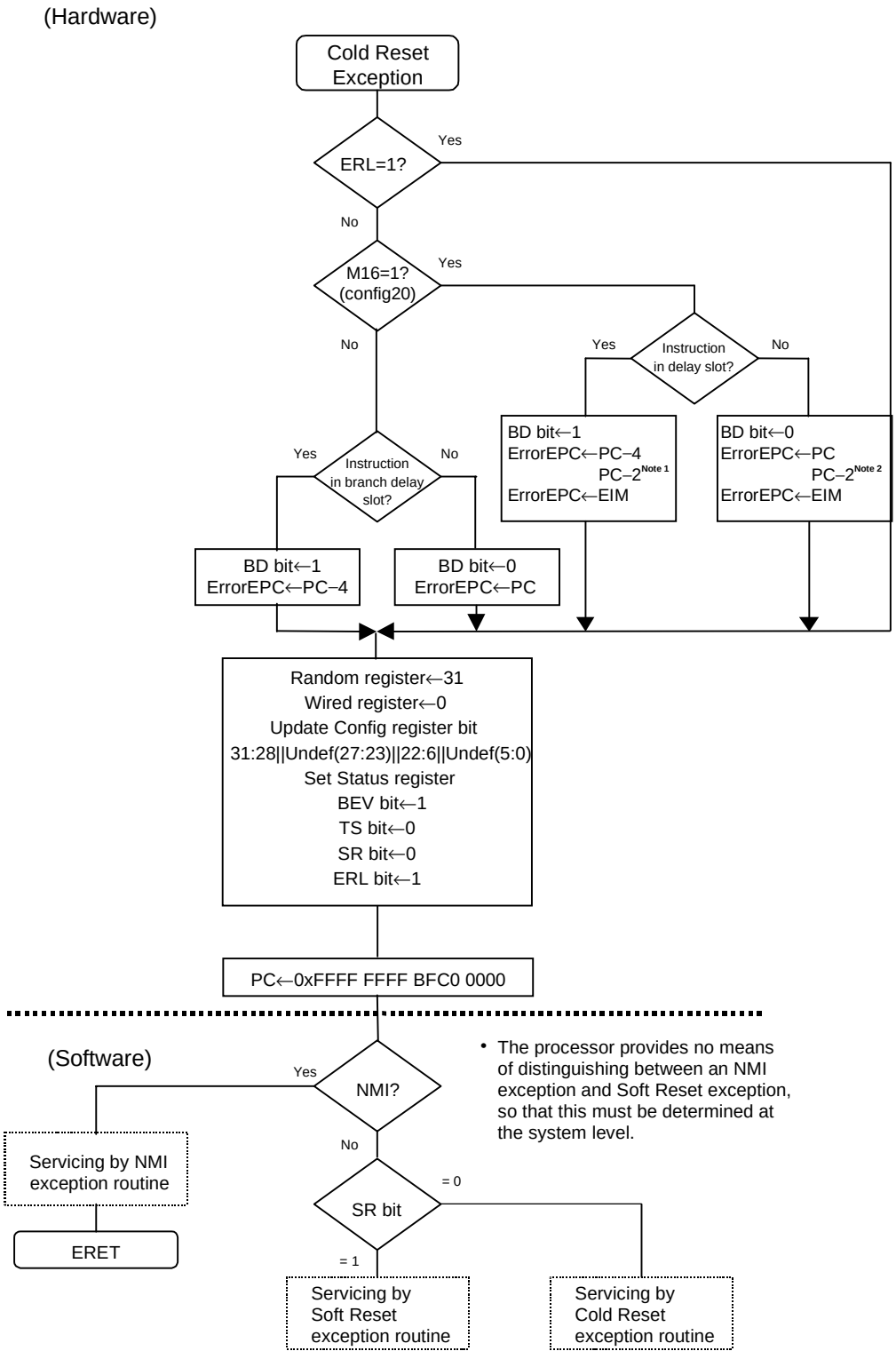


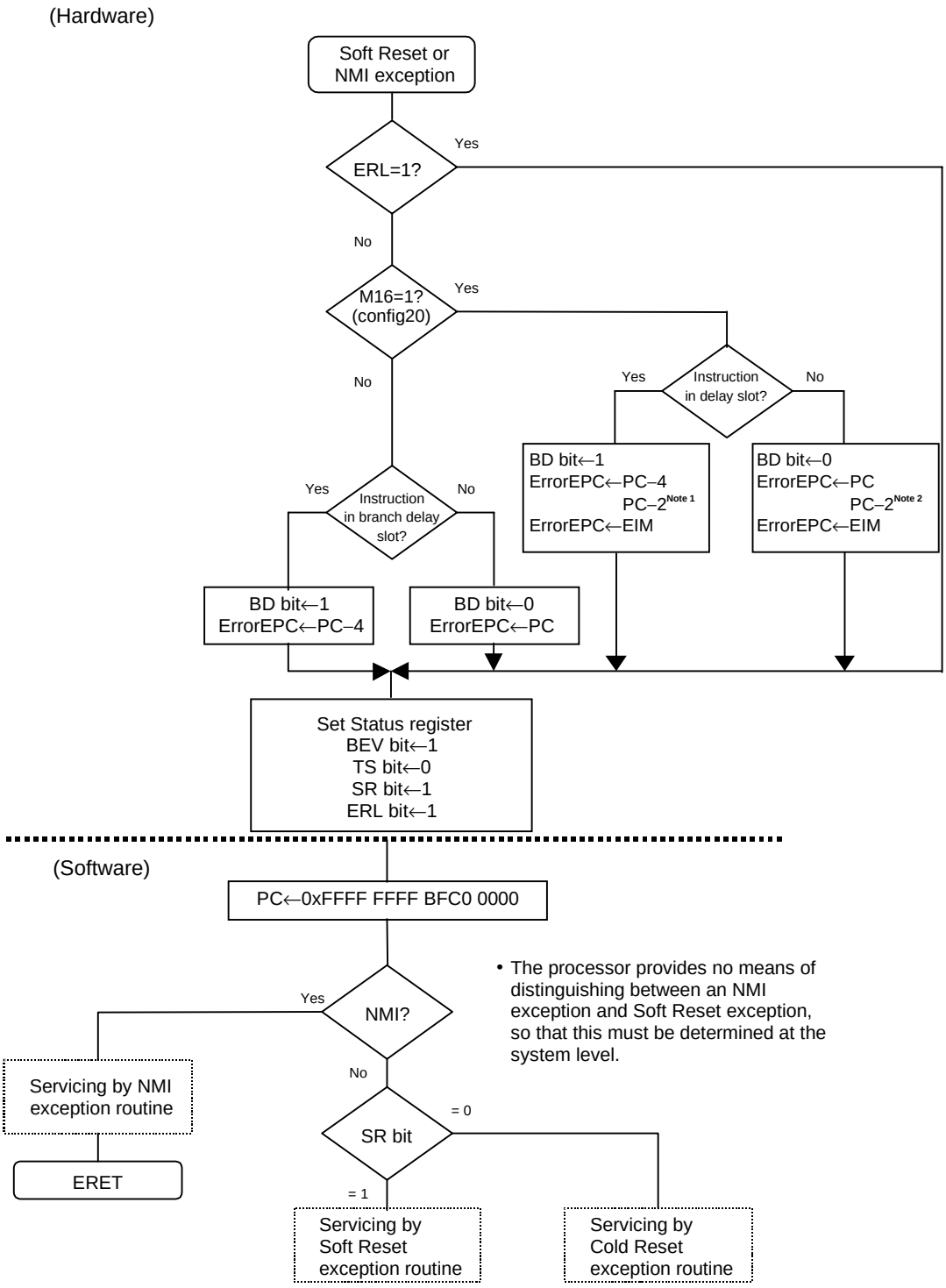
**Figure 2-75. Cold Reset Exception Handling**




**Notes 1.** When the JR or JALR instruction of MIPS16 instructions

**2.** When the Extend instruction of MIPS16 instructions

**Figure 2-76. Soft Reset and NMI Exception Handling**




**Notes 1.** When the JR or JALR instruction of MIPS16 instructions

**2.** When the Extend instruction of MIPS16 instructions

## 2.7 Initialization Interface

This section describes the reset sequence of the VR4120A Core. For details about factors of reset or reset of the whole VR4120A Core.

### 2.7.1 Cold reset

In the VR4120A Core, a cold reset sequence is executed in the CPU core in the following cases:

- Hardware reset
- Deadman's SW shutdown
- Software shutdown
- HAL Timer shutdown

A Cold Reset completely initializes the CPU core, except for the following register bits.

- The TS and SR bits of the Status register are cleared to 0.
- The ERL and BEV bits of the Status register are set to 1.
- The upper limit value (31) is set in the Random register.
- The Wired register is initialized to 0.
- Bits 31 to 28 of the Config register are set to 0 and bits 22 to 3 to 0x04800; the other bits are undefined.
- The values of the other registers are undefined.

### 2.7.2 Soft reset

A Soft Reset initializes the CPU core without affecting the clocks; in other words, a Soft Reset is a logic reset. In a Soft Reset, the CPU core retains as much state information as possible; all state information except for the following is retained:

- The TS bit of the Status register is cleared to 0.
- The SR, ERL and BEV bits of the Status register are set to 1.
- The Count register is initialized to 0.
- The IP7 bit of the Cause register is cleared to 0.
- Any Interrupts generated on the SysAD bus are cleared.
- NMI is cleared.
- The Config register is initialized.

### 2.7.3 VR4120A processor modes

The VR4120A supports various modes, which can be selected by the user. The CPU core mode is set each time a write occurs in the Status register and Config register. The on-chip peripheral unit mode is set by writing to the I/O register.

This section describes the CPU core's operation modes. For operation modes of on-chip peripheral units, see the chapters describing the various units.

#### 2.7.3.1 Power modes

The VR4120A supports four power modes: Fullspeed mode, Standby mode, Suspend mode, and Hibernate mode.

**(1) Fullspeed mode**

This is the normal operation mode.

The VR4120A's default status sets operation under Fullspeed mode. After the processor is reset, the VR4120A returns to Fullspeed mode.

**(2) Standby mode**

When a STANDBY instruction has been executed, the processor can be set to Standby mode. During Standby mode, all of the internal clocks in the CPU core except for the timer and interrupt clocks are held at high level. The peripheral units all operate as they do during Fullspeed mode. This means that DMA operations are enabled during Standby mode.

When the STANDBY instruction completes the WB stage, the VR4120A remains idle until the SysAD internal bus enters the idle state. Next, the clocks in the CPU core are shut down and pipeline operation is stopped. However, the PLL, timer, and interrupt clocks continue to operate, as do the internal bus clocks (TClock and MasterOut).

During Standby mode, the processor is returned to Fullspeed mode if any interrupt occurs, including a timer interrupt that occurs internally.

**(3) Suspend mode**

When the SUSPEND instruction has been executed, the processor can be set to Suspend mode. During Suspend mode, the processor stalls the pipeline and supplying all of the internal clocks in the CPU core except for PLL timer and interrupt clocks are stopped. The VR4120A stops supplying TClock to peripheral units. Accordingly, during Suspend mode peripheral units can only be activated by a special interrupt unit (DCD_B control, etc.). While in this mode, the register and cache contents are retained.

When the SUSPEND instruction completes the WB stage, the VR4120A switches the DRAM to self refresh mode and then waits for the SysAD internal bus to enter the idle state. Next, the clocks in the CPU core are shut down and pipeline operation is stopped. The VR4120A then stops supplying TClock to peripheral units. However, the PLL, timer, and interrupt clocks continue to operate, as do the MasterOut.

The processor remains in Suspend mode until an interrupt is received, at which time it returns to Fullspeed mode.

**(4) Hibernate mode**

When the HIBERNATE instruction has been executed, the processor can be set to Hibernate mode. During Hibernate mode, the processor stops supplying clocks to all units. The register and cache contents are retained and output of TClock and MasterOut is stopped. The processor remains in Hibernate mode until the POWER pin is asserted or a WakeUpTimer interrupt occurs at which the processor returns to Fullspeed mode.

In this mode, supplying voltage to the 2.5-V power-supply systems (VDD2, VDDP, VDDPD) can be stopped. When the voltage of the 2.5-V power supplies becomes 0 V, the power dissipation becomes almost 0 W (it is not exactly 0 V because there are a 32.768-kHz oscillator and on-chip peripheral circuits operating at 32.768 kHz).

### 2.7.3.2 Privilege mode

The VR4120A supports three system modes: kernel expanded addressing mode, supervisor expanded addressing mode, and user expanded addressing mode. These three modes are described below.

**(1) Kernel expanded addressing mode**

When the Status register's KX bit has been set, an expanded TLB miss exception vector is used when a TLB miss occurs for the kernel address. While in kernel mode, the MIPS III operation code can always be used, regardless of the KX bit.

**(2) Supervisor expanded addressing mode**

When the Status register's SX bit has been set, the MIPS III operation code can be used when in supervisor mode and an expanded TLB miss exception vector is used when a TLB miss occurs for the supervisor address.

**(3) User expanded addressing mode**

When the Status register's UX bit has been set, the MIPS III operation code can be used when in user mode, and an expanded TLB miss exception vector is used when a TLB miss occurs for the user address. When this bit is cleared, the MIPS I and II operation codes can be used, as can 32-bit virtual addresses.

#### 2.7.3.3 Reverse endian

When the Status register's RE bit has been set, the endian ordering is reversed to adopt the user software's perspective. However, the RE bit of the Status register must be set to 0 since the VR4120A supports the little-endian order only.

#### 2.7.3.4 Bootstrap exception vector (BEV)

The BEV bit is used to generate an exception during operation testing (diagnostic testing) of the cache and main memory system. This bit is automatically set to 1 after reset or NMI exception.

When the Status register's BEV bit has been set, the address of the TLB miss exception vector is changed to the virtual address 0xFFFF FFFF BFC0 0200 and the ordinary execution vector is changed to address 0xFFFF FFFF BFC0 0380.

When the BEV bit is cleared, the TLB miss exception vector's address is changed to 0xFFFF FFFF 8000 0000 and the ordinary execution vector is changed to address 0xFFFF FFFF 8000 0180.

#### 2.7.3.5 Cache error check

The Status register's CE bit has no meaning because the VR4120A does not support cash parity.

#### 2.7.3.6 Parity error prohibit

When the Status register's DE bit has been set, the processor does not issue any cache parity error exceptions.

#### 2.7.3.7 Interrupt enable (IE)

When the Status register's IE bit has been cleared, no interrupts can be received except for reset interrupts and nonmaskable interrupts.

## 2.8 Cache Memory

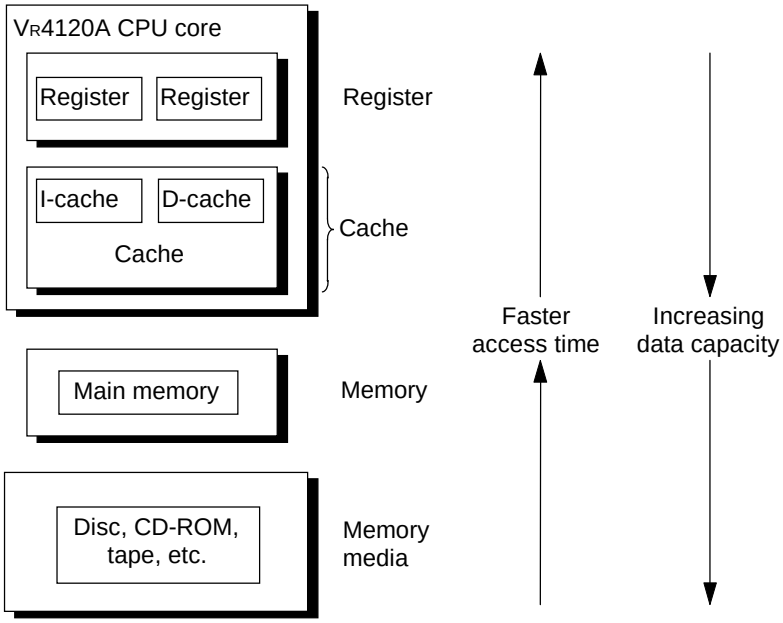
This section describes in detail the cache memory: its place in the VR4120A Core memory organization, and individual organization of the caches.

### 2.8.1 Memory organization

Figure 2-77 shows the VR4120A Core system memory hierarchy. In the logical memory hierarchy, the caches lie between the CPU and main memory. They are designed to make the speedup of memory accesses transparent to the user.

Each functional block in Figure 2-77 has the capacity to hold more data than the block above it. For instance, main memory (physical memory) has a larger capacity than the caches. At the same time, each functional block takes longer to access than any block above it. For instance, it takes longer to access data in main memory than in the CPU on-chip registers.

**Figure 2-77. Logical Hierarchy of Memory**



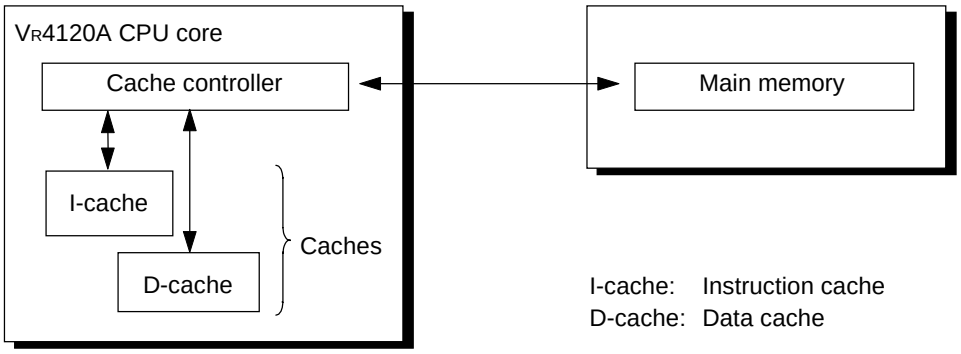

The VR4120A has two on-chip caches: one holds instructions (the instruction cache), the other holds data (the data cache). The instruction and data caches can be read in one PClock cycle.

2 PCycles are needed to write data. However, data writes are pipelined and can complete at a rate of one per PClock cycle. In the first stage of the cycle, the store address is translated and the tag is checked; in the second stage, the data is written into the data RAM.

### 2.8.2 Cache organization

This section describes the organization of the on-chip data and instruction caches. Figure 2-78 provides a block diagram of the VR4120A Core cache and memory model.

**Figure 2-78. Cache Support**




**(1) Cache line lengths**

A cache line is the smallest unit of information that can be fetched from main memory for the cache, and that is represented by a single tag.

The line size for the instruction/data cache is 4 words (16 bytes).

For the cache tag, see **2.8.2.1 Organization of the instruction cache (I-cache)** and **2.8.2.2 Organization of the data cache (D-cache)**.

**(2) Cache sizes**

The instruction cache in the VR4120A Core is 16 Kbytes; the data cache is 8 Kbytes.

#### 2.8.2.1 Organization of the instruction cache (I-cache)

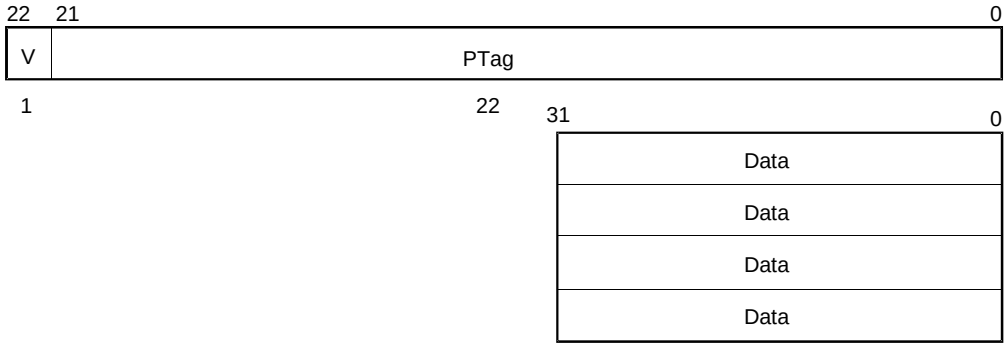
Each line of I-cache data (although it is actually an instruction, it is referred to as data to distinguish it from its tag) has an associated 23-bit tag that contains a 22-bit physical address, and a single valid bit.

The VR4120A Core I-cache has the following characteristics:

- ✧ direct-mapped
- ✧ indexed with a virtual address
- ✧ checked with a physical tag
- ✧ organized with a 4-word (16-byte) cache line

Figure 2-79 shows the format of a 4-word (16-byte) I-cache line.

**Figure 2-79. Instruction Cache Line Format**




PTag : Physical tag (bits 31 to 10 of physical address)
V : Valid bit
Data : Cache data

### 2.8.2.2 Organization of the data cache (D-cache)

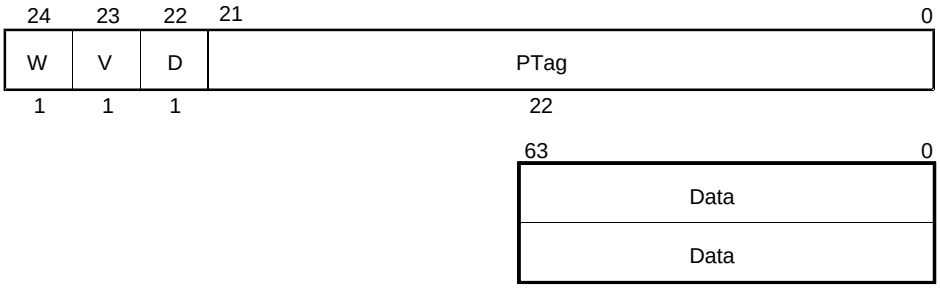
Each line of D-cache data has an associated 25-bit tag that contains a 22-bit physical address, a Valid bit, a Dirty bit, and a Write-back bit.

The VR4120A Core D-cache has the following characteristics :

- ✧ write-back
- ✧ direct-mapped
- ✧ indexed with a virtual address
- ✧ checked with a physical tag
- ✧ organized with a 4-word (16-byte) cache line.

Figure 2-80 shows the format of a 4-word (16-byte) D-cache line.

**Figure 2-80. Data Cache Line Format**




W : Write-back bit (set if cache line has been written)
D : Dirty bit
V : Valid bit
PTag : Physical tag (bits 31 to 10 of physical address)
Data : D-cache data

### 2.8.2.3 Accessing the caches

Figure 2-81 shows the virtual address (VA) index into the caches. The number of virtual address bits used to index the instruction and data caches depends on the cache size.

**(1) Data cache addressing**

Using VA (12:4). The most-significant bit is VA12 because the cache size is 8 Kbytes.

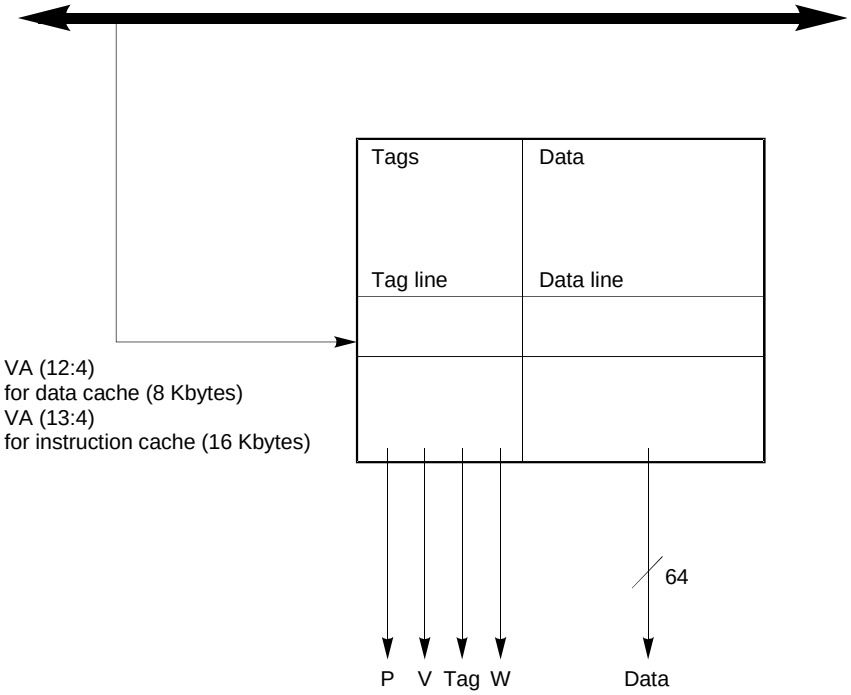
The least-significant bit is VA4 because the line size is 4 words (16 bytes).

**(2) Instruction cache addressing**

Using VA (13:4). The most-significant bit is VA13 because the cache size is 16 Kbytes.

The least-significant bit is VA4 because the line size is 4 words (16 bytes).

**Figure 2-81. Cache Data and Tag Organization**




### 2.8.3 Cache operations

As described earlier, caches provide temporary data storage, and they make the speedup of memory accesses transparent to the user. In general, the CPU core accesses cache-resident instructions or data through the following procedure:

1. The CPU core, through the on-chip cache controller, attempts to access the next instruction or data in the appropriate cache.
2. The cache controller checks to see if this instruction or data is present in the cache.
   - ✧ If the instruction/data is present, the CPU core retrieves it. This is called a cache hit.
   - ✧ If the instruction/data is not present in the cache, the cache controller must retrieve it from memory. This is called a cache miss.
3. The CPU core retrieves the instruction/data from the cache and operation continues.

It is possible for the same data to be in two places simultaneously: main memory and cache. This data is kept consistent through the use of a write-back methodology; that is, modified data is not written back to memory until the cache line is to be replaced.

Instruction and data cache line replacement operations are described in the following sections.

#### 2.8.3.1 Cache write policy

The VR4120A Core manages its data cache by using a write-back policy; that is, it stores write data into the cache, instead of writing it directly to memory[Note]. Some time later this data is independently written into memory. In the VR4120A implementation, a modified cache line is not written back to memory until the cache line is to be replaced either in the course of satisfying a cache miss, or during the execution of a write-back CACHE instruction.

When the CPU core writes a cache line back to memory, it does not ordinarily retain a copy of the cache line, and the state of the cache line is changed to invalid.

**Note** Contrary to the write-back, the write-through cache policy stores write data into the memory and cache simultaneously.

### 2.8.4 Cache states

#### (1) Cache line

The three terms below are used to describe the state of a cache line:

- ✧ Dirty: a cache line containing data that has changed since it was loaded from memory.
- ✧ Clean: a cache line that contains data that has not changed since it was loaded from memory.
- ✧ Invalid: a cache line that does not contain valid information must be marked invalid, and cannot be used. For example, after a Soft Reset, software sets all cache lines to invalid. A cache line in any other state than invalid is assumed to contain valid information. Neither Cold reset nor Soft reset makes the cache state invalid. Software makes the cache state invalid.

#### (2) Data cache

The data cache supports three cache states:

- ✧ Invalid
- ✧ Valid clean
- ✧ Valid dirty

#### (3) Instruction cache

The instruction cache supports two cache states:

- ✧ Invalid
- ✧ Valid

The state of a valid cache line may be modified when the processor executes a CACHE operation. CACHE operations are described in **APPENDIX A MIPS III INSTRUCTION SET DETAILS**.

### 2.8.5 Cache state transition diagrams

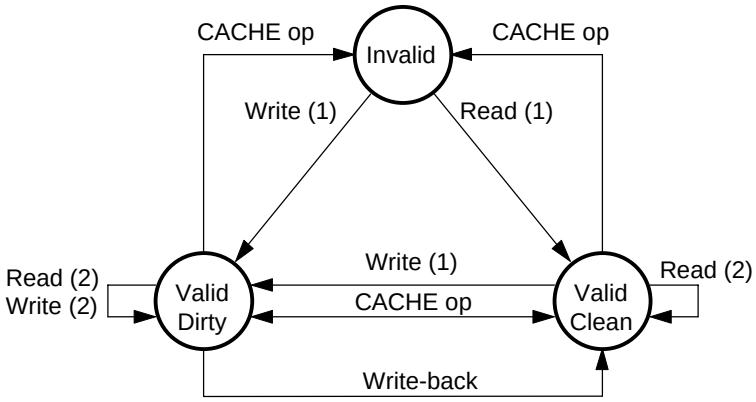
The following section describes the cache state diagrams for the data and instruction cache lines. These state diagrams do not cover the initial state of the system, since the initial state is system-dependent.

#### 2.8.5.1 Data cache state transition

The following diagram illustrates the data cache state transition sequence. A load or store operation may include one or more of the atomic read and write operations shown in the state diagram below, which may cause cache state transitions.

- ✧ Read (1) indicates a read operation from main memory to cache, inducing a cache state transition.
- ✧ Read (2) indicates a read operation from cache to the CPU core, which induces no cache state transition.
- ✧ Write (1) indicates a write operation from CPU core to cache, inducing a cache state transition.
- ✧ Write (2) indicates a write operation from CPU core to cache, which induces no cache state transition.

**Figure 2-82. Data Cache State Diagram**

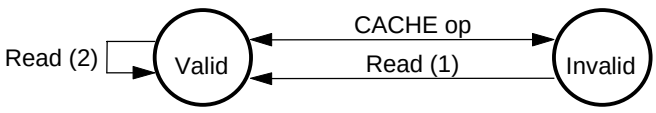



#### 2.8.5.2 Instruction cache state transition

The following diagram illustrates the instruction cache state transition sequence.

Read (1) indicates a read operation from main memory to cache, inducing a cache state transition.
Read (2) indicates a read operation from cache to the CPU core, which induces no cache state transition.

**Figure 2-83. Instruction Cache State Diagram**



### 2.8.6 Cache data integrity

Figures 2-84 to 2-98 shows checking operations for various cache accesses.

**Figure 2-84. Data Check Flow on Instruction Fetch**

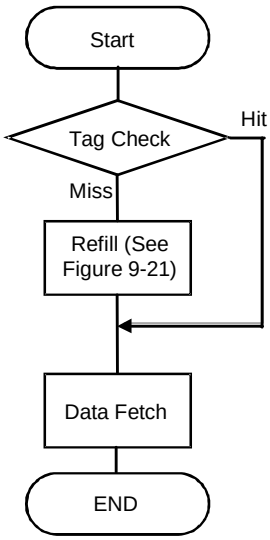



**Figure 2-85. Data Check Flow on Load Operations**

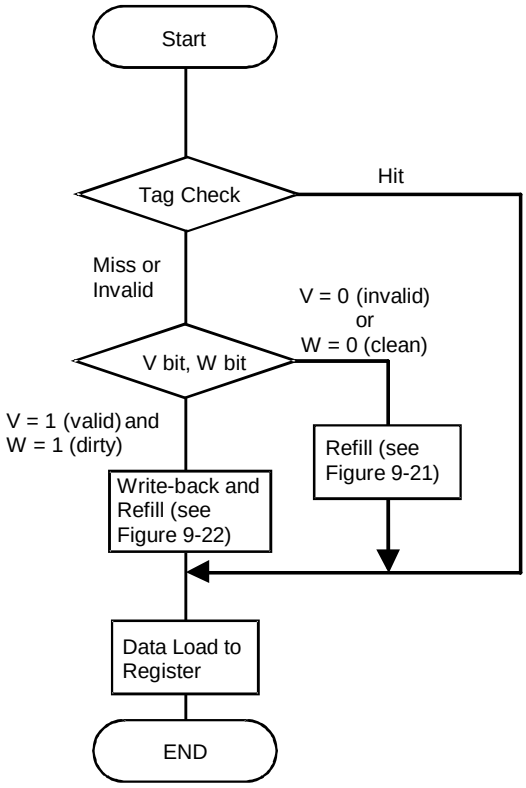


**Figure 2-86. Data Check Flow on Store Operations**

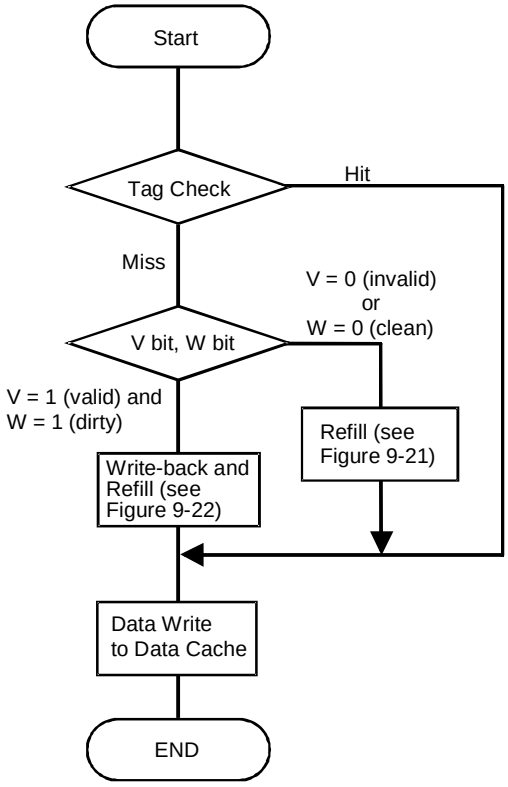



**Figure 2-87. Data Check Flow on Index_Invalidate Operations**

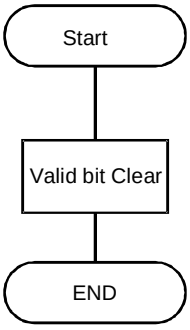


**Figure 2-88. Data Check Flow on Index_Writeback_Invalidate Operations**

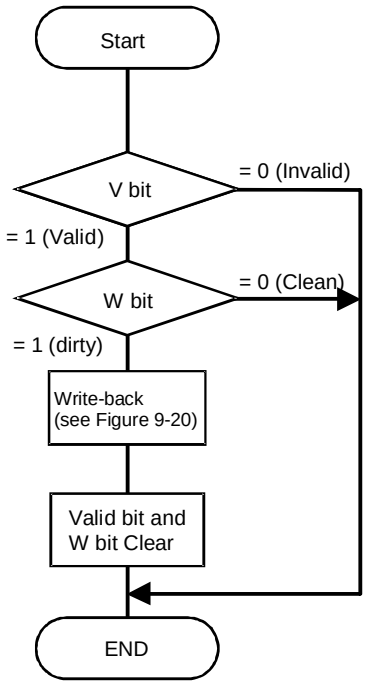



**Figure 2-89. Data Check Flow on Index_Load_Tag Operations**

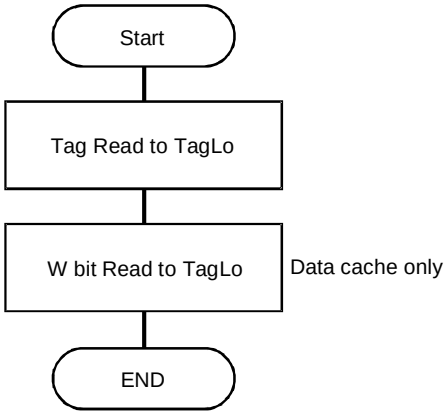



**Figure 2-90. Data Check Flow on Index_Store_Tag Operations**

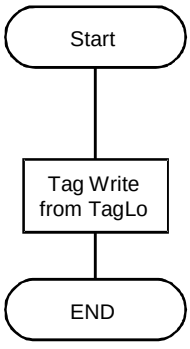


**Figure 2-91. Data Check Flow on Create_Dirty Operations**

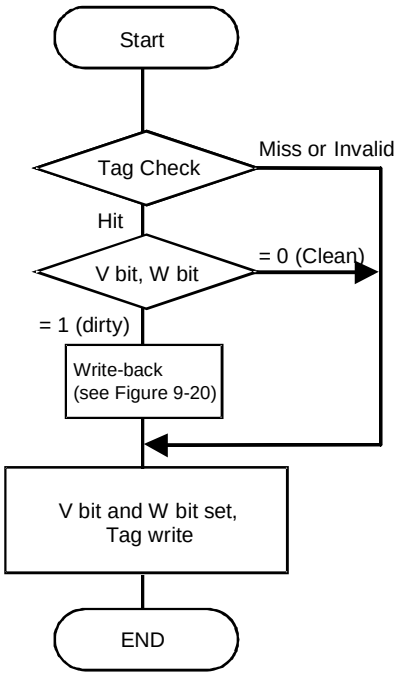



**Figure 2-92. Data Check Flow on Hit_Invalidate Operations**

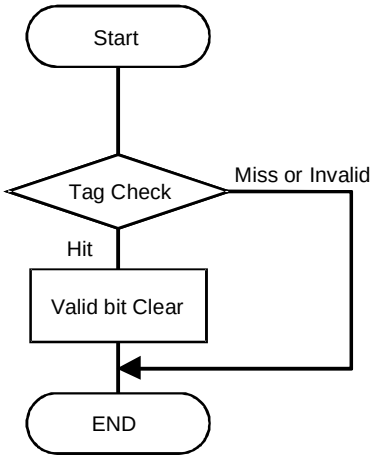


**Figure 2-93. Data Check Flow on Hit_Writeback_Invalidate Operations**

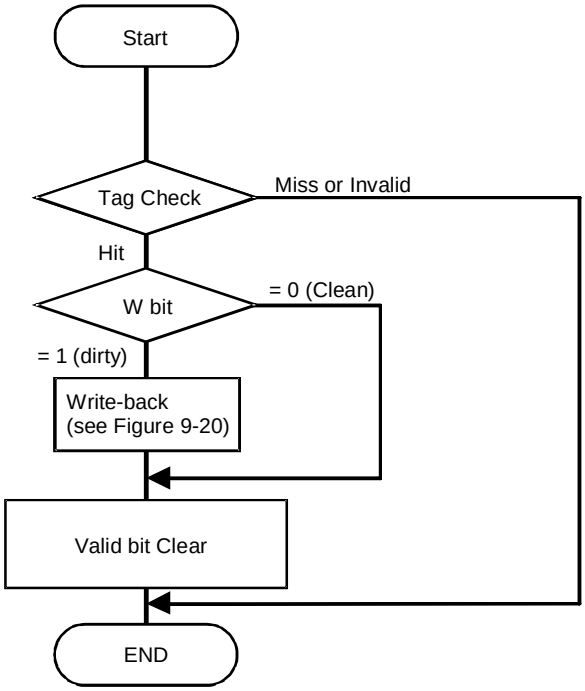



**Figure 2-94. Data Check Flow on Fill Operations**

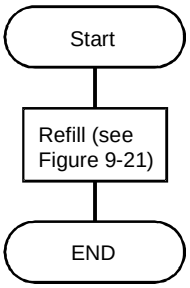


**Figure 2-95. Data Check Flow on Hit_Writeback Operations**

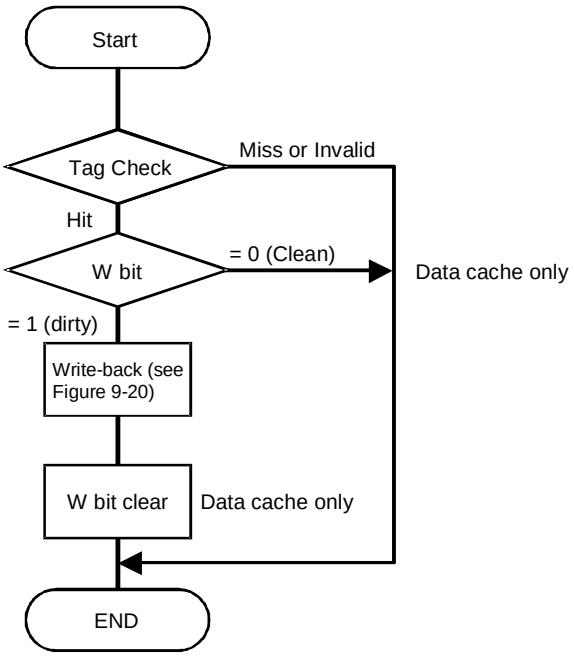



**Figure 2-96. Writeback Flow**

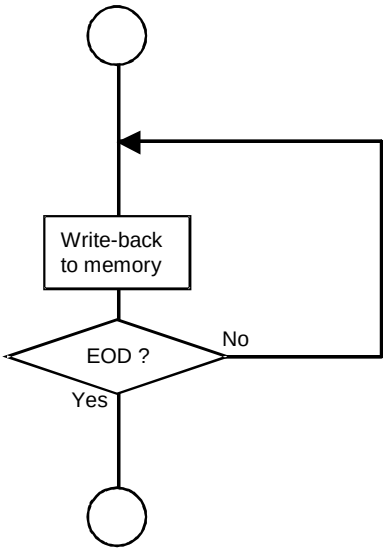


**Figure 2-97. Refill Flow**

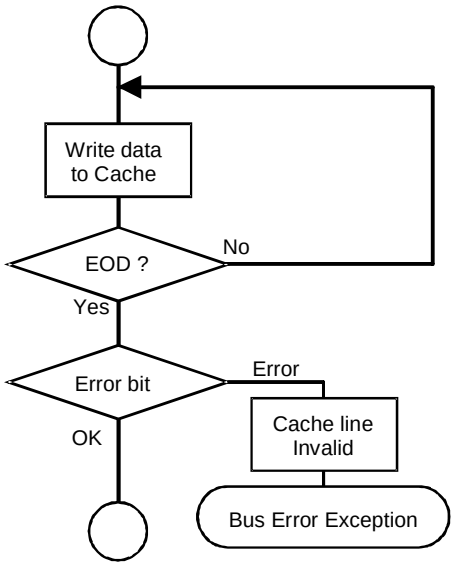
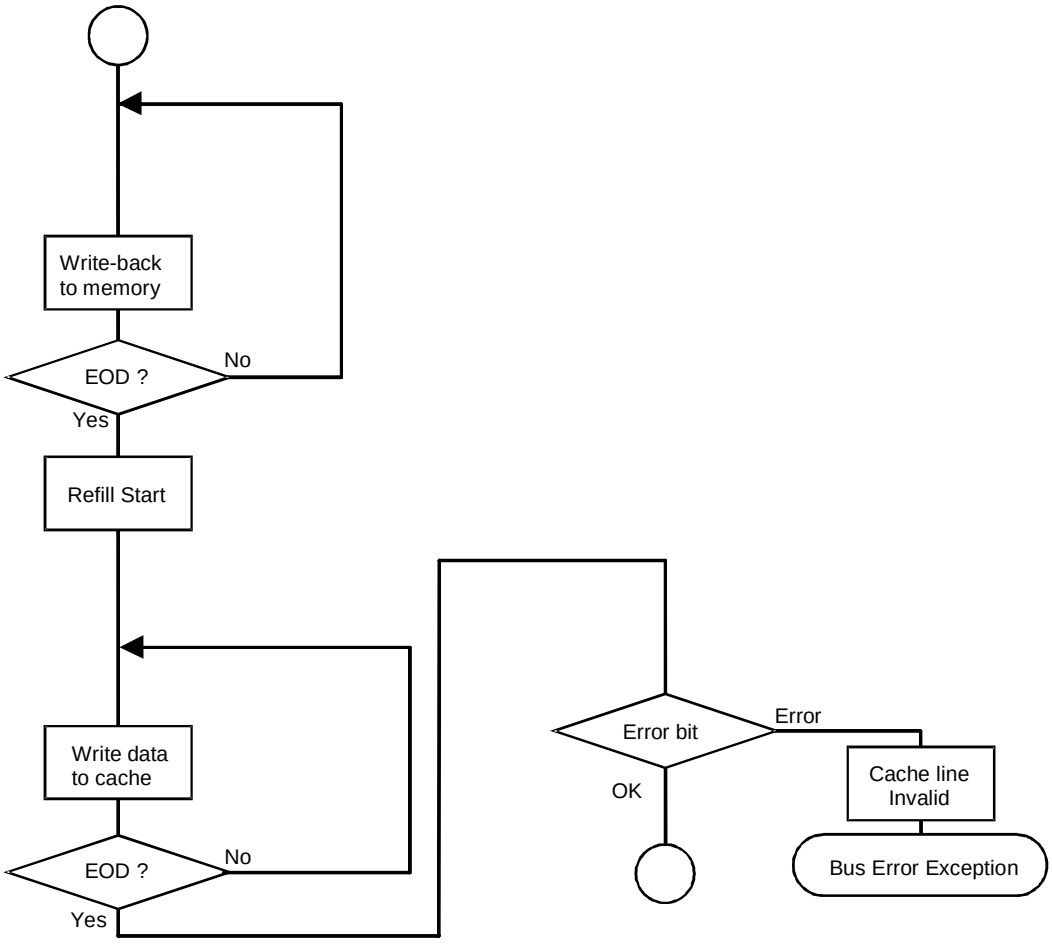



**Figure 2-98. Writeback & Refill Flow**




**Remark** Write-back Procedure:

On a store miss write-back, data tag is checked and data is transferred to the write buffer. If an error is detected in the data field, the write back is not terminated; the erroneous data is still written out to main memory. If an error is detected in the tag field, the write-back bus cycle is not issued.

The cache data may not be checked during CACHE operation.

### 2.8.7 Manipulation of the caches by an external agent

The VR4120A does not provide any mechanisms for an external agent to examine and manipulate the state and contents of the caches.

## 2.9 CPU Core Interrupts

Four types of interrupt are available on the CPU core. These are:

- ✧ one non-maskable interrupt, NMI
- ✧ five ordinary interrupts
- ✧ two software interrupts
- ✧ one timer interrupt

For the interrupt request input to the CPU core.

### 2.9.1 Non-maskable interrupt (NMI)

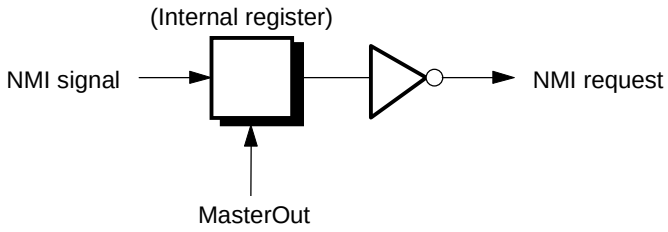
The non-maskable interrupt is acknowledged by asserting the NMI signal (internal), forcing the processor to branch to the Reset Exception vector. This signal is latched into an internal register at the rising edge of MasterOut signal (internal), as shown in Figure 2-99.

NMI only takes effect when the processor pipeline is running.

This interrupt cannot be masked.

Figure 2-99 shows the internal service of the NMI signal. The NMI signal is latched into an internal register by the rising edge of MasterOut. The latched signal is inverted to be transferred to inside the device as an NMI request.

**Figure 2-99. Non-maskable Interrupt Signal**




### 2.9.2 Ordinary interrupts

Ordinary interrupts are acknowledged by asserting the Int(4:0) signals (internal). However, Int4 never occurs in the VR4120A.

This interrupt request can be masked with the IM (6:2), IE, and EXL fields of the Status register.

### 2.9.3 Software interrupts generated in CPU core

Software interrupts generated in the CPU core use bits 1 and 0 of the IP (interrupt pending) field in the Cause register. These may be written by software, but there is no hardware mechanism to set or clear these bits.

After the processing of a software interrupt exception, corresponding bit of the IP field in the Cause register must be cleared before returning to ordinary routine or enabling multiple interrupts until the operation returns to normal routine.

This interrupt request is maskable through the IM (1:0), IE, and EXL fields of the Status register.

### 2.9.4 Timer interrupt

The timer interrupt uses bit 7 of the IP (interrupt pending) field of the Cause register. This bit is set automatically whenever the value of the Count register equals the value of the Compare register, and an interrupt request is acknowledged.

This interrupt is maskable through IM7 of the IM field of the Status register.

### 2.9.5 Asserting interrupts

#### 2.9.5.1 Detecting hardware interrupts

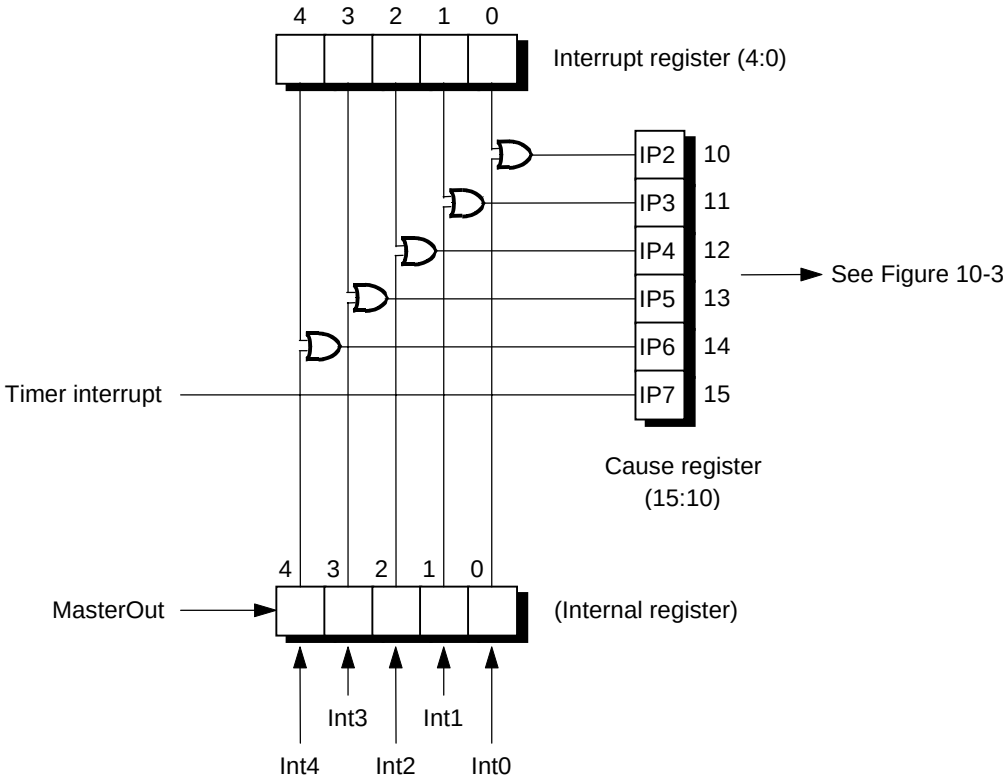
Figure 2-100 shows how the hardware interrupts are readable through the Cause register.

The timer interrupt signal, IP7, is directly readable as bit 15 of the Cause register.

Bits 4 to 0 of the Interrupt register are bit-wise ORed with the current value of the Int4 to 0 signals and the result is directly readable as bits 14 to 10 of the Cause register.

IP1 and IP0 of the Cause register, which are described in **Section 2.6 Exception Processing**, are software interrupts. There is no hardware mechanism for setting or clearing the software interrupts.

**Figure 2-100. Hardware Interrupt Signals**

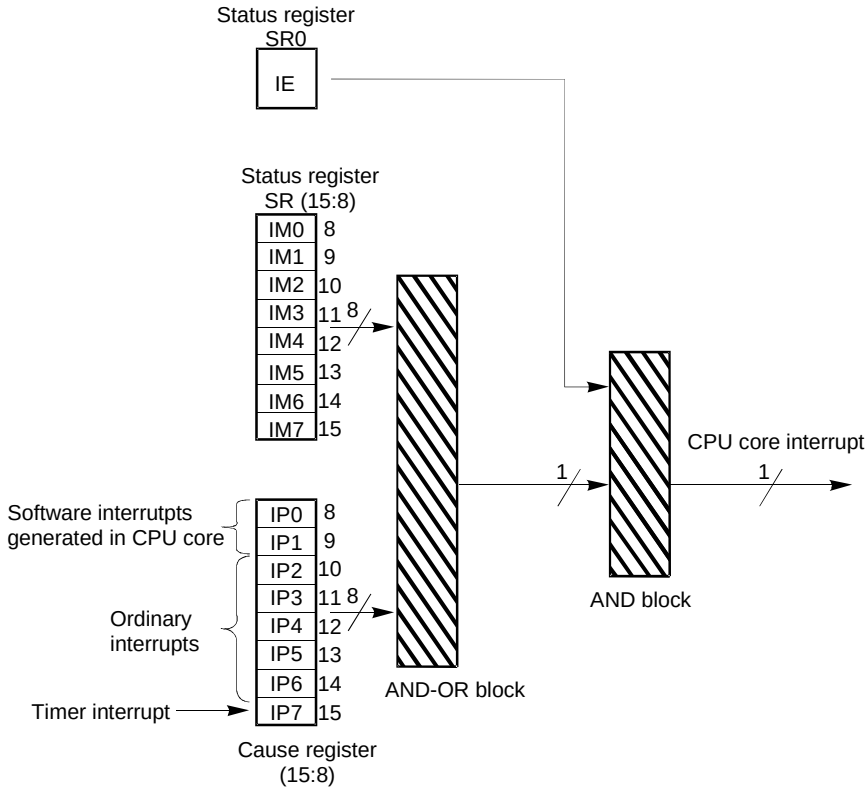


### 2.9.5.2 Masking interrupt signals

Figure 2-101 shows the masking of the CPU core interrupt signals.

- ✧ Cause register bits 15 to 8 (IP7 to IP0) are AND-ORed with Status register interrupt mask bits 15 to 8 (IM7 to IM0) to mask individual interrupts.
- ✧ Status register bit 0 is a global Interrupt Enable (IE). It is ANDed with the output of the AND-OR logic shown in Figure 2-101 to produce the CPU core interrupt signal. The EXL bit in the Status register also enables these interrupts.

**Figure 2-101. Masking of Interrupt Request Signals**




| Bit | Function | Setting |
|---|---|---|
| IE | Whole interrupts enable | 1 : Enable<br>0 : Disable |
| IM(7:0) | Interrupt mask | Each bit 1 : Enable<br>0 : Disable |
| IP(7:0) | Interrupt request | Each bit 1 : Pending<br>0 : Not pending |

**[MEMO]**

# CHAPTER 3 SYSTEM CONTROLLER

## 3.1 Overview

This block is an internal system controller for the μPD98501. System Controller provides bridging function among the CPU system bus “SysAD”, NEC original high speed on-chip bus “IBUS” and memory bus for SDRAM/PROM/FLASH.

Features of System Controller are as follows.

- Provides bus bridging function among SysAD bus, IBUS and MEMORY
- Supports Endian Converting function on SysAD bus
- Can directly connect 16M/64/128Mbit-SDRAM and PROM/FLASH
- Supports Deadman’s SW Timer and separated 2ch Timers
- Supports NS16550 compatible UART

### 3.1.1 CPU interface

- Connects directly to the VR4120A CPU bus “SysAD bus”.
- Supports all VR4120A bus cycles at 66MHz or 100MHz.
- Supports only data rate D.
- Supports only sequential ordering.
- 4word (16byte) x 4entry write command buffer.
- Little-Endian or Big-Endian byte order.

### 3.1.2 Memory interface

- 66MHz or 100MHz memory bus.
- Up to 32MB Base memory range supports SDRAM.
- Up to 8MB write-protectable Boot memory range supports PROM/FLASH.
- On-chip programmable SDRAM refresh controller.
- 4word (16byte) Write data buffer.
- 4word (16byte) Prefetch data buffer (memory-to-CPU).
- PROM/FLASH data signals multiplexed on SDRAM data signals.
- Programmable memory bus arbitration priority.
- Programmable address ranges for the memory.
- Programmable RAS-CAS Delay (2,3,4 clock).
- Programmable CAS Latency (2,3 clock).

### 3.1.3 IBUS interface

- Master and target capability.
- 64word (256byte) IBUS Slave TxFIFO (IBUS read data from MEMORY).
- 64word (256byte) IBUS Slave RxFIFO (IBUS writes data to MEMORY).
- 4word (16byte) IBUS Master TxFIFO (VR4120A read data from IBUS).
- 4word (16byte) IBUS Master RxFIFO (VR4120A writes data to IBUS).
- Supports the bus timer to detect IBUS stall.
- 66MHz IBUS clock rate.

- Supports 266 MB/sec (32bit @66MHz) bursts on IBUS.

### 3.1.4 UART

- Universal Asynchronous Receiver/Transmitter
- Modem control functions
- Even, odd or no parity bit generation
- Fully prioritized interrupt control

### 3.1.5 Timer

- Two 32bit loadable general purpose timers generating Interrupt to CPU

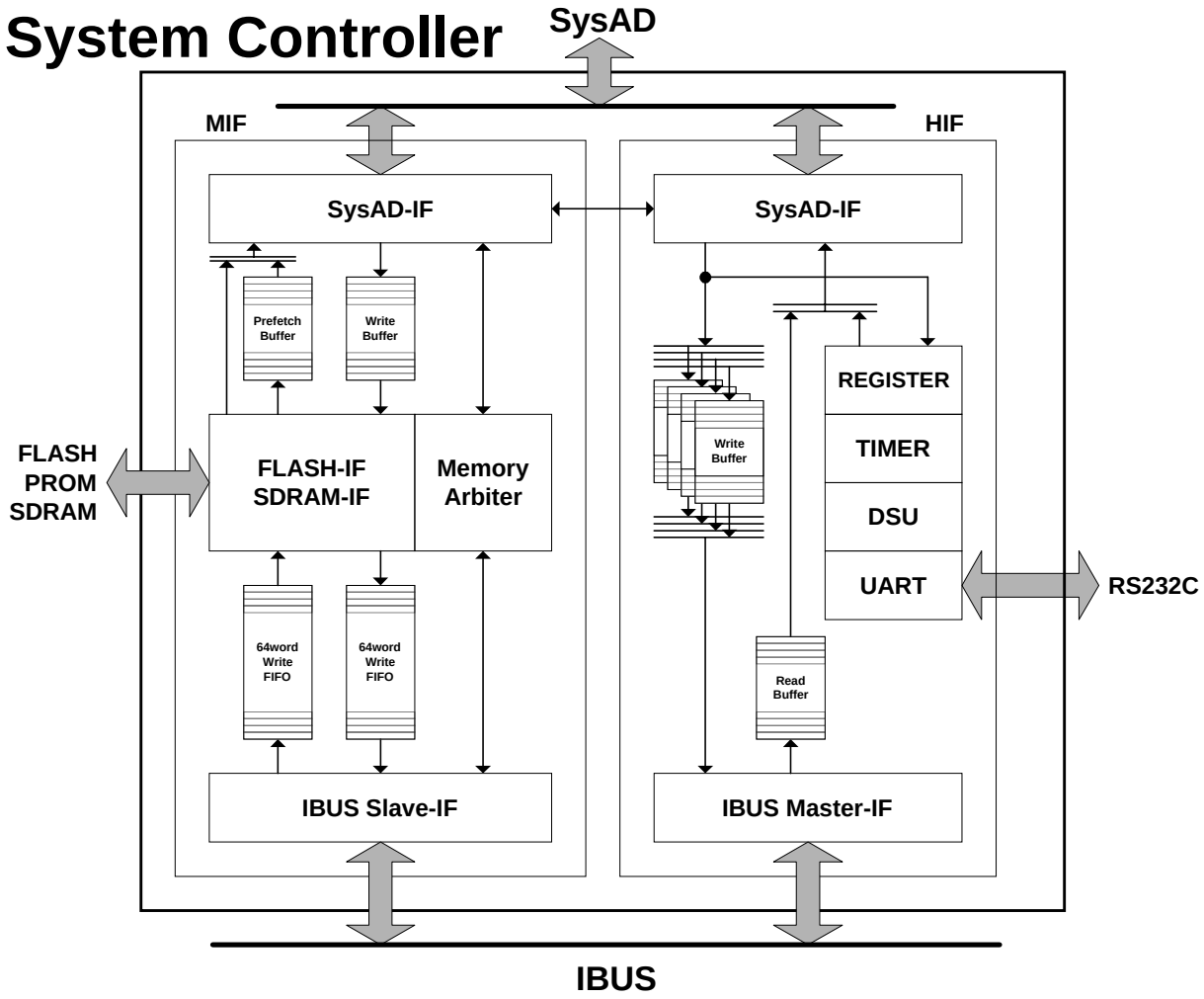
### 3.1.6 Interrupt controller

- Generates NMI and INT
- All interrupt causing events maskable

### 3.1.7 DSU (Deadman's SW UNIT)

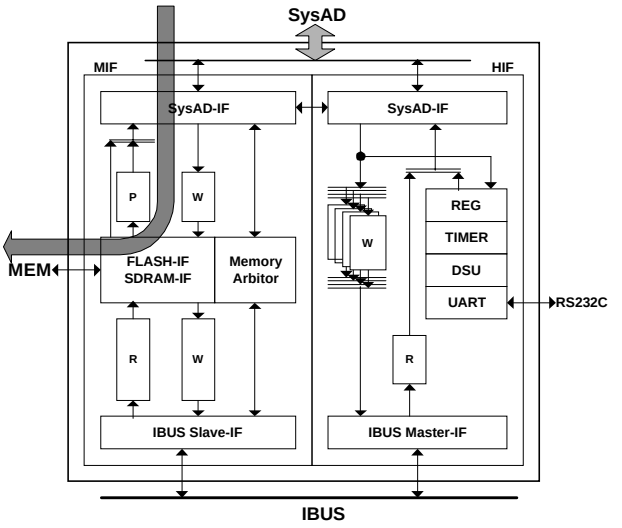
- Deadman's SW UNIT generates cold reset to CPU

### 3.1.8 System block diagram

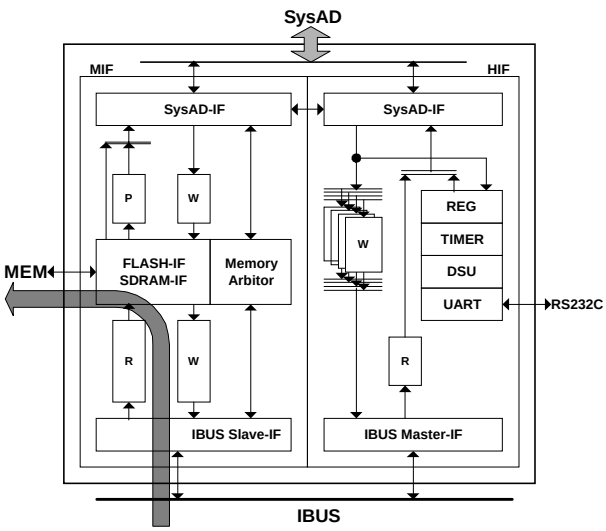


### 3.1.9 Data flow diagram

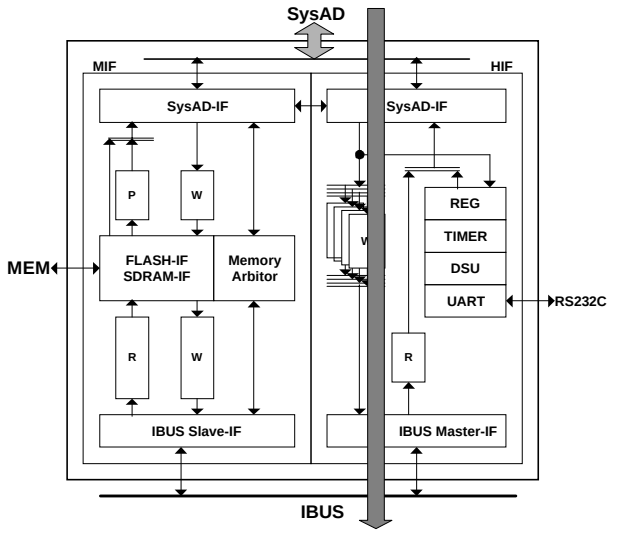
**VR4120A Core to SDRAM**



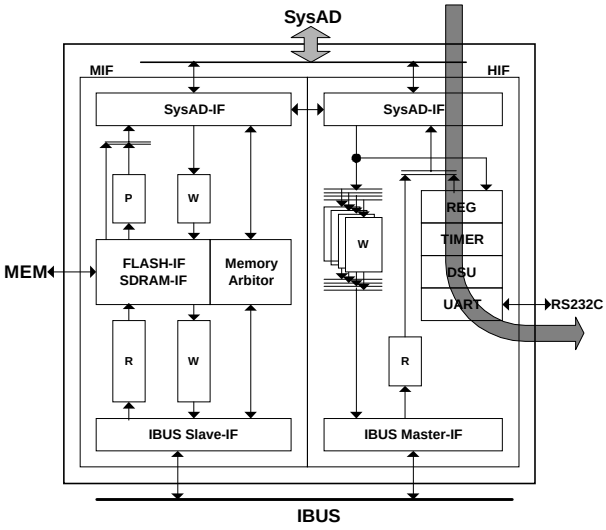

**IBUS to SDRAM**




**VR4120A Core to IBUS**




**VR4120A Core to UART**



## 3.2 Registers

### 3.2.1 Register summary

Following Table summarizes the controller's register set. The base address for the set is 1000_0000H in the physical address space.

| Offset | Name | R/W | Access | Description | Default |
|---|---|---|---|---|---|
| 00H | S_GMR | R/W | W/H/B | General Mode Register | 00000000H |
| 04H | S_GSR | R | W/H/B | General Status Register | unknown |
| 08H | S_ISR | RC | W/H/B | Interrupt Status Register | 00000000H |
| 0CH | S_IMR | R/W | W/H/B | Interrupt Mask Register | 00000000H |
| 10H | S_NSR | RC | W/H/B | NMI Status Register | 00000000H |
| 14H | S_NMR | R/W | W/H/B | NMI Mask Register | 00000000H |
| 18H | S_VER | R | W/H/B | Version Register | 00000100H |
| 1CH | S_IOR | R/W | W/H/B | IO Port Register | 00000000H |
| 20H: 2FH | N/A | ---- | ---- | Reserved | unknown |
| 30H | S_WRCR | W | W/H/B | Warm Reset Control Register | 00000000H |
| 34H | S_WRSR | R | W/H/B | Warm Reset Status Register | 00000000H |
| 38H | S_PWCR | R/W | W/H/B | Power Control Register | 00000000H |
| 3CH | S_PWSR | R | W/H/B | Power Control Status Register | 00000000H |
| 40H: 4BH | N/A | ---- | ---- | Reserved | unknown |
| 4CH | ITCNTR | R/W | W/H/B | IBUS Timeout Timer Control Register | 00000000H |
| 50H | ITSETR | R/W | W/H/B | IBUS Timeout Timer Set Register | 80000000H |
| 54H: 7FH | N/A | ---- | ---- | Reserved | unknown |
| 80H | UARTRBR | R | W/H/B | UART, Receiver Buffer Register [DLAB=0,READ] | unknown |
| 80H | UARTTHR | W | W/H/B | UART, Transmitter Holding Register [DLAB=0,WRITE] | unknown |
| 80H | UARTDLL | R/W | W/H/B | UART, Divisor Latch LSB Register [DLAB=1] | unknown |
| 84H | UARTIER | R/W | W/H/B | UART, Interrupt Enable Register [DLAB=0] | unknown |
| 84H | UARTDLM | R/W | W/H/B | UART, Divisor Latch MSB Register [DLAB=1] | unknown |
| 88H | UARTIIR | R | W/H/B | UART, Interrupt ID Register [READ] | unknown |
| 88H | UARTFCR | W | W/H/B | UART, FIFO control Register [WRITE] | unknown |
| 8CH | UARTLCR | R/W | W/H/B | UART, Line control Register | unknown |
| 90H | UARTMCR | R/W | W/H/B | UART, Modem Control Register | unknown |
| 94H | UARTLSR | R/W | W/H/B | UART, Line status Register | unknown |
| 98H | UARTMSR | R/W | W/H/B | UART, Modem Status Register | unknown |
| 9CH | UARTSCR | R/W | W/H/B | UART, Scratch Register | unknown |
| A0H | DSUCNTR | R/W | W/H/B | DSU Control Register | 00000000H |
| A4H | DSUSETR | R/W | W/H/B | DSU Dead Time Set Register | 80000000H |
| A8H | DSUCLRR | W | W/H/B | DSU Clear Register | 00000000H |
| ACH | DSUTIMR | R | W/H/B | DSU Elapsed Time Register | 00000000H |
| B0H | TMMR | R/W | W/H/B | Timer Mode Register | 00000000H |
| B4H | TM0CSR | R/W | W/H/B | Timer CH0 Count Set Register | 00000000H |
| B8H | TM1CSR | R/W | W/H/B | Timer CH1 Count Set Register | 00000000H |
| BCH | TM0CCR | R | W/H/B | Timer CH0 Current Count Register | FFFFFFFFH |
| C0H | TM1CCR | R | W/H/B | Timer CH1 Current Count Register | FFFFFFFFH |
| C4H: CFH | N/A | ---- | ---- | Reserved | unknown |

| Offset | Name | R/W | Access | Description | Default |
|---|---|---|---|---|---|
| D0H | ECCR | W | W/H/B | EEPROM Command Control Register | 00000000H |
| D4H | ERDR | R | W/H/B | EEPROM Read Data Register | 80000000H |
| D8H | MACAR1 | R | W/H/B | MAC Address Register 1 | 00000000H |
| DCH | MACAR2 | R | W/H/B | MAC Address Register 2 | 00000000H |
| E0H | MACAR3 | R | W/H/B | MAC Address Register 3 | 00000000H |
| E4H: FFH | N/A | ---- | ---- | Reserved | unknown |
| 100H | RMMDR | R/W | W | Boot ROM Mode Register | 00000000H |
| 104H | RMATR | R/W | W | Boot ROM Access Timing Register | 00000000H |
| 108H | SDMDR | R/W | W | SDRAM Mode Register | 00000330H |
| 10CH | SDTSR | R/W | W | SDRAM Type Selection Register | 00000000H |
| 110H | SDPTR | R/W | W | SDRAM Precharge Timing Register | 00000142H |
| 114: 11BH | N/A | ---- | ---- | Reserved | unknown |
| 11CH | SDRMR | R/W | W | SDRAM Refresh Mode Register | 00000200H |
| 120H | SDRCR | R | W | SDRAM Refresh Timer Count Register | 00000200H |
| 124H | MBCR | R/W | W | Memory Bus Control Register | 00000000H |
| 128H | MESR | RC | W | Memory Error Status Register | 00000000H |
| 12CH | MEAR | RC | W | Memory Error Address Register | 00000000H |
| 130H: FFFH | N/A | ---- | ---- | Reserved | unknown |

**Remarks 1.** In the “R/W” field,
“W” means “writeable”
“R” means “readable”
“RC” means “read-cleared”
“----“ means “not accessible”

**2.** All internal registers are 32bit word aligned register.

**3.** The burst access to the internal register is prohibited.
If such burst access has been occurred, IRERR bit in NSR is set and NMI will assert to CPU.

**4.** Read access to the reserved area will set the CBERR bit in the NSR register, and the dummy read response data with the data-error bit set on SysCMD[0] is returned.

**5.** Write access to the reserved area will set the CBERR bit in the NSR register, and the write data is lost.

**6.** In the “Access” filed,
“W” means that Word access is valid.
“H” means that Half word access is valid.
“B” means that Byte access is valid.

**7.** Write access to the read-only register cause no error, but the write data is lost.

**8.** The CPU can access all internal register, but IBUS Master device cannot access them.

### 3.2.2 General registers

#### 3.2.2.1 General Mode Register (S_GMR)

The General Mode Register “S_GMR” is read-write and word aligned 32bit register. After initializing, VR4120A set the IAEN bit to enable the IBUS arbiter. S_GMR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 0 | CRST | Cold Reset:<br>0 = do nothing<br>1 = perform cold reset (same as hardware system reset) |
| 1 | IAEN | IBUS Arbiter Enable:<br>0 = disable (IBUS Arbiter does not allow the grant except System Controller)<br>1 = enable. |
| 2 | MPFD | Memory-to-CPU Prefetch FIFO disable:<br>0 = enable<br>1 = disable |
| 3 | UCSEL | UART Source Clock Selection:<br>0 = use 1/2 of CPU clock<br>1 = use external clock (18.432MHz) |
| 7:4 | Reserved | Hardwired to 0. |
| 8 | HSWP | HIF Block Data swap function enable:<br>0 = enable<br>1 = disable |
| 9 | MSWP | MIF Block Data swap function enable:<br>0 = enable<br>1 = disable |
| 31:10 | Reserved | Hardwired to 0. |

#### 3.2.2.2 General Status Register (S_GSR)

The General Status Register “S_GSR” is read-only and word aligned 32bit register. S_GSR shows the status of external pin of the μPD98501. S_GSR contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 0 | ENDCEN | This field reflects the status of external pin “ENDCEN” after reset.<br>0 = connected to GND, It means that Endian Converter is disabled.<br>1 = connected to VCC, It means that Endian Converter is enabled. |
| 1 | CCLKSEL | This field reflects the status of external pin “CCLKSEL” after reset.<br>0 = connected to GND, It means that CPU is operated at 100MHz.<br>1 = connected to VCC, It means that CPU is operated at 66MHz. |
| 31:2 | Reserved | Hardwired to 0. |

### 3.2.2.3 Interrupt Status Register (S_ISR)

The Interrupt Status Register “S_ISR” is read-clear and word aligned 32bit register. “S_ISR” shows the interruption status from SysAD / IBUS interfaces, TIMER, UART and so on. If corresponding bit in S_IMR(Interrupt Mask Register) is reset and the interrupt is not masked, System Controller interrupt to VR4120A using interrupt signal. The bit in S_ISR is reset after being read by the VR4120A. When the same type of incidents occurs before the bit has been read, the bit will be set again. S_ISR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 0 | TM0IS | TIMER CH0 interrupt.<br>1 = TIMER CH0 interrupt pending<br>0 = No TIMER CH0 interrupt pending |
| 1 | TM1IS | TIMER CH1 interrupt.<br>1 = TIMER CH1 interrupt pending<br>0 = No TIMER CH1 interrupt pending |
| 2 | UARTIS | UART interrupt.<br>1 = UART interrupt pending.<br>0 = No UART interrupt pending.<br>UART interruption are one of the following interruption:<br>1. UART Receive data Buffer Full Interrupt<br>2. UART Transmitter Buffer empty Interrupt<br>3. UART Line status Interrupts<br>4. UART Modem status Interrupts |
| 3 | EXTIS | External Interrupt.<br>1 = External Interrupt pending.<br>0 = No External interrupt pending. |
| 4 | WUIS | Wakeup Interrupt.<br>1 = Any wakeup requests pending.<br>0 = No wakeup requests pending. |
| 31:5 | Reserved | Hardwired to 0. |

**Remark** To clear this register, the CPU must read the byte contained the TMS0IS field.

#### 3.2.2.4 Interrupt Mask Register (S_IMR)

The Interrupt Mask Register “S_IMR” is read-write and word aligned 32bit register. S_IMR masks interruption for each corresponding incident. A Mask bit, which locates in the same bit location to a corresponding bit in S_ISR, controls interruption triggered by the incident. If a bit of this register is reset to zero, the corresponding bit of the S_ISR is masked. If it is set to 1, the corresponding bit is unmasked. When the mask bit is reset and the bit in S_ISR is set, System Controller assert interruption signal to interrupt VR4120A. It is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 0 | TM0IM | TIMER CH0 interrupt mask.<br>1 = unmask.<br>0 = mask. |
| 1 | TM1IM | TIMER CH1 interrupt mask.<br>1 = unmask.<br>0 = mask. |
| 2 | UARTIM | UART interrupt mask.<br>1 = unmask.<br>0 = mask. |
| 3 | EXTIM | External interrupt mask<br>1 = unmask.<br>0 = mask. |
| 4 | WUIM | Wakeup interrupt mask<br>1 = unmask.<br>0 = mask. |
| 31:5 | Reserved | Hardwired to 0. |

#### 3.2.2.5 NMI Status Register (S_NSR)

The Interrupt Status Register “S_NSR” is read-clear and word aligned 32bit register. “S_NSR” shows the non-maskable interruption “NMI” status from SysAD / IBUS interfaces, External NMI, Memory Interface and so on. If corresponding bit in S_NMR (NMI Mask Register) is reset and the NIM is disabled, System Controller interrupt to VR4120A using non-maskable interrupt signal. The bit in S_NSR is reset after being read by the VR4120A. When the same type of incidents occurs before the bit has been read, the bit will be set again. S_NSR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 0 | CBERR | CPU Bus Error.<br>CPU Bus error includes the illegal bus command, illegal data align, illegal bust size, and illegal access to the RFU area in register space.<br>1 = CPU bus error.<br>0 = No such error. |
| 1 | IBERR | IBUS Bus Error.<br>IBUS Bus error is occurred when CPU access to the RFU area in the IBUS Target Address Space (see MEMORY MAP section).<br>1 = A bus error occurred during the IBUS master access.<br>0 = No such error. |
| 2 | ITERR | IBUS Timeout Error.<br>IBUS Timeout error is occurred when the IBUS is stalled.<br>1 = IBUS timeout error.<br>0 = No such error. |
| 3 | MAERR | Memory Address Error.<br>Memory Address error includes the memory access to the illegal memory space (RFU space and out range of the SDRAM/ROM space) and the illegal memory access (byte or half-word ROM access or burst write access to the ROM).<br>1 = An address range error occurred during the memory access.<br>0 = No such error. |
| 4 | EXTNMI | External NMI.<br>1 = External NMI is asserted<br>0 = External NMI is not asserted. |
| 5 | IRERR | Illegal Internal Register Access Error.<br>1 = illegal internal Register access, ex burst access, has been performed.<br>0 = No such access |
| 31:6 | Reserved | Hardwired to 0. |

**Remark** To clear this register, the CPU must read the byte contained the CBERR field.

### 3.2.2.6 NMI Mask Register (S_NMR)

The NMI Mask Register “S_NMR” is read-write and word aligned 32bit register. S_NMR enables NMI for each corresponding incident. A Enable bit, which locates in the same bit location to a corresponding bit in S_NSR, controls interruption triggered by the incident. If a bit of this register is reset to zero, the corresponding bit of the S_NSR is disabled. If it is set to 1, the corresponding bit is enabled. When the enable bit is reset and the bit in S_NSR is set, System Controller assert interruption signal to interrupt VR4120A. S_NMR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 0 | CBERRE | CPU Bus Error enable.<br>1 = enable.<br>0 = disable. |
| 1 | IBERRE | IBUS Bus Error.<br>1 = enable.<br>0 = disable. |
| 2 | ITERRE | IBUS Timeout Error.<br>1 = enable<br>0 = disable |
| 3 | MAERRE | Memory Address Error.<br>1 = enable<br>0 = disable |
| 4 | EXTNMIE | External NMI.<br>1 = enable<br>0 = disable |
| 5 | IRERRE | Illegal Internal Register Access Error.<br>1 = enable<br>0 = disable |
| 31:6 | Reserved | Hardwired to 0. |

### 3.2.2.7 Version register (S_VER)

The Version Register “S_VER” is read-only and word aligned 32bit register. Version Register shows version number of the μPD98501. S_VER is initialized to 100H at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 7:0 | MINOR | Minor revision. Hardwired to 00H. |
| 15:8 | MAJOR | Major revision. Hardwired to 01H. |
| 31:16 | Reserved | Hardwired to 0. |

### 3.2.2.8 IO Port Register (S_IOR)

The IO Port Register “S_IOR” is read-write and word aligned 32bit register. IO Port Register is used to indicate the status of Software. Each bit of the following LED field are connected to the External IO Port directly. S_IOR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 0 | LED0 | LED0 enable.<br>0 = deassert External LED Port0.<br>1 = assert External LED Port0 |
| 1 | LED1 | LED1 enable.<br>0 = deassert External LED Port1.<br>1 = assert External LED Port1 |
| 2 | LED2 | LED2 enable.<br>0 = deassert External LED Port2.<br>1 = assert External LED Port2 |
| 3 | LED3 | LED3 enable.<br>0 = deassert External LED Port3.<br>1 = assert External LED Port3 |
| 4 | LED4 | LED4 enable.<br>0 = deassert External LED Port4.<br>1 = assert External LED Port4 |
| 5 | LED5 | LED5 enable.<br>0 = deassert External LED Port5.<br>1 = assert External LED Port5 |
| 6 | LED6 | LED6 enable.<br>0 = deassert External LED Port6.<br>1 = assert External LED Port6 |
| 7 | LED7 | LED7 enable.<br>0 = deassert External LED Port7.<br>1 = assert External LED Port7 |

### 3.2.2.9 Warm Reset Control Register (S_WRCR)

The Warm Reset Control Register “S_WRCR” is read-write and word aligned 32bit register. S_WRCR generates warm-reset request to USB Controller, Ethernet Controller, and ATM Cell Processor independently. S_WRCR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 0 | USBWR | Warm Reset request for USB Controller:<br>0 = do nothing.<br>1 = perform warm reset. |
| 1 | MACWR | Warm Reset request for Ethernet Controller #1:<br>0 = do nothing.<br>1 = perform warm reset. |
| 2 | ATMWR | Warm Reset request for ATM Cell Processor:<br>0 = do nothing.<br>1 = perform warm reset. |
| 3 | MAC2WR | Warm Reset request for Ethernet Controller #2:<br>0 = do nothing.<br>1 = perform warm reset. |
| 4 | UARTWR | Warm Reset request for UART:<br>0 = do nothing.<br>1 = perform warm reset. |
| 31:5 | Reserved | Hardwired to 0. |

**Remark** All field in this register will read back as zero

### 3.2.2.10 Warm Reset Status Register (S_WRSR)

The Warm Reset Status Register “S_WRSR” is read-only and word aligned 32bit register. S_WRSR indicates the response from USB Controller, Ethernet Controller, and ATM Cell Processor independently. S_WRCR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 0 | USBWRST | shows Warm Reset Status from USB Controller:<br>0 = USB Controller is busy to perform the warm reset..<br>1 = warm reset has been done. USB Controller is ready. . |
| 1 | MACWRST | shows Warm Reset Status from Ethernet Controller #1:<br>0 = Ethernet Controller #1 is busy to perform the warm reset..<br>1 = warm reset has been done. MAC Ethernet Controller #1 is ready. |
| 2 | ATMWRST | shows Warm Reset Status from ATM Cell Processor:<br>0 = ATM Cell Processor is busy to perform the warm reset.<br>1 = warm reset has been done. ATM Cell Processor is ready. |
| 3 | MAC2WRST | shows Warm Reset Status from Ethernet Controller #2:<br>0 = Ethernet Controller #2 is busy to perform the warm reset.<br>1 = warm reset has been done. MAC Ethernet Controller #2 is ready. |
| 4 | UARTWRST | shows Warm Reset Status from UART:<br>0 = UART is busy to perform the warm reset.<br>1 = warm reset has been done. UART is ready. |
| 31:5 | Reserved | Hardwired to 0. |

### 3.2.2.11 Power Control Register (S_PWCR)

The Power Control Register “S_PWCR” is read-write and word aligned 32bit register. S_PWCR requests to keep the IDLE State for USB Controller, Ethernet Controller, and ATM Cell Processor. CPU must request these interface block to keep the IDLE State and check their acknowledgement using read the Power Status Register “S_PWSR” prior to perform SUSPEND by setting following xxxSTOP field. S_WRCR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 0 | USBIDRQ | IDLE request for USB Controller:<br>0 = do nothing.<br>1 = request to keep the IDLE State. |
| 1 | MACIDRQ | IDLE request for Ethernet Controller #1:<br>0 = do nothing.<br>1 = request to keep the IDLE State. |
| 2 | ATMIDRQ | IDLE request for ATM Cell Processor:<br>0 = do nothing.<br>1 = request to keep the IDEL State. |
| 3 | MAC2IDRQ | IDLE request for Ethernet Controller #2:<br>0 = do nothing.<br>1 = request to keep the IDEL State. |
| 15:4 | Reserved | Hardwired to 0. |
| 16 | USBSTOP | SUSPEND request for USB Controller:<br>0 = enable system clock for USB Controller.<br>1 = disable system clock for USB Controller. |
| 17 | MACSTOP | SUSPEND request for Ethernet Controller #1:<br>0 = enable system clock for Ethernet Controller #1.<br>1 = disable system clock for Ethernet Controller #1. |
| 18 | ATMSTOP | SUSPEND request for ATM Cell Processor:<br>0 = enable system clock for ATM Cell Processor.<br>1 = disable system clock for ATM Cell Processor. |
| 19 | MAC2STOP | SUSPEND request for Ethernet Controller #2:<br>0 = enable system clock for Ethernet Controller #2.<br>1 = disable system clock for Ethernet Controller #2. |
| 31:20 | Reserved | Hardwired to 0. |

**Remark** Before accesses to this register, the CPU must flush the internal write command buffer by reading the IBUS target

### 3.2.2.12 Power Status Register (S_PWSR)

The Power Status Register “S_PWSR” is read-only and word aligned 32bit register. The IDLE filed in S_PWSR indicates the status that it is ready to SUSPEND. The WKUP filed in S_PWSR indicates the wakeup request. When IDLE field becomes one, VR4120A RISC Processor can disable the system clock for the corresponding device by setting the STOP field in S_PWCR. When WKUP field in S_PWSR becomes one, VR4120A must enable the system clock for the corresponding device by resetting the STOP field in S_PWCR. The S_WRCR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 0 | USBIDLE | This field indicate the IDLE status in USB Controller<br>0 = Not in IDLE state. It means USB Controller is NOT ready to SUSPEND.<br>1 = in IDLE state. It means USB Controller is ready to SUSPEND. |
| 1 | MACIDLE | This field indicate the IDLE status in Ethernet Controller #1<br>0 = Not in IDLE state. It means Ethernet Controller #1 is NOT ready to SUSPEND.<br>1 = in IDLE state. It means Ethernet Controller #1 is ready to SUSPEND. |
| 2 | ATMIDLE | This field indicates the IDLE status in ATM Cell Processor.<br>0 = Not in IDLE state. It means ATM Cell Processor is NOT ready to SUSPEND.<br>1 = in IDLE state. It means ATM Cell Processor is ready to SUSPEND. |
| 3 | MAC2IDLE | This field indicates the IDEL status in Ethernet Controller #2.<br>0 = Not in IDLE state. It means Ethernet Controller #2 is NOT ready to SUSPEND.<br>1 = in IDLE state. It means Ethernet Controller #2 is ready to SUSPEND. |
| 7:4 | Reserved | Hardwired to 0. |
| 8 | USBWKUP | This field indicate the wakeup request from USB Controller<br>0 = No wakeup request is pending.<br>1 = wakeup request is pending. |
| 9 | MACWKUP | This field indicate the wakeup request form Ethernet Controller #1<br>0 = No wakeup request is pending.<br>1 = wakeup request is pending. |
| 10 | ATMWKUP | This field indicates the wakeup request from ATM Cell Processor.<br>0 = No wakeup request is pending.<br>1 = wakeup request is pending. |
| 11 | MAC2WKUP | This field indicates the wakeup request from Ethernet Controller #2.<br>0 = No wakeup request is pending.<br>1 = wakeup request is pending. |
| 31:12 | Reserved | Hardwired to 0. |

## 3.3 CPU Interface

The System Controller provides the direct interface for the VR4120A using the 32bit SysAD Bus operated at 100MHz or 66MHz.

### 3.3.1 Overview

- Connects directly to the VR4120A CPU bus “SysAD bus”.
- Supports all VR4120A bus cycles at 66MHz or 100MHz.
- Supports only data rate D.
- Supports only sequential ordering.
- 4word (16byte) x 4entry Write command buffer.
- Little-Endian or Big-Endian byte order.

### 3.3.2 Data rate control

The CPU-to-System Controller data rate is programmable in the EP field (bits 27:24) of the CPU’s Configuration Register. The controller supports only data rate D.

### 3.3.3 Address decoding

The controller latches the address on the SysAD bus. It then decodes the address and SysCmd signals to determine the transaction type. Ten address ranges can be decoded:

- One range for External Boot PROM or FLASH.
- One range for External SDRAM.
- One range for System Controller’s internal configuration registers.

Boot PROM/FLASH is mapped according to its size. System Controller’s internal registers are fixed at base address 1000_0000H, to allow the VR4120A to access them during boot, before they have been configured. All other decode ranges are programmable.

### 3.3.4 Endian conversion

The BE bit in the CPU’s Configuration Register specifies the CPU’s byte ordering at reset. BE=0 configures little-endian order, BE=1 configures big-endian order. CPU interface of the system controller supports either big- or little-endian byte ordering on the SysAd bus by using Endian Converter. All of the other interfaces in the System Controller operate only in little-endian mode.

When the CPU is operated in the big-endian mode(external BIG-pin is HIGH), the System Controller provides the two endian conversion method controlled by external ENDCEN-pin. If ENDCEN-pin is LOW, the System Controller performs the data swap on the SysAD Bus (see Endian Translation Table for data swap mode). If ENDCEN-pin is HIGH, The System Controller performs the address swap on the SysAD Bus(the detail is described in the Endian Transfer Table for the address swap mode).

**Table 3-1. Endian Configuration Table**

| BIG-pin | ENDCEN-pin | Status Reg RE field in CPU | Endian in CPU | Endian in System Controller | Endian Converter Operation |
|---|---|---|---|---|---|
| 0 | 0 | 0 | LITTLE | LITTLE | Transparent |
| 0 | 1 | 0 | LITTLE | LITTLE | Transparent |
| 1 | 0 | 0 | BIG | LITTLE | Data swap mode |
| 1 | 1 | 0 | BIG | LITTLE | Address swap mode |

**Remark** VR4120A CORE does NOT support reverse endian mode.

**Table 3-2. Endian Translation Table in Endian Converter**

| CPU Access Type | | BIG | ENDCEN | Before Translation SysAD[1:0] | After Translation SysAD[1:0] | NOTE |
|---|---|---|---|---|---|---|
| Block | 2word 4word | 1 | 1 | 00 | 00 | valid |
| | | | | 01 | 01 | invalid |
| | | | | 10 | 10 | invalid |
| | | | | 11 | 11 | invalid |
| Single | 1byte | 1 | 1 | 00 | 11 | valid |
| | | | | 01 | 10 | valid |
| | | | | 10 | 01 | valid |
| | | | | 11 | 00 | valid |
| Single | 2byte | 1 | 1 | 00 | 10 | valid |
| | | | | 01 | 11 | invalid |
| | | | | 10 | 00 | valid |
| | | | | 11 | 01 | invalid |
| Single | 3byte | 1 | 1 | 00 | 01 | valid |
| | | | | 01 | 00 | valid |
| | | | | 10 | 11 | invalid |
| | | | | 11 | 10 | invalid |
| Single | 4byte | 1 | 1 | 00 | 00 | valid |
| | | | | 01 | 01 | invalid |
| | | | | 10 | 10 | invalid |
| | | | | 11 | 11 | invalid |

| CPU Access Type | BIG | ENDCEN | Before Translation SysAD[1:0] | After Translation SysAD[1:0] | NOTE |
|---|---|---|---|---|---|
| Any | 1 | 0 | [31:24][23:16][15:8][7:0] | [7:0][15:8][23:16][31:24] | Byte swap |

### 3.3.5 I/O performance

The following table shows the I/O performance accessing from the VR4120A via Host Interface.

| W/R | Target Area | Burst Length | Access Latency [CPU clocks] |
|---|---|---|---|
| R | IBUS Target | 1 | 24 |
| R | IBUS Target | 2 | 24-1 |
| R | IBUS Target | 4 | 27-1-1-1 |
| R | Internal Register (except UART) | 1 | 7 |
| R | Internal Register (except UART) | 2 | INVALID |
| R | Internal Register (except UART) | 4 | INVALID |
| R | Internal UART Register | 1 | 14 (depend UART source clock) |
| R | Internal UART Register | 2 | INVALID |
| R | Internal UART Register | 4 | INVALID |
| W | IBUS Target | 1 | 23 |
| W | IBUS Target | 2 | 23 |
| W | IBUS Target | 4 | 23-2-1-2 |
| W | Internal Write Command FIFO | 1 | 6-1(WAIT) |
| W | Internal Write Command FIFO | 2 | 6-1 |
| W | Internal Write Command FIFO | 4 | 6-1-1-1 |
| W | Internal Register (except UART) | 1 | 7 |
| W | Internal Register (except UART) | 2 | INVALID |
| W | Internal Register (except UART) | 4 | INVALID |
| W | Internal UART Register | 1 | 15 (depend UART source clock) |
| W | Internal UART Register | 2 | INVALID |
| W | Internal UART Register | 4 | INVALID |

**Remarks 1.** BUS frequency : SysAD=100MHz , IBUS=66MHz

**2.** The latency value accessing to the IBUS Target does NOT include IBUS bus arbitration cycle (about 6 CPU clocks)

## 3.4 Memory Interface

The CPU accesses memory attached to the controller in the normal way, by addressing the memory space.

### 3.4.1 Overview

- 66MHz or 100MHz memory bus.
- Up to 32MB Base memory range supports SDRAM.
- Up to 8MB write-protectable Boot memory range supports PROM/FLASH.
- On-chip programmable SDRAM refresh controller.
- 4word (16byte) Prefetch data buffer (Memory-to-CPU).
- PROM/FLASH data signals multiplexed on SDRAM data signals.
- Programmable memory bus arbitration priority.
- Programmable address ranges for the memory.
- Programmable RAS-CAS delay (2,3,4 clock).
- Programmable CAS latency (2,3 clock).

### 3.4.2 Memory regions and devices

The controller connects directly to memory and manages the addresses, data and control signals for the following address ranges:

- One Boot PROM/FLASH range (programmable)
- One System Memory range (programmable)

The following types of memory modules as an example but not limited to, can be used:

- FLASH can be used in the Boot ROM.
- PROM can be used in the Boot ROM.
- 16Mbit/64Mbit/128Mbit SDRAM can be used in the system memory.

Boot ROM can be populated with PROM or 85-ns FLASH chips. Prior to accessing PROM/FLASH, software must configure this address range. The system memory can be populated with 16Mbit/64Mbit/128Mbit SDRAM chips. The system memory is used for the RTOS, M/W and F/W. Prior to accessing SDRAM, software must configure this address range.

Recommended SDRAM Devices are listed following:

- 16Mbit SDRAM (NEC part numbers μPD4516161).
- 64Mbit SDRAM (NEC part numbers μPD4564323, μPD4564163).
- 128Mbit SDRAM (NEC part numbers μPD45128163).

Recommended FLASH Devices are listed following:

- 4Mbit FLASH (NEC part numbers μPD29F800AL).
- 16Mbit FLASH (NEC part numbers μPD29F1600AL).

### 3.4.3 Memory signal connections

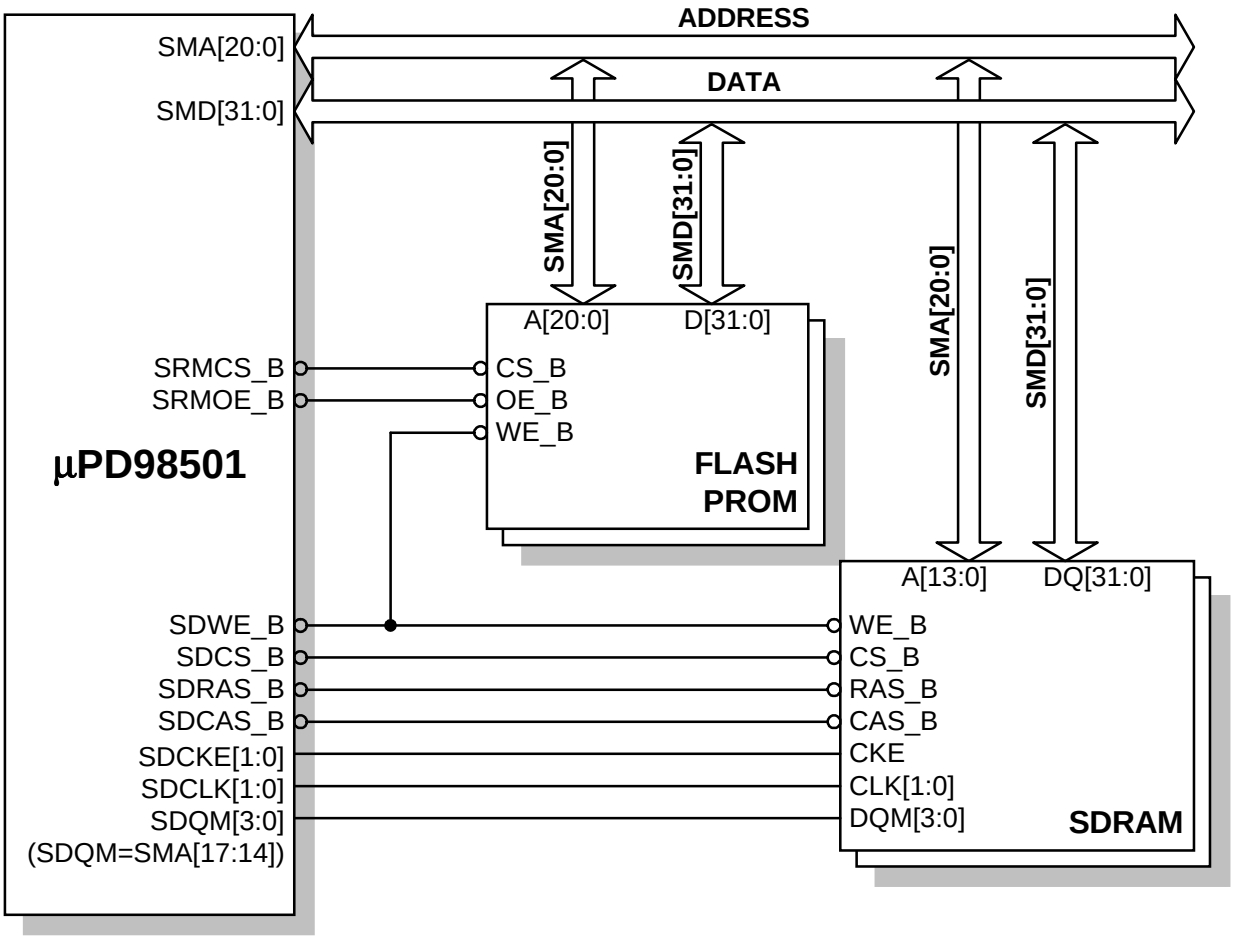



**Table 3-3. External Pin Mapping**

| External Pin | | Access to ROM | Access to SDRAM |
|---|---|---|---|
| Name | Bits | | |
| SMA | [13:0] | A[13:0] | A[13:0] |
| | [17:14] | A[17:14] | SDQM[3:0] |
| | [20:18] | A[20:18] | |
| SMD | [31:0] | D[31:0] | DQ[31:0] |
| SDCS_B | | --- | SDCS_B |
| SDRAS_B | | --- | SDRAS_B |
| SDCAS_B | | --- | SDCAS_B |
| SDWE_B | | SDWE_B | SDWE_B |
| SDCKE | [1:0] | --- | SDCKE[1:0] |
| SDCLK | [1:0] | --- | SDCLK[1:0] |
| SRMCS_B | | SRMCS_B | --- |
| SRMOE_B | | SRMOE_B | --- |

### 3.4.4 Memory performance

The latency of memory accesses is determined by memory type, speed and prefetch scheme. Following lists some examples of the number of 66MHz or 100MHz memory-bus clocks required for each transfer of an 4-word (16-byte) CPU instruction-cache line fill. The first number in the “SysAD CPU Clocks” column is for the first word; the remaining numbers for the subsequent words. Only the most common combinations are shown.

**Table 3-4. Examples of Memory Performance (4word-burst access from CPU)**

| Memory Type | Bank-Interleaved | Page Hit | R/W | Prefetch Hit | Access Latency view from CPU [SysAD clocks] |
|---|---|---|---|---|---|
| SDRAM,10ns | No | Yes | R | Yes | 6-1-1-1 |
| SDRAM,10ns | No | Yes | R | No | 14-1-1-1 |
| SDRAM,10ns | No | Yes | W | N/A | 9-1-1-1 |
| Flash, 85ns | No | No | R | N/A | 19-12-1-12 |
| Flash, 85ns | No | No | W | N/A | 18 (Single Access Only)) |
| PROM | No | No | R | N/A | 19-12-12-12 |

**Remarks 1.** SDRAM Configuration : RCD=3, CL=2, SDCLK=100MHz, FAT=10

**2.** BUS frequency : SysAD=100MHz , IBUS=66MHz

**3.** Read performance is calculated by counting the rising edge for CPU clock where the read command is issued by the CPU. Because the CPU issues write data with no wait-states once the write command is issued, the numbers in the table represent the rate at which data is written to memory. The sum of the numbers represents the number of cycles between when the write operation was issued and when the next CPU memory operation can begin.

**4.** The burst write access to the FLASH/ROM is invalid. The CPU can access to the FLASH / ROM using single access only

**Table 3-5. Examples of Memory Performance (4word-burst access from IBUS Master)**

| Memory Type | Bank-Interleaved | Page Hit | R/W | Prefetch Hit | Access Latency view from IBUS {IBUS clocks} |
|---|---|---|---|---|---|
| SDRAM,10ns | No | Yes | R | N/A | 18-1-1-1 |
| SDRAM,10ns | No | Yes | W | N/A | 12-1-1-1 |
| Flash, 85ns | No | No | R | N/A | 45-1-1-1 |
| Flash, 85ns | No | No | W | N/A | INVALID |
| PROM | No | No | R | N/A | 45-1-1-1 |

**Remarks 1.** SDRAM Configuration : RCD=3, CL=2, SDCLK=100MHz, FAT=10

**2.** BUS frequency : SysAD=100MHz , IBUS=66MHz

**3.** Above access latency doses Not include the IBUS arbitration cycle (4 IBUS clocks)

**4.** Any write access to the FLASH/ROM is prohibited. If the IBUS master perform the write access to the FLASH/ROM, The IBUS bus error will be occurred.

### 3.4.5 Memory control registers

#### 3.4.5.1 ROM Mode Register (RMMDR)

The ROM Mode Register “RMMDR” is read-write and word aligned 32bit register. RMMDR is used to setup the PROM/FLASH memory interface. RMMDR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 1:0 | FSM | FLASH/PROM Size Model:<br>00 = mode1 (4MByte mode)<br>01 = mode2 (8MByte mode)<br>10 = reserved<br>11 = reserved |
| 7:2 | Reserved | Hardwired to 0. |
| 8 | WM | Write Mask:<br>0 = Masked. FLASH data is protected from unintentional write.<br>1 = Not Masked. FLASH data is not protected. |
| 31:9 | Reserved | Hardwired to 0. |

**Remarks 1.** DON’T change the value on the FSM field after setting the value 01 into the FSM field

**2.** DON’T set the reserved value to each field in this register.

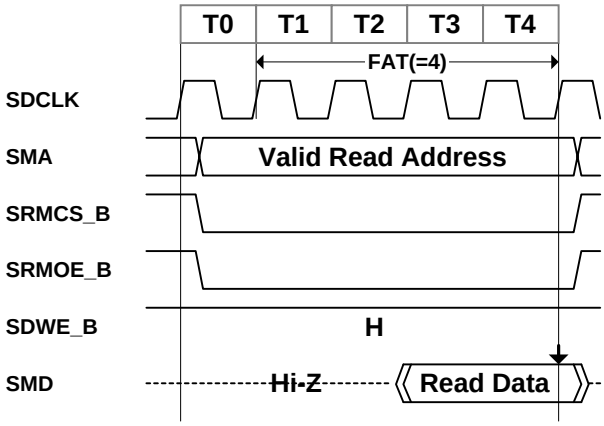
#### 3.4.5.2 ROM Access Timing Register (RMATR)

The ROM Access Timing Register “RMATR” is read-write and word aligned 32bit register. RMATR is used to set the access time in the PROM/FLASH interface. RMATR is initialized to 0 at reset and contains the following fields:

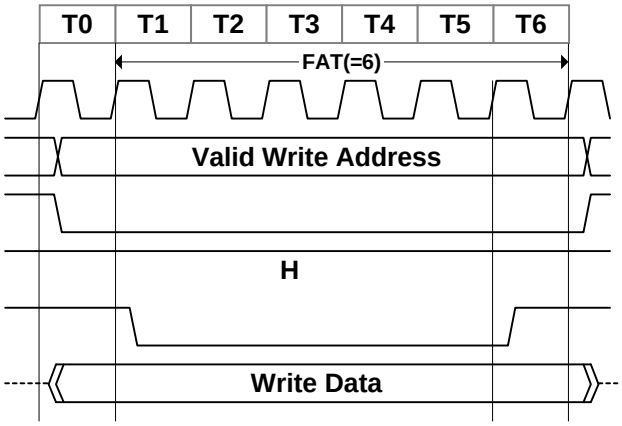
| Bits | Field | Description | | |
|---|---|---|---|---|
| 2:0 | FAT | FLASH/PROM Access Timing for Normal ROM: | | |
| | | 000 = 18clock | 66MHz:272.4ns | 100MHz:180ns |
| | | 001 = 4clock | 66MHz: 60.6ns | 100MHz: 40ns |
| | | 010 = 6clock | 66MHz: 90.9ns | 100MHz: 60ns |
| | | 011 = 8clock | 66MHz:121.2ns | 100MHz: 80ns |
| | | 100 = 10clock | 66MHz:151.5ns | 100MHz:100ns |
| | | 101 = 12clock | 66MHz:181.8ns | 100MHz:120ns |
| | | 110 = 14clock | 66MHz:212.1ns | 100MHz:140ns |
| | | 111 = 16clock | 66MHz:242.4ns | 100MHz:160ns |
| 31:3 | Reserved | Hardwired to 0. | | |

**Remark** ROM Access Timing is depended on the system clock frequency.

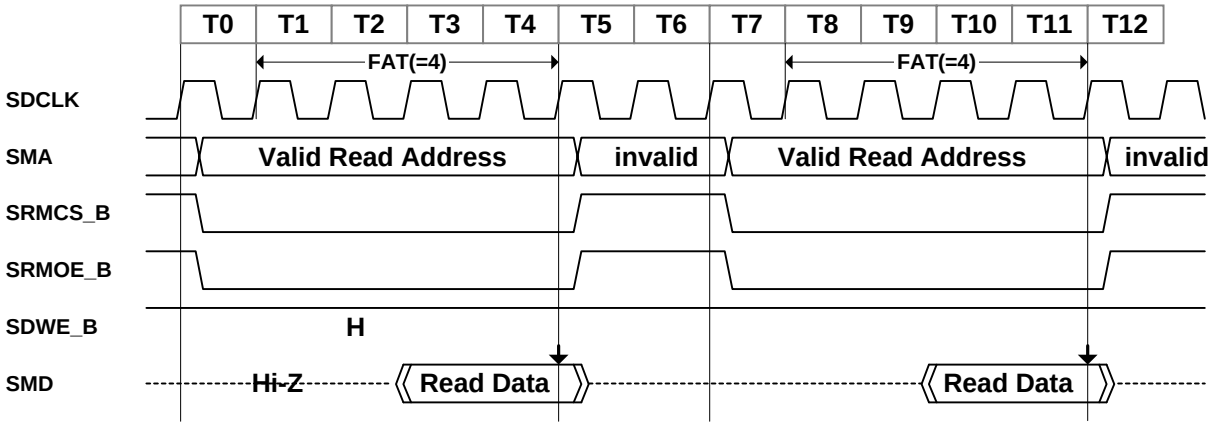
## Normal ROM Read Cycle




## FLASH Memory Write Cycle




## ROM Burst Read Cycle



### 3.4.5.3 SDRAM Mode Register (SDMDR)

The SDRAM Mode Register “SDMDR” is read-write and word aligned 32bit register. SDMDR is used to setup the SDRAM interface. SDMDR is initialized to 330H at reset and contains the following fields:

| Bits | Field | Description | | |
|---|---|---|---|---|
| 2:0 | BL | SDRAM Burst Length:<br>000 = 1 (default)<br>001 = reserved<br>010 = reserved<br>011 = reserved<br>1xx = reserved | | |
| 3 | Reserved | Hardwired to 0. | | |
| 6:4 | LTMD | SDRAM CAS Latency:<br>000 = reserved<br>001 = reserved<br>010 = 2<br>011 = 3 (default)<br>1xx = reserved | | |
| 7 | Reserved | Hardwired to 0. | | |
| 9:8 | RCD | SDRAM RAS-CAS Delay: | | |
| | | 00 = reserved | | |
| | | 01 = 2clock | 66MHz: 30.3ns | 100MHz: 20ns |
| | | 10 = 3clock | 66MHz: 45.5ns | 100MHz: 30ns |
| | | 11 = 4clock (default) | 66MHz: 60.6ns | 100MHz: 40ns |
| 31:10 | Reserved | Hardwired to 0. | | |

**Remark** RAS-CAS Delay time is depended on the system clock frequency.

### 3.4.5.4 SDRAM Type Selection Register (SDTSR)

The SDRAM Type Selection Register “SDTSR” is read-write and word aligned 32bit register. SDTSR is used to setup the type of SDRAM. SDTSR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description | |
|---|---|---|---|
| 1:0 | CAB | Number of column address bits (except bank select pins):<br>00 = 7bit (default)<br>01 = 8bit<br>10 = 9bit<br>11 = 10bit | |
| 3:2 | Reserved | Hardwired to 0. | |
| 6:4 | RAB | Total number of SDRAM address bits (RAS+CAS) (except bank select pins):<br>000 = 17bit (default)<br>001 = 18bit<br>010 = 19bit<br>011 = 20bit<br>100 = 21bit<br>101 = 22bit<br>110 = reserved<br>111 = reserved | |
| 7 | BTM | Number of Bank:<br>0 = 1 or 2 bank (default)<br>1 = 3 or 4 bank | |
| 9:8 | SDS | Total SDRAM Size: | |
| | | 00 = 4MByte (default) | 003F_FFFFH – 0000_0000H |
| | | 01 = 8MByte | 007F_FFFFH – 0000_0000H |
| | | 10 = 16MByte | 00FF_FFFFH – 0000_0000H |
| | | 11 = 32Mbyte | 01FF_FFFFH – 0000_0000H |
| 31:10 | Reserved | Hardwired to 0. | |

**EXAMPLE:**

| SDRAM | Parts No. | Type (bank x word x bit) | column address | row address | CAB | RAB |
|---|---|---|---|---|---|---|
| 16Mbit | μPD4516161 | 2 x 0.5M x 16 | 8 | 11 | 8 | 19 |
| 64Mbit | μPD4564323 | 4 x 0.5M x 32 | 8 | 11 | 8 | 19 |
| 64Mbit | μPD4564163 | 4 x 1.0M x 16 | 8 | 12 | 8 | 20 |
| 128Mbit | μPD45128163 | 4 x 2.0M x 16 | 9 | 12 | 9 | 21 |

### 3.4.5.5 SDRAM Precharge Timing Register (SDPTR)

The SDRAM Precharge Timing Register “SDPTR” is read-write and word aligned 32bit register. SDPTR is used to set the Precharge Timing for the SDRAM Controller. SDPTR is initialized to 142H at reset and contains the following fields:

| Bits | Field | Description | | |
|---|---|---|---|---|
| 1:0 | PAT | Precharge command -> Active command timing ($t_{RP}$): | | |
| | | 00 = 2clock | 66MHz: 30.3ns | 100MHz: 20ns |
| | | 01 = 3clock | 66MHz: 45.5ns | 100MHz: 30ns |
| | | 10 = 4clock(default) | 66MHz: 60.6ns | 100MHz: 40ns |
| | | 11 = reserved | | |
| 3:2 | Reserved | Hardwired to 0. | | |
| 6:4 | APT | Active command -> Precharge command timing ($t_{RAS}$): | | |
| | | 000 = 4clock | 66MHz: 60.6ns | 100MHz: 40ns |
| | | 001 = 5clock | 66MHz: 75.7ns | 100MHz: 50ns |
| | | 010 = 6clock | 66MHz: 90.9ns | 100MHz: 60ns |
| | | 011 = 7clock | 66MHz: 106.0ns | 100MHz: 70ns |
| | | 100 = 8clock(default) | 66MHz: 121.2ns | 100MHz: 80ns |
| | | 101 = reserved | | |
| | | 110 = reserved | | |
| | | 111 = reserved | | |
| 7 | Reserved | Hardwired to 0. | | |
| 8 | DPL | Input data -> Precharge command timing ($t_{DPL}$): | | |
| | | 0 = 1clock | 66MHz: 15.2ns | 100MHz: 10ns |
| | | 1 = 2clock(default) | 66MHz: 30.3ns | 100MHz: 20ns |
| 31:9 | Reserved | Hardwired to 0. | | |

### 3.4.5.6 SDRAM Refresh Mode Register (SDRMR)

The SDRAM Refresh Mode Register “SDRMR” is read-write and word aligned 32bit register. SDRMR is used to initialize the SDRAM Refresh Controller. SDRMR is initialized to 0000_0200H at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 15:0 | RCSET | Reload value for SDRAM Refresh Timer Counter.<br>This value, in system clock ticks, is automatically reloaded into the Refresh timer counter after the counter reached zero. The Refresh timer counter counts down from this value. Thus, time of the count cycle corresponds to 1 plus this filed value. The default value (200H = 512) is the refresh rate for an SDRAM chip that requires 4096 refresh cycle every 32ms (ex. one refresh every 7.8125us) for system clock running at 66MHz. This is very conservative but it allows for successful boot, after which the reload value can be increased. RCSET[7:0] is hardwired to 0, thus the timer value less than 100H cannot set this field. If such value is set into this field, the default value “200H” is automatically loaded. |
| 31:16 | Reserved | Hardwired to 0. |

### 3.4.5.7 SDRAM Refresh Timer Count Register (SDRCR)

The SDRAM Refresh Timer Count Register “SDRCR” is read-only and word aligned 32bit register. SDRCR is 16bit timer that causes an SDRAM refresh when it expires. The SDRAM refresh controller automatically reloads this free-running timer. SDRCR is initialized to 000_0200H at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 15:0 | RCC | This filed shows the current value of SDRAM Refresh Timer. |
| 31:16 | Reserved | Hardwired to 0. |

### 3.4.5.8 Memory Bus Control Register (MBCR)

The Memory Bus Control Register “MBCR” is read-write and word aligned 32bit register. MBCR is used to selects priority for ether CPU or IBUS for memory access. The VR4120A can assign higher priority to CPU request for memory than IBUS request or assign equal priority to CPU and IBUS request for memory. MBCR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 0 | BPR | Priority for memory access:<br>0 : SysAD(CPU) > IBUS = Refresh (default)<br>The memory arbiter allows the CPU to perform one memory transaction when the current word counts of burst access from IBUS reaches each 16words or 32words.<br>1 : SysAD(CPU) = IBUS = Refresh<br>The memory arbiter doses allows the CPU to perform the memory transaction when the burst memory access form IBUS are |
| 31:1 | Reserved | Hardwired to 0. |

#### 3.4.5.9 Memory Error Status Register (MESR)

The Memory Error Status Register “MESR” is read-cleared and word aligned 32bit register. MESR indicates the error status in the Memory Controller. MESR can accept the next error after the CPU read both MESR and MEAR Register (see below). MESR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 0 | SBAER | The memory access form SysAD cause an address range error:<br>0 = No Error<br>1 = Error is occurred |
| 1 | IBAER | The memory access form IBUS cause an address range error:<br>0 = No Error<br>1 = Error is occurred |
| 2 | RMAE | Indicate the address error in ROM space.<br>0 = No Error in ROM space.<br>1 = Error is occurred in ROM space. |
| 3 | SDAE | Indicate the address error in SDRAM space.<br>0 = No Error in SDRAM space.<br>1 = Error is occurred in SDRAM space. |
| 7:4 | Reserved | Hardwired to 0. |
| 8 | RMWE | ROM write error:<br>0 = No Error<br>1 = Illegal ROM access, byte or half-word or burst write, is occurred. |
| 31:9 | Reserved | Hardwired to 0. |

#### 3.4.5.10 Memory Error Address Register (MEAR)

The Memory Error Address Register “MEAR” is read-cleared and word aligned 32bit register. MEAR indicates the memory address in the Memory Controller. MEAR can accept the next error address after the CPU read both MESR Register and MEAR. MEAR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 31:0 | BEA | This field maintains a memory address in error. |

### 3.4.6 Boot ROM

The System Controller supports up to 8MB of boot memory. This memory must be populated with either of the following two types of memory devices:

#### 3.4.6.1 Boot ROM configuration and address ranges

Boot ROM can be populated with PROM or 85-ns Flash chips, and it must have an access time of 200ns or less. The System Controller supports 8,16 and 32-bit Boot ROM at locations 1F80_0000H through 1FFF_FFFFH in the physical memory space on VR4120A. The Boot ROM does not support CPU cache operations.

**Table 3-6. Boot-ROM Size Configuration at Reset**

| RMMDR.FSM | Boot ROM Size | Address Range |
|---|---|---|
| 00 | 4MB | 1FC0_0000H through 1FFF_FFFH |
| 01 | 8MB | 1F80_0000H through 1FFF_FFFH |
| 10 | 1MB | 1FC0_0000H through 1FCF_FFFH |
| 11 | 2MB | 1FC0_0000H through 1FDF_FFFH |

| ROMSEL[1:0] | 11 | 10 | 01 | | 00 | |
|---|---|---|---|---|---|---|
| | --- | 8bit | 16bit | | 32bit | |
| RMMDR.FSM | | Byte Mode | Word Mode | Byte Mode | Word Mode | Byte Mode |
| 00 | --- | --- | 4MB (2Mx16bit) | 2MBx2 (2Mx8bit) | 2MBx2 (1Mx16bit) | 1MBx4 (1Mx8bit) |
| 01 | --- | --- | --- | --- | 4MBx2 (2Mx16bit) | 2MBx4 (2Mx8bit) |
| 10 | --- | 1MB (1Mx8bit) | --- | --- | --- | --- |
| 11 | --- | 2MB (2Mx8bit) | 2MB (1Mx16bit) | 1MBx2 (1Mx8bit) | --- | --- |

The controller asserts the FLASH/ROM chip select (SRMCS_B) in the address range 1F80_0000H through 1FFF_FFFFH. When writes are performed to the ROM/FLASH memory space, the controller asserts SRMWE_B in conjunction with SRMCS_B. When reads are performed, the controller asserts SRMOE_B in conjunction with SRMCS_B. If the CPU attempts to access Boot ROM addresses outside the defined size of the FLASH/ROM, the controller returns 0 with the data error bit set on SysCMD[0]. In addition, the NMI Status Register “S_NSR” is updated and NMI is asserted on the signal, if the interrupt is enabled in the NMI Mask Register “S_NMR”.

#### 3.4.6.2 FLASH memory write-protection

The FLASH Memory can be protected in software. Software protection is implemented by programming the WM field in ROM Mode Register “RMMDR”.

#### 3.4.6.3 FLASH memory operations

The FLASH Memory can be programmed using following write cycle sequence by VR4120A. The following commands are example of operations for the NEC μPD29F800L FLASH Memory (in Byte Mode).

**Table 3-7. Command Sequence**

**(a) Program Command Sequence (4 Write Cycles)**

| 1st Write | | 2nd Write | | 3rd Write | | 4th Write | | 5th Write | | 6th Write | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 1FC2AAA8H | A | 1FC15554H | A | 1FC2AAA8H | A | PA* | A | | A | |
| D | AAAAAAAAH | D | 55555555H | D | A0A0A0A0H | D | PD* | D | | D | |

**(b) Chip Erase Command Sequence (6 Write Cycles)**

| 1st Write | | 2nd Write | | 3rd Write | | 4th Write | | 5th Write | | 6th Write | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 1FC2AAA8H | A | 1FC15554H | A | 1FC2AAA8H | A | 1FC2AAA8H | A | 1FC15555H | A | 1FC2AAA8H |
| D | AAAAAAAAH | D | 55555555H | D | 80808080H | D | AAAAAAAAH | D | 55555555H | D | 10101010H |

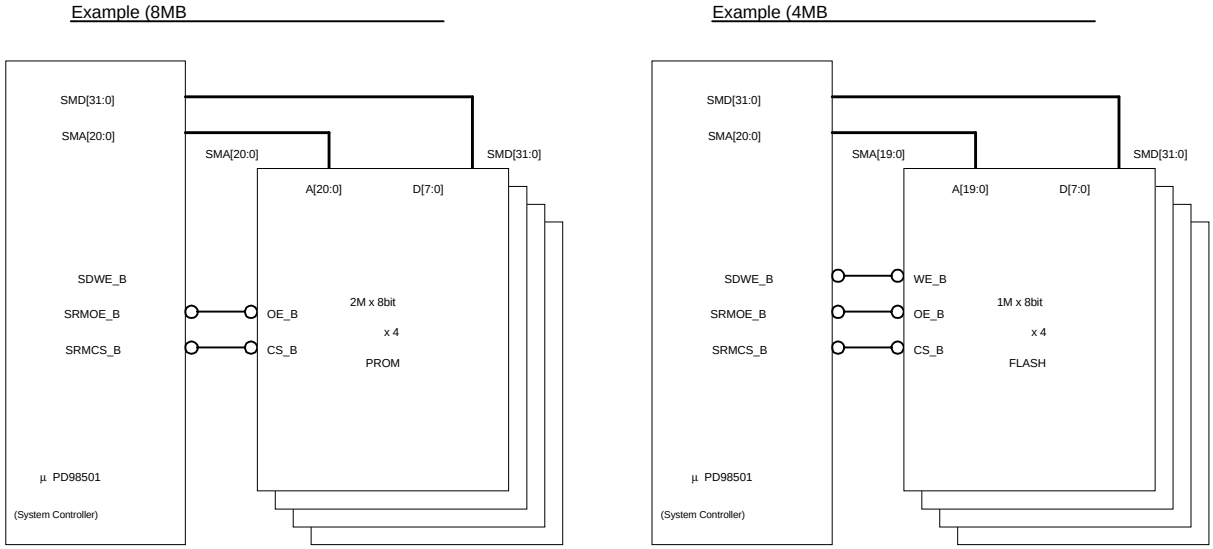
**(c) Sector Erase Command Sequence (6 Write Cycles)**

| 1st Write | | 2nd Write | | 3rd Write | | 4th Write | | 5th Write | | 6th Write | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 1FC2AAA8H | A | 1FC15554H | A | 1FC2AAA8H | A | 1FC2AAA8H | A | 1FC15554H | A | EA* |
| D | AAAAAAAAH | D | 55555555H | D | 80808080H | D | AAAAAAAAH | D | 55555555H | D | 30303030H |

**Remark** A = Memory Write Address
D = Memory Write Data
PA = Address of FLASH Location to be programmed.
EA = Data to be programmed at Location PA.
EA = Block Address of FLASH Location to be erased.

### 3.4.6.4 Boot ROM signal connections

FLASH/ROM Configuration



### 3.4.7 SDRAM

#### 3.4.7.1 SDRAM address range

System Memory can be populated with SDRAM chips, and it must have an access time of 10ns or less. The System Controller supports 16Mbit or 64Mbit and 128Mbit SDRAM at locations 0000_0000H through 01FF_FFFFH in the physical memory space on VR4120A. The SDRAM supports CPU cache operations,.

**Table 3-8. SDRAM Size Configuration at Reset**

| SDMDR.SDS | SDRAM Size | Address Range |
|---|---|---|
| 00 | 4MB | 0000_0000H through 003F_FFFH |
| 01 | 8MB | 0000_0000H through 007F_FFFH |
| 10 | 16MB | 0000_0000H through 00FF_FFFH |
| 11 | 32MB | 0000_0000H through 01FF_FFFH |

#### 3.4.7.2 SDRAM device configurations

The controller supports the following 16Mbit 64Mbit and 128Mbit. NEC SDRAM parts and configurations in System Memory. Following Table shows some of the SDRAM configurations supported for system memory.

**Table 3-9. SDRAM Configurations Supported**

| Memory Size | SMA Address Bits Required | Organization (bank x word x bit) | Quantity | NEC Part Number |
|---|---|---|---|---|
| 4MB | 12:0 | 2 x 0.5M x 16 | 2 | μPD4516161 |
| 8MB | 12:0 | 4 x 0.5M x 32 | 1 | μPD4564323 |
| 16MB | 13:0 | 4 x 1.0M x 16 | 2 | μPD4564163 |
| 32MB | 13:0 | 4 x 2.0M x 16 | 2 | μPD45128163 |

#### 3.4.7.3 SDRAM burst-type and banks

The terms interleaved and bank have multiple meanings in the context of memory design using SDRAM chips. The meanings are:

- Banks (applied to memory modules and SDRAM chips in different ways): The banks referenced with respect to memory modules differ from the banks inside an SDRAM chip. For module, This controller does not support what identifies their bank. Following Table shows bank select signals.

**Table 3-10. SDRAM Bank Select Signals Mapping**

| MuxAd Bank Signal | Memory Bank select inputs | NEC Parts Number |
|---|---|---|
| SMA [11] | A[11] | μPD4516161 |
| SMA [12:11] | BA[1:0] | μPD4564323 |
| SMA [13:12] | A[13:12] | μPD4564163 |
| SMA [13:12] | A[13:12] | μPD45128163 |

- Burst Type (applies to SDRAM chips): The burst type of a single SDRAM chip is programmed in the chip's Mode Register to be either interleaved or sequential. This concept relates only to the word-order in which data is read into and written out of the SDRAM chip. The concept does not relate to the number of words transferred in a given clock cycle. The burst type for all SDRAM chips attached to the μPD98501 is configured during the Memory Initialization procedure. The memory controller in the System Controller does NOT support the interleaved burst mode and support only sequential burst mode.

#### 3.4.7.4 SDRAM word ordering

Following Table shows the word-address order for a 4-word instruction-cache line fill from SDRAM. This order is determined by the SDRAM chips' burst type, which is programmed during the Memory Initialization procedure. The memory controller programs the burst type and word order the same for all SDRAM chips connected to it (in the System Memory ranges). The term "sequential" in this table refers to the SDRAM burst type. Burst length depends only on the access type performed by the CPU.

**Table 3-11. SDRAM Word Order for Instruction-Cache Line-Fill**

| Start Column Address A1.A0 | SDRAM-Chip Burst Type | |
|---|---|---|
| | Sequential | Interleaved |
| 00 | 0-1-2-3 | not supported |
| 01 | 1-2-3-0 | not supported |
| 10 | 2-3-0-1 | not supported |
| 11 | 3-0-1-2 | not supported |

**Remark** The memory controller does not support the interleaved burst type for SDRAMs. It assumes that all SDRAMs are initialized to the sequential burst type, using a burst length of 4 words.

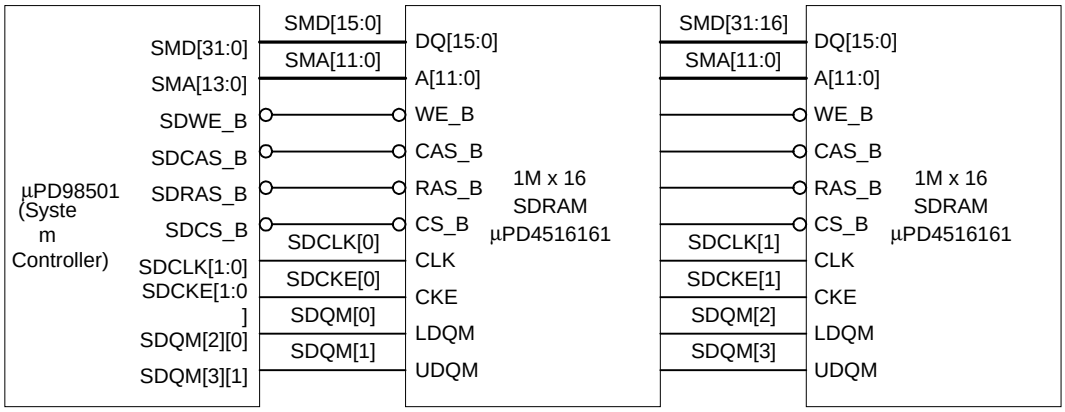
#### 3.4.7.5 SDRAM signal connections

Following Figure shows how the NEC 16Mbit SDRAMs are connected for System Memory. SMA[11] is the bank select signal. In command cycle, SMA[11] low selects Bank A and SMA[11] High selects Bank B. Both banks share the same SDCSB, SDRASB, SDCASB, and SDWEB signals.

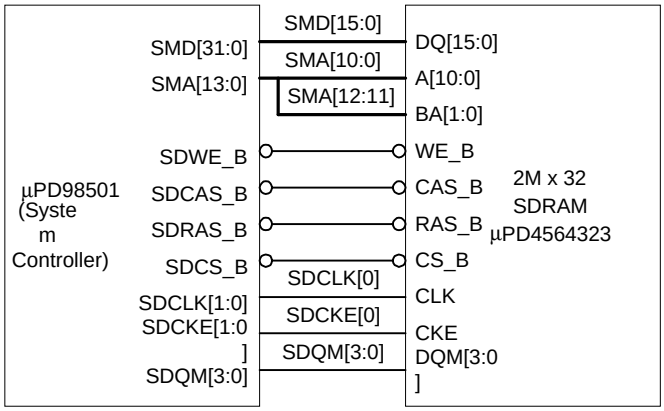
The two banks of System Memory behave as two halves of the address range, with the highest unmasked address bit controlling bank selection.

# SDRAM Configuration

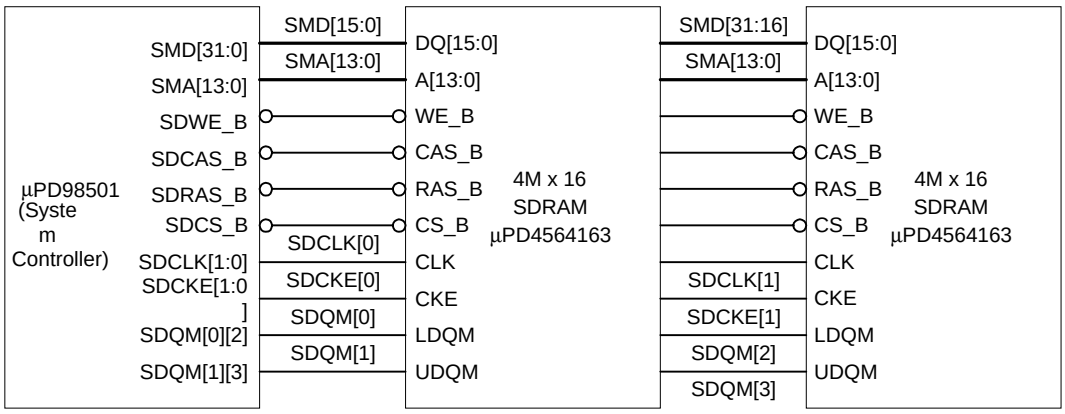
4MB



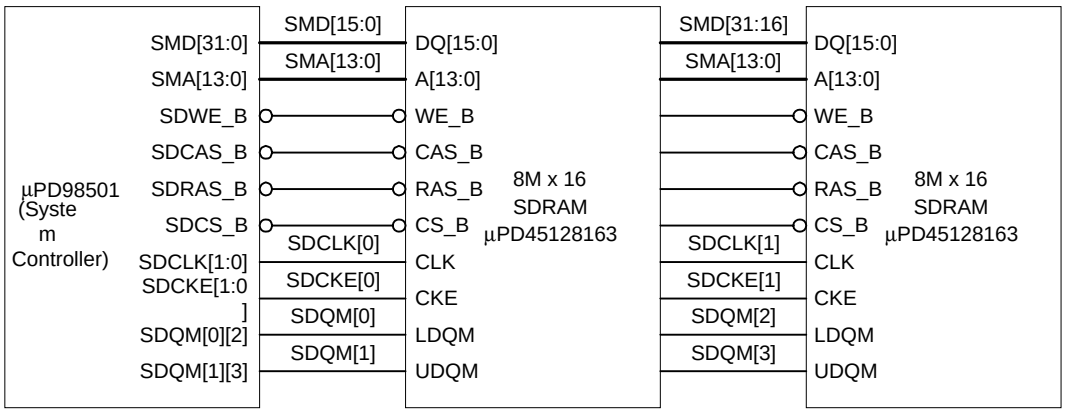

8MB




16MB




32MB



### 3.4.8 SDRAM refresh

The System Controller supports CAS-Before-RAS (CBR) DRAM refresh to all SDRAM address ranges. The refresh clock is derived from the system clock; its rate is determined by programming the RCR filed in the SDRAM Refresh Mode Register “SDRMR”.

The refresh logic requests access to SDRAM from the internal bus-arbitration logic each time the counter reaches 0. The refresh logic can accumulate up to a maximum of 15 refresh requests while it is waiting for the bus. Once the refresh logic owns the bus, all accumulated refreshes are performed to system memory, and no other accesses (CPU or IBUS) are allowed. Refreshes are staggered by one clock; that is, there will be at least one bus clock between transitions on any pair of SDRASB signals. Refresh automatically clears the system-memory prefetch FIFO.

### 3.4.9 Memory-to-CPU prefetch FIFO

The memory controller automatically prefetches 4 words from system memory into a 4-words prefetch FIFO. That is, after each burst 4-words read, the memory controller prefetches 4 additional words into its internal prefetch FIFO. If the processor subsequently attempts a read from an address immediately following (sequential to) the address of the last read cycle, the first 4 words will supplied from the prefetch FIFO.

The memory controller compares the current SysAD address with the previous address to determine the sequential nature of the access. Prefetched words are retained in the prefetch FIFO if accesses to resources other than System Memory are performed between System Memory accesses.

### 3.4.10 CPU-to-memory write FIFO

The memory controller has a 4-word CPU-to-Memory Write FIFO. This FIFO accepts writes at the maximum CPU speed. A single address is held for the buffered write, allowing the buffering of a single write transaction. That transaction may be a word, double-word, 4-word data-cache write-back. When a word is placed in the FIFO by the CPU, the memory controller attempts to write the FIFO’s contents to memory as quickly as possible. If the next CPU read or write is addressed to memory, the controller negates ready signal, thus causing the next CPU transaction (read or write) to stall until the controller empties its FIFO. If the next CPU transaction (read or write) is addressed to a IBUS target, the memory controller asserts ready signal, thus the CPU transaction to complete.

### 3.4.11 SDRAM memory initialization

The memory controller automatically initializes the SDRAM after the global reset. The following sections describe the configuration sequence used for this initialization.

#### 3.4.11.1 Power-on initialization sequence by memory controller

Follow this sequence to configure memory after reset:

1. Waits for 100us after power-on.
2. Performs all bank Precharge.
3. Performs two sequential auto Refresh (CBR).

#### 3.4.11.2 Memory initialization sequence using software

The SDRAM must be initialized by the Software using following sequence after power-on initialization.

1. Program the SDRAM Type Selection Register “SDTSR”
2. Program the SDRAM Mode Register “SDMDR”.
3. Wait for 20us.
4. Program the DRAM Refresh Counter Register.

At this point, memory is ready to use. All other configuration registers in the controller should then be programmed before commencing normal operation.

**Remark** The Software should NOT change the SDTSR and SDMDR after SDRAM initialization sequence

## 3.5 IBUS Interface Register

### 3.5.1 ITCNTR (IBUS timeout timer control register)

The IBUS Timeout timer Control Register “ITCNTR” is read-write and word aligned 32bit register. ITCNTR is used to enable use of the IBUS Timeout Timer. ITCNTR is initialized to 0H at reset and contains the following field:

| Bits | Field | Description |
|---|---|---|
| 0 | ITWEN | IBUS Timeout Timer enable<br>1 = Enable<br>0 = Disable |
| 31:1 | Reserved | Hardwired to 0. |

### 3.5.2 ITSETR (IBUS timeout timer set register)

The IBUS Timeout timer Set Register “ITSETR” is read-write and word aligned 32bit register. ITSET is used to detect the stall of the IBUS using the bus timer. The IBUS Timeout Timer counts the rising edge of the CPU clock while the IBUS Frame Signal (ibframe) is asserting. If the bus timer reaches the following ITTIME value, IBUS timer assert the NMI to CPU when of NER Register is set. The IBUS Timeout value can be set in 1-clock increments in a range from 1 to 2^32-1 CPU clock. However, the CPU operation is undefined when 0 has been set to ITTIME field. ITSETR is initialized to 8000_0000H at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 31:0 | ITTIME | IBUS Timeout value setting<br>Timeout = ITTIME value * system clock period(100MHz:10ns,66MHz:15ns)<br><br>example:<br>ITTIME=05F5E100H (100MHz) or 03F940AAH(66MHz) -> Timeout = 1sec<br>ITTIME=0BEBC200H(100MHz) or 07F28154H(66MHz) -> Timeout = 2sec<br>ITTIME=11E1A300H(100MHz) or 0BEBC200H(66MHz) -> Timeout = 3sec<br>: : : |

## 3.6 DSU (Deadman's SW Unit)

### 3.6.1 Overview

The DSU detects when the CPU is in runaway (endless loop) state and resets the CPU to minimize runaway time. The use of the DSU to minimize runaway time effectively minimizes data loss that can occur due to software-related runaway states.

### 3.6.2 Registers

#### 3.6.2.1 DSUCNTR

This register is used to enable use of the Deadman's Switch functions.

DSUCNTR is 32bit word-aligned register. Default is 00000000H.

| Bits | Field | Description |
|---|---|---|
| 0 | DSWEN | Deadman's Switch function enable<br>1 = Enable<br>0 = Disable |
| 31:1 | Reserved | Hardwired to 0. |

#### 3.6.2.2 DSUSETR

This register sets the cycle for Deadman's Switch functions. The Deadman's Switch cycle can be set in 1-clock increments in a range from 1 to 2^32-1 clock. The DSUCLRR's DSWCLR bit must be set by means of software within the specified cycle time. DSUSETR is 32bit word-aligned register. Default is 80000000H.

| Bits | Field | Description |
|---|---|---|
| 31:0 | DEDTIM | Deadman's Switch cycle setting<br>DSU cycle = DEDTIME value * system clock period(100MHz:10ns,66MHz:15ns)<br>example:<br>DEDTIM=05F5E100H (100MHz) or 03F940AAH(66MHz) -> DSU cycle = 1sec<br>DEDTIM=0BEBC200H(100MHz) or 07F28154H(66MHz) -> DSU cycle = 2sec<br>DEDTIM=11E1A300H(100MHz) or 0BEBC200H(66MHz) -> DUS cycle = 3sec<br>: : : |

#### 3.6.2.3 DSUCLRR

This register clears the Deadman's Switch counter by setting the DSWCLR bit in this register to 1. The CPU automatically shuts down if 1 is not written to this register within the period specified in DSUSETR. DSUCLR is 32bit word-aligned register. Default is 00000000H.

| Bits | Field | Description |
|---|---|---|
| 0 | DSWCLR | Deadman's Switch counter clear. Cleared to 0 when 1 is written.<br>1=Clear<br>0=Don't clear |
| 31:1 | Reserved | Hardwired to 0. |

#### 3.6.2.4 DSUTIMR

This register indicates the elapsed time for the current Deadman's Switch timer.
DSUTIMR is 32bit word-aligned and read-only register. Default is 00000000H.

| Bits | Field | Description |
|---|---|---|
| 31:0 | CRTTIM | Current Deadman's Switch timer value (elapsed time)<br>example:<br>CRTTIM=05F5E100H (100MHz) or 03F940AAH(66MHz) ->1sec<br>CRTTIM=0BEBC200H(100MHz) or 07F28154H(66MHz) -> 2sec<br>CRTTIM=11E1A300H(100MHz) or 0BEBC200H(66MHz) -> 3sec<br>: : : |

### 3.6.3 DSU register setting flow

The DSU register setting flow is described below.

1. Set the DSU's count-up value (From 1 to 2^31-1).
   The CPU will be reset if it does not clear (1 is not written to DSUCLRR) the timer within this time period.
2. Enable the DSU
3. Clear the timer within the time period specified in step 1 above.
   For normal use, repeat step 3. To obtain the current elapsed time:
4. Disable the DSU for shutdown.

## 3.7 Endian Mode Software Issues

### 3.7.1 Overview

The native endian mode for MIPS processors, like Motorola and IBM 370 processors, is big endian. However, the native mode for Intel (which developed the PCI standard) and VAX processors is little endian. For PCI-compatibility reasons, most PCI peripheral chips operate natively in little-endian mode. While the μPD98501 is natively little-endian, it supports either big- or little-endian mode on the SysAD bus. The state of the ENDIAN signal at reset determines this endian mode. However, there are important considerations when using the controller in a mixed-endian design. The most important aspect of the endian issue is which byte lanes of the SysAD bus are activated for a particular address. If the big-endian mode is implemented for the CPU interface, the controller swaps bytes within words and halfwords that are coming in and going out on the SysAD bus. All of the System Controller's other interfaces operate in little-endian mode.

The sections below view the endian issue from a programmer's perspective. They describe how to implement mixed-endian designs and how to make code endian-independent.

Data in memory is always ordered in little-endian mode, even with a big-endian CPU.

Data in all internal registers and FIFOs is considered little-endian regardless of CPU endianness.

Data addresses are not swapped inside the device for accesses from a little-endian CPU to all local registers when "ENDCEN" bit in the General Status Register "S_GSR" is reset.

Data addresses are swapped inside the device for accesses from a big-endian CPU to all local registers when "ENDCEN" bit in the General Status Register "S_GSR" is set.

Data addresses are swapped inside the device for accesses from a big-endian CPU to memory when "ENDCEN" bit in the General Status Register "S_GSR" is set.

### 3.7.2 Endian modes

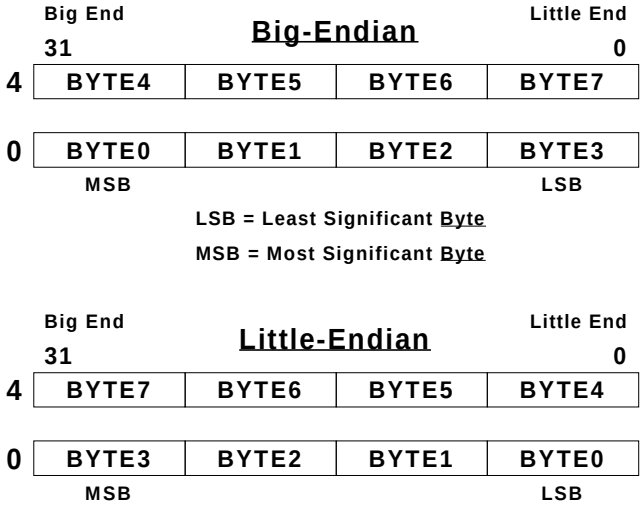
The endian mode of a device refers to its word-addressing method and byte order:

Big-endian devices address data items at the big end (most significant bit number). The most-significant byte (MSB) in an addressed data item is at the lowest address.

Little-endian devices address data items at the little end (least significant bit number). The most significant byte (MSB) in an addressed data item is at the highest address.

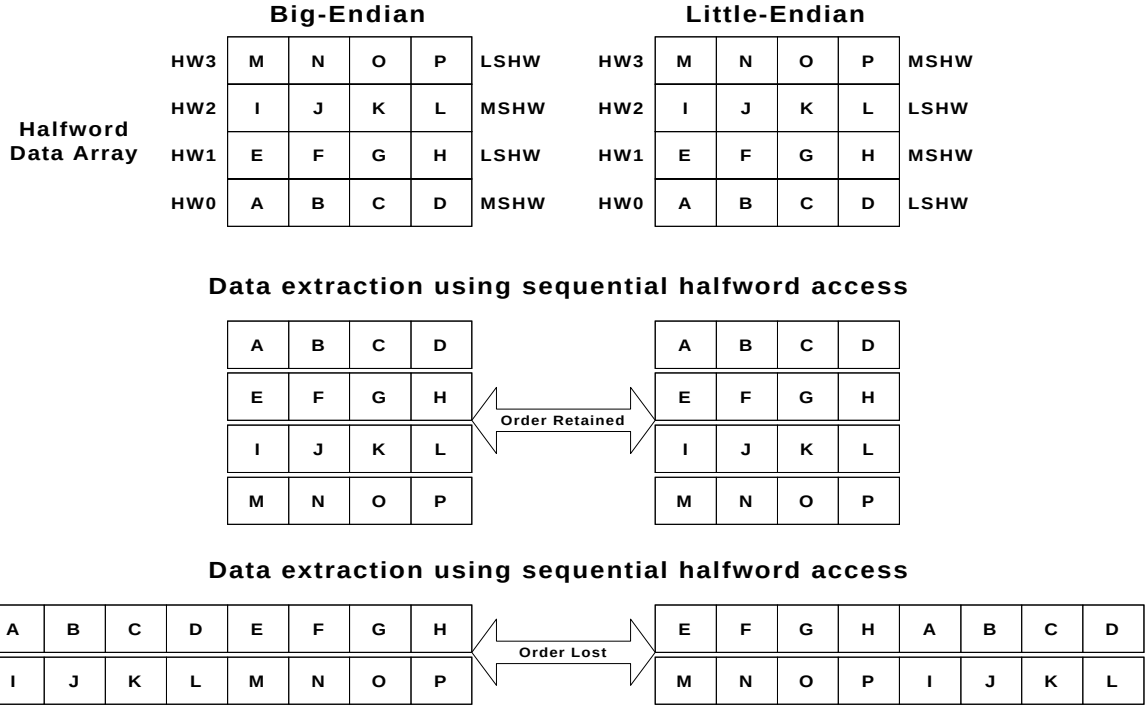
The following figures shows the bit and byte order of the two endian modes, as it applies to bytes within word-sized data items. The bit order within bytes is the same for both modes. The big (most-significant) bit is on the left side, and the little (least significant) bit is on the right side. Only the bit order of sub-items is reversed within a larger addressable data item (halfword, word, doubleword) when crossing between the two endian modes. The sub-items' order of significance within the larger data item remains the same. For example, the least significant half word (LSHW) in a word is always to the right and the most-significant halfword (MSHW) is to the left.

**Figure 3-1. Bit and Byte Order of Endian Modes**




If the access type matches the data item type, no swapping of data sub-items is necessary. Thus, when making half-word accesses into a data array consisting of halfword data, no byte-swapping takes place. In this case, data item bit order is retained between the two endian modes. The code that sequentially accesses the half-word data array would be identical, regardless of the endianness of its CPU. The code would be endian-independent.

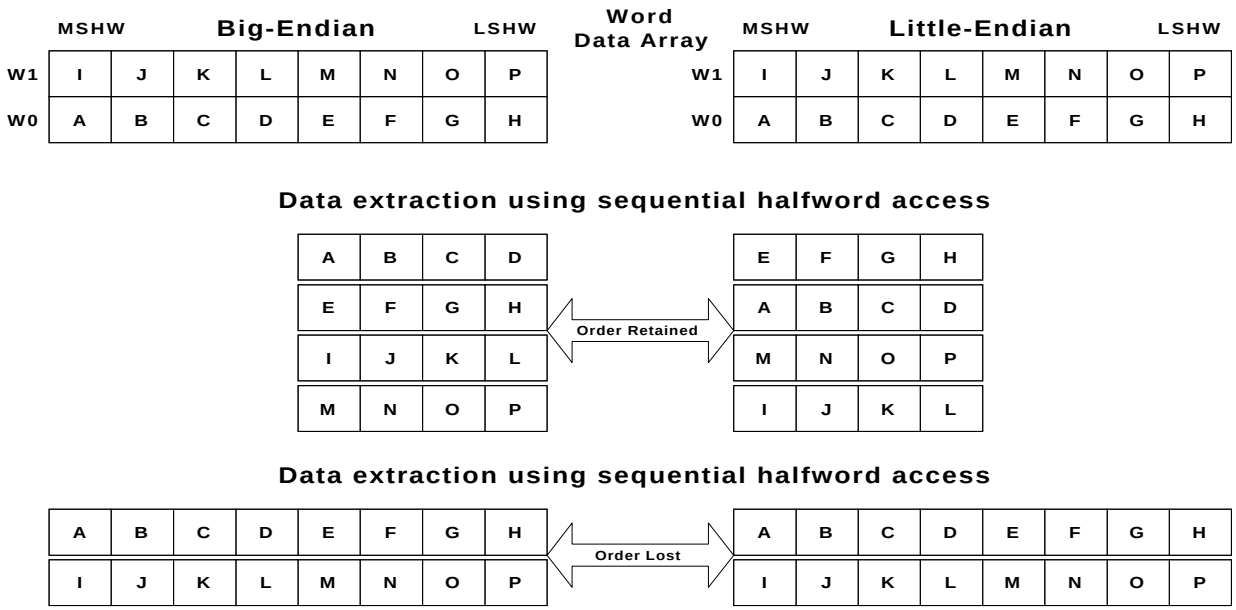
**Figure 3-2. Halfword Data-Array Example**




However, when making half-word accesses into a data array consisting of word data, access to the more-significant halfword requires the address corresponding to the less significant half word (and vice versa). Such code is not endian-independent. A supergroup access (for example, accessing two half words simultaneously as a word from

a half-word data array) causes the same problem. Such problems also arise when a halfword access is made into a 32-bit register, whereas a word access into a 32-bit register creates no problem.

**Figure 3-3. Word Data-Array Example**



**[MEMO]**

# CHAPTER 4 ATM CELL PROCESSOR

## 4.1 Overview

This section describes functional specifications of ATM cell processor unit.

### 4.1.1 Function features

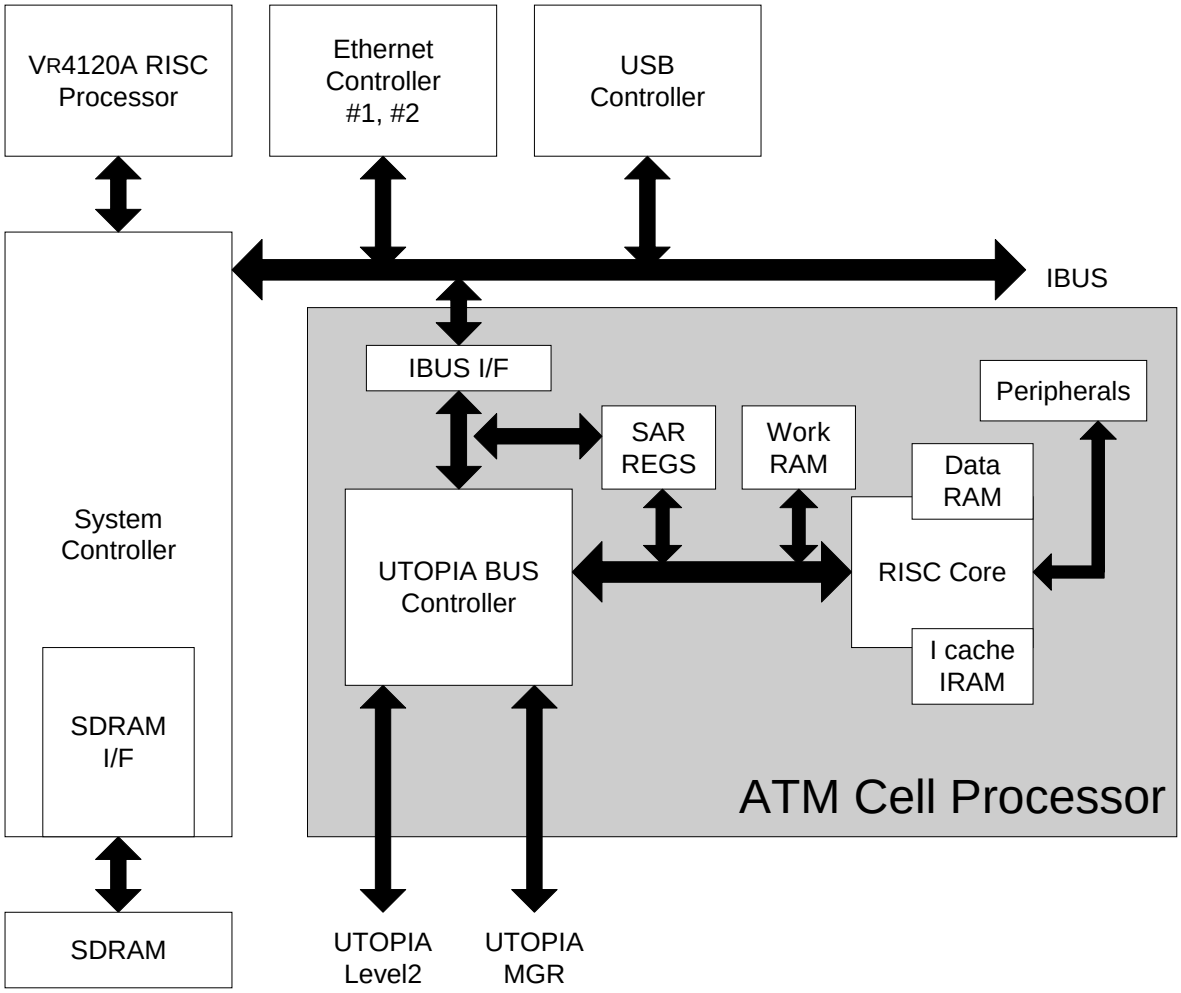
Features of ATM Cell Processor is as follows:

- Data Transmission Capacity
  Aggregated transmission capacity is 50Mbps, 25Mbps for downstream and 25Mbps for upstream.
- Supported ATM Adaptation Layers (AAL)
  AAL0 (raw cells) and AAL5 are supported. AAL2 will be supported.
- Supported Service Classes
  CBR, VBR, ABR, GFR, and UBR.
- Number of VC's
  Maximum of 64 VC's will be supported.
- Scheduling
  Cell rate shaping will be performed in one-cell-time granularity on per VC basis
- Support IP Flow
  IP flow will be mapped dynamically to ATM VCC.
- Support LLC and PPP encapsulation
- OAM will be supported.
- Switching function will be supported.
- Re-configurable F/W

ATM Cell Processor has a 32-bit micro-controller who controls ATM cells. It has its F/W in an internal instruction RAM. F/W will be stored in an external flash ROM to be re-writable /re-configurable.

### 4.1.2 Block diagram of ATM cell processor

**Figure 4-1. Block Diagram of ATM Cell Processor**




This block is a “Soft-ware” ATM cell processor. It consists a 32-bit MCU, Peripherals (Interrupt Controller, Cell Timer, Scheduling Table, Rx Lookup Table, and IP Lookup Table), DMA controllers, a Work-RAM, and SAR-Registers.

#### 4.1.2.1 RISC core

This block is RISC micro-controller. Its features are as follows:

- High performance 32-bit RISC micro-controller, 76MIPS @ 66MHz
- 32x32-bit General Purpose Registers
- 32-bit ALU, 32-bit Shifter, 16x16 Multiply-Adder
- 1KB Data RAM, 8KB Instruction RAM, 8KB Instruction Cache

#### 4.1.2.2 Peripherals

- Interrupt Controller (INTC) and Interrupt Edge Detector (INTEDGE)
- Peripherals for ATM – Scheduling Table, Rx Lookup Table, IP Lookup Table, and Cell Timer

#### 4.1.2.3 UTOPIA bus controller

This block has some H/W resources - DMA controller, FIFOs, CRC calculators/checkers - inside, and realizes “SW-SAR” being controlled by F/W on RISC Core. Its features are as follows:

- Scatter/Gather-DMA controller which can operate the distributed datum according to descriptor tables, without F/W help

  This DMA controller use for each transmission and reception.
  For the improvement of the performance, UTOPIA Bus Controller has the scatter/gather-DMA controllers, which can handle distributed datum without F/W helps. This DMA controller can write datum to distributed areas (scattering) and read datum from distributed areas (gathering), according to descriptor tables. Normal DMA mode is also supported.
  Furthermore This DMA controller updates the information related to the DMA operations in VC table in Work RAM.

- Internal BUS interface(IBUS)
- External memory interface
- 32-bit bus interface that can be directly connected to the RISC Core
- ATM Zero-padding

  Zero-padding is often required to make ATM cells. This block has the circuits for the padding. Only giving the source address and the number of bytes to be padded, this block pads “0” as desired.

- 66MHz clock
- Transmission and reception SAR FIFO consisting of four cells each

UTOPIA I/F Control block has a four-cell-depth FIFO for each transmission and reception. The last cell in Tx FIFO and the first cell in Rx FIFO are mapped to VR4120A RISC Processor/ RISC Core memory space.

UTOPIA 2 I/F is an 8-bit bus I/F to PHY devices, which is defined in a ATM Forum document, “ATM-PHY-0039”. UTOPIA MGR I/F will be supported as well.

Its features are as follows:

- It supports up to 16 PHY devices at a time. Either PHY addresses from zero through fifteen or from sixteen through thirty can be selected by setting command register.
- The first word of cell header and the last two word of payload can be read in Big-Endien byte order to insert/extract some special bit-fields.
- In Tx side, it generates payload for an idle cell and an unassigned cell automatically when detects their pre-determined header pattern.
- To avoid Head-of-line Blocking, later cells can pass earlier cells if their destination PHY devices are not ready.
- In Rx side, it filters out idle cells and unassigned cells when detects their pre-determined header pattern.

- UTOPIA Level2 data path interface as a physical layer data interface
- UTOPIA Level2 management interface as a physical layer management interface
- CRC32/CRC10 calculator & checker

UTOPIA Bus Controller block has the CRC32 calculator for each transmission and reception. CRC32 value is calculated for every packet. For transmission, CRC32 value is inserted into the CRC32 field in trailer. For reception, CRC32 value is compared with the value in the CRC32 field in trailer, in order to check whether any errors occurred or not.

UTOPIA Bus Controller block has CRC10 calculator for each transmission and reception, too. It can be selected if CRC10 is inserted to the payload or not. CRC10 value is calculated for every cell. For transmission, CRC10 value is inserted into the last 10-bit area of the payload if selected. For reception, CRC10 value is compared with the value in the last 10-bit of the payload if selected.

#### 4.1.2.4 Other blocks

Register set is similar to that of the μPD98401A/μPD98405. Work-RAM is 12K-byte memory. It is shared between RISC Core and UTOPIA Bus Controller block. It also can be accessed by VR4120A RISC Processor, using Indirect-Access. Tables and Pool Descriptors are located in this RAM.

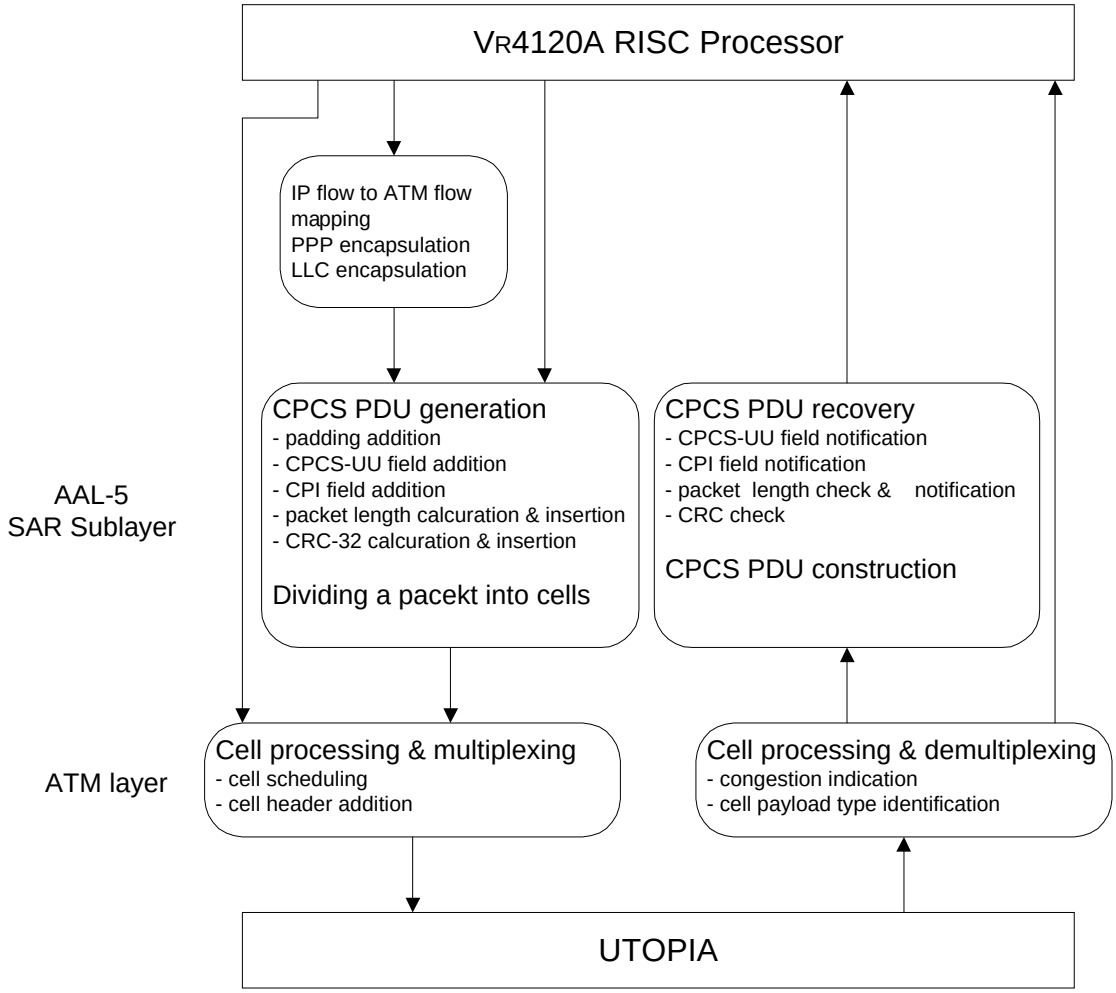
IBUS is a 32-bit PCI-like bus.

### 4.1.3 ATM cell processing operation overview

In this section, only overview is described. Please refer to Section. 4.7 for more detailed information.

ATM Cell Processor supports AAL-5 SAR sublayer and ATM layer functions. This block supports not only ATM protocol but upper layer protocol as well. As an upper layer support, this block provides dynamic IP flow/ ATM flow mapping, LLC encapsulation and PPP encapsulation.

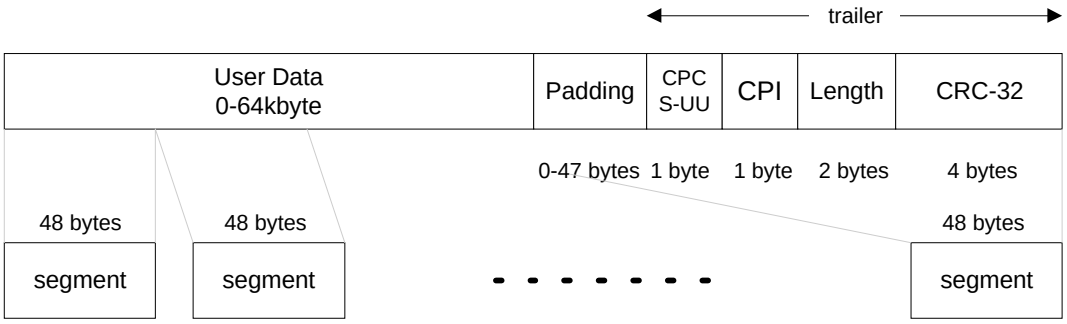
**Figure 4-2. AAL-5 Sublayer and ATM Layer**



#### 4.1.3.1 AAL-5 SAR sublayer function

When ATM Cell Processor transmits a cell, it adds a trailer to the variable-length data, as well as padding, so that their overall length becomes a multiple of 48 bytes, thereby generating an AAL-5 PDU. When ATM Cell Processor receives a cell, it builds up a CPCS PDU from cells received and store it in the SDRAM. ATM Cell Processor checks the trailer if the error has occurred and informs the result to VR4120A RISC Processor with Rx indication.

**Figure 4-3. AAL-5 Sublayer and ATM Layer**




(a) Padding field: Field of 0 to 47 bytes that is inserted between the user data and the trailer to adjust the length of the resulting packet to a multiple of 48 bytes. Angel writes zeros to all its bits.

(b) CPCS-UU field: Used to transfer user information. The value set in the packet descriptor by the host is written in this field.

(c) CPI field: The use of this field has yet to be finalized. According to the current specifications, all its bits must be set to zeros. ATM Cell Processor, however, writes the value set in the packet descriptor into this field, as it does to the CPCS-UU field.

(d) Packet length (Length) field: Indicates the user data length in bytes, in binary notation.

(e) CRC-32 field: The CRC-32 value calculated for the range from the user data to the end of the Length field is set in this field. Generation polynomial is following

$$G(x) = 1+x+x^2+ x^4+ x^5+ x^7+ x^8+ x^{10}+ x^{11}+ x^{12}+ x^{16}+ x^{22}+ x^{23}+ x^{26}+ x^{32}$$

#### 4.1.3.2 ATM layer function

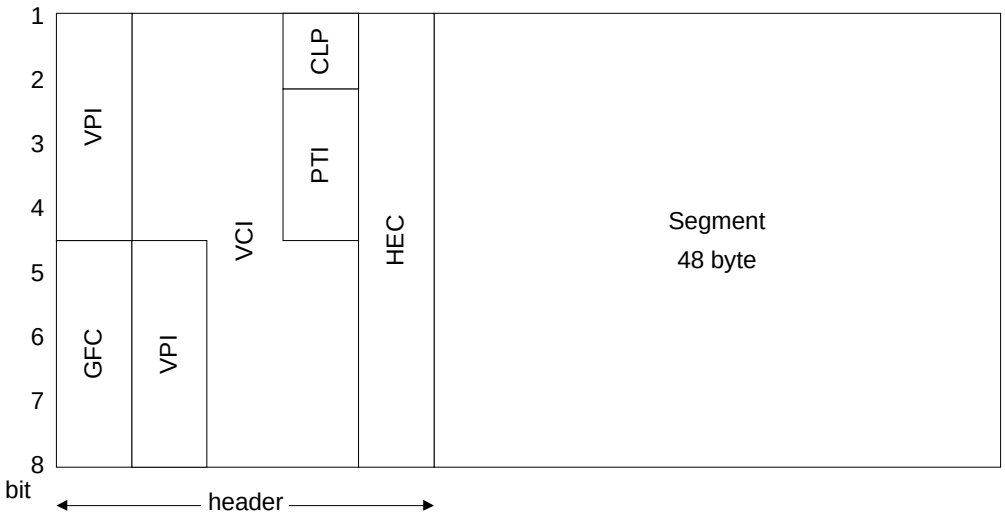
**(1) Traffic classes**

ATM Cell Processor supports 5 traffic classes; CBR (Constant Bit Rate), VBR (Variable Bit Rate) , ABR (Available Bit Rate), GFR (Guaranteed Frame Rate), and UBR (Undefined Bit Rate).

**(2) Generation a cell**

ATM Cell Processor generates a cell by adding 5 bytes header showing bellow to the segment.

**Figure 4-4. ATM Cell**



(a) GFC (General Flow Control) field: Used for flow control. At transmission, the value set in the packet descriptor by the host is written into this field. At reception, this field is ignored, and its contents are not reported to the host.

(b) VPI/VCI fields: VPI(Virtual Path Identifier), VCI(Virtual Channel Identifier) are routing fields which indicate the routing path. ATM Cell Processor supports a total of 64 VPI/VCI combinations for transmission and reception. For transmission, the 24-bit value set in the Tx VC table by the host is written into these fields. For reception, only the VPIs/VCIs registered in the reception lookup table are allowed for reception.

(c) PTI(Payload Type Indication) field: 3-bit field indicating whether the cell payload is user data or management data. It also contains congestion information.

| PTI | usage |
|---|---|
| 000 | user data cell, no congestion, SDU type = 0 |
| 001 | user data cell, no congestion, SDU type = 1 |
| 010 | user data cell, congestion occurred, SDU type = 0 |
| 011 | user data cell, congestion occurred, SDU type = 1 |
| 100 | OAM F5 flow cell |
| 101 | OAM F5 flow cell |
| 110 | RM cell(for ABR service) |
| 111 | reserved for future use |

Here,

SDU type = 0: all segment except last cell of AAL-PDU

SDU type = 1: last cell of AAL-PDU. In this segment, trailer is included.
However, setting SDU type = 1 by user is prohibited.

OAM F5 flow cell: The cells for Operation, Administration and Maintenance.

RM cell: Resource management used in ABR service. Setting PTI=110 by user is prohibited.

(d) CLP(Cell Loss Priority) field: Indicates whether this cell is to be discarded preferentially if the network is congested. A CLP value of 1 indicates that the cell is to be discarded preferentially. ATM Cell Processor sets the appropriate value in this field according to the value that the host sets in the CLPM field of the packet descriptor.

(e) HEC(Header Error Control) field: Used for cell delineation, header error detection and correction. This field is processed in TC sublayer.

**(3) Cell scheduling**

ATM Cell Processor uses Scheduling Table, Cell Timer and Tx VC table for scheduling. Before VR4120A RISC Processor starts transmitting a packet, it sets the rate information in Tx VC table. ATM Cell Processor calculates cell transmission interval from the rate information, and put the next transmission time in Scheduling Table. When the Cell Timer and the next transmission time of certain VC becomes equal, the cell belongs the VC is transmitted.

If the VC is CBR or UBR, only PCR( Peak Cell Rate) is used for scheduling, or if VC is VBR, PCR, SCR( Sustained Cell Rate) and MBS (Maximum Burst Size ) are used. If the VC is ABR, the VR4120A RISC Processor has to set all the ABR parameters ( described in Section. 4.7.6) before packet transmission, and only ACR is used. However, every time ATM Cell Processor receives RM (Resource Management) cell, it recalculates ACR using ABR parameters and overwrites the value to ACR field in Tx VC table.

**(4) Non AAL-5 traffic support**

ATM Cell Processor also handles a cell as a raw cell, in order to support non AAL-5 traffic. For the VC which is set as raw cell mode, ATM Cell Processor doesn't execute any AAL-5 dependent operation, such as calculating CRC-32 and adding trailers, but just generate cells from data and transmits them. In receiving mode, ATM Cell Processor stores cells with header and 11 bytes indication in SDRAM.

ATM Cell Processor has a function of CRC-10 insertion and verification for non AAL-5 traffic. If CRC-10 insertion is set to enable, ATM Cell Processor calculates CRC-10 for each cell and insert it in the end of payload. ATM Cell Processor always verifies CRC-10 in receiving cells. If ATM Cell Processor detects an error, it sets error flag in indication and inform to the VR4120A RISC Processor.

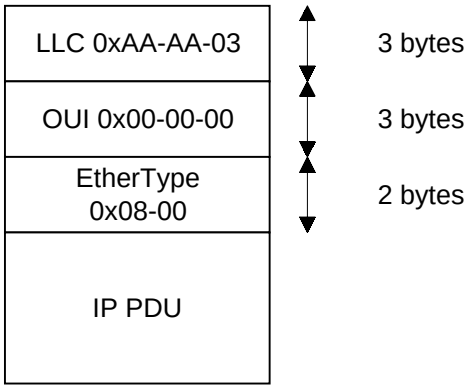
#### 4.1.3.3 IP flow mapping

ATM Cell Processor supports dynamic mapping between IP address and ATM address. VR4120A RISC Processor sets the flow table prior to Tx_Ready command when it sends a packet with new IP address which has not been registered in this Block yet. Flow table contains the parts of TCP/IP header information. VR4120A RISC Processor also reserves the VC table area if it uses new connection and set the parameters in VC table. After that, ATM Cell Processor automatically recognizes the TCP/IP header from the transmitting packet, and chooses the VC corresponding to the IP.

#### 4.1.3.4 LLC encapsulation

If LLC encapsulation mode is set in the VC table, ATM Cell Processor adds the LLC header to the top of the IP packet. In this case, ATM Cell Processor always encapsulates CPCS-PDU as Internet IP PDU. However, if ATM mode Tx_Ready command is used, ATM Cell Processor does not execute encapsulation.

**Figure 4-5. LLC Encapsulation**



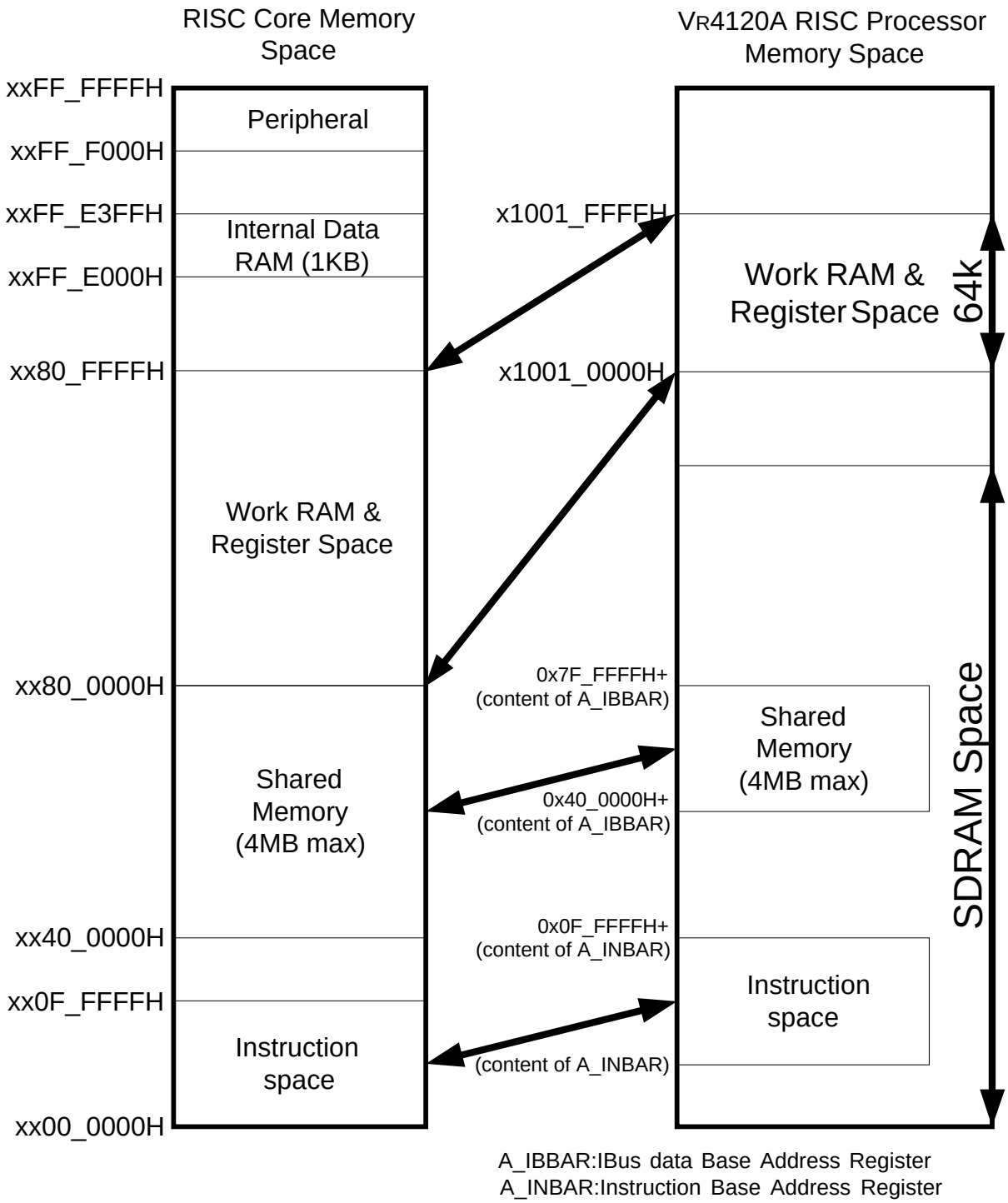

#### 4.1.3.5 PPP encapsulation

If PPP encapsulation mode is set in the VC table, ATM Cell Processor adds the PPP header to the top of the IP packet. ATM Cell Processor encapsulates CPCS-PDU as IP-PDU. There two PPP encapsulation mode, PPP and NT-PPP. ATM Cell Processor adds 0x0021 for PPP mode, or 0xff03-0021 for NT-PPP mode at the top of the CPCS-PDU. However, if ATM mode Tx_Ready command is used, ATM Cell Processor does not execute encapsulation.

## 4.2 Memory Space

Even though RISC Core is a 32-bit MCU, physical memory space of ATM Cell Processor is 24-bit wide.

**Figure 4-6. Memory Space from VR4120A and RISC Core**



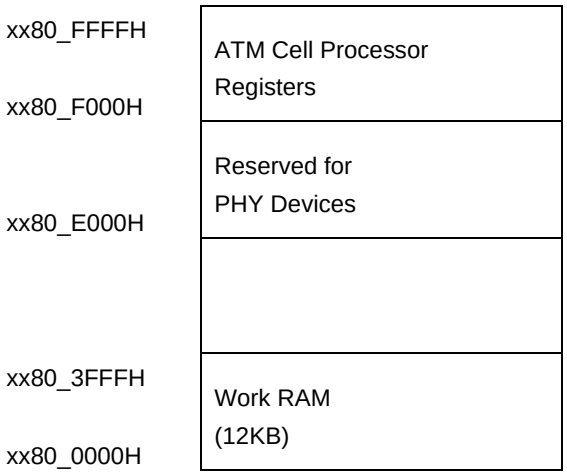

The configuration is shown as Figure 4-6. It contains instruction space, shared memory space, RISC Core space, internal memory space, and peripheral space.

VR4120A and RISC Core share external memory spaces. Shared memory will be implemented by using SDRAM devices. The address in VR4120A RISC Processor memory space will be determined by S/W and notified to RISC Core by setting A_IBBAR (IBus data Base Address Register). Its capacity depends on the total capacity of physical memory, but not exceeds 4MB.

### 4.2.1 Work RAM and register space

Work RAM and Register Space is shown in Figure 4-7. It contains Work RAM, PHY space, and register space. The capacity of Work RAM is 16KB max. It can be accessed by using “Indirect command”. In register space, A_GMR (general mode register), A_GSR (general status register), A_CMR (command register), A_CER (command extension register) and other registers will be mapped. In PHY space, PHY devices can be accessed through UTOPIA management I/F.

**Figure 4-7. Work RAM and Register Space**




Internal memory space and peripheral space are exclusively used by RISC Core and cannot be seen by other blocks. Internal memory space will be used as stack and global variable space. In peripheral space, an interrupt controller and some other special blocks will be mapped. Scheduling table, VC lookup table, IP lookup table, Cell Timer, and ABR rate translator will be mapped in peripheral space as well.

Instruction memory space will be placed in SDRAM space. The address VR4120A RISC Processor memory space will be determined by S/W and notified to RISC Core by setting A_INBAR (Instruction Base Address Register). Instruction memory space is mapped on SDRAM space to achieve faster instruction access.

### 4.2.2 Shard memory

ATM Cell Processor has at most 4MB of buffer memory that is mainly used as cell buffer and packet buffer. This memory will be implemented off the chip, or on SDRAM. VR4120A RISC Processor can access this memory via. System Controller. Address of External Shared Memory will be translated into VR4120A RISC Processor memory space by adding content of A_IBBAR (IBus data Base Address Register). A_IBBAR will be set during initializing period.

## 4.3 Interruption

When it sets a bit in A_GSR (General Status Register) which is NOT masked by the corresponding bit in A_IMR, an interruption is sent out to VR4120A RISC Processor. The status of interruption is obtained by reading in A_GSR. When VR4120A RISC Processor reads A_GSR, the bits which are set and are NOT masked by A_IMR will be reset.

The interruption can be masked by resetting bits of corresponding incidents in A_IMR (Interrupt Mask Register).

Interruptions from PHY devices are forwarded to VR4120A RISC Processor by PI bit of A_GSR automatically. RISC core receives them as well.

## 4.4 Registers for ATM Cell Processing

Registers in this ATM Cell Processor block can be classified into 3 groups: SAR registers, DMA registers and FIFO Control registers. These registers are mapped to both VR4120A RISC Processor memory space and RISC Core memory space.

In the μPD98405, There are two way to access to the registers from host: Direct-Access and Indirect-Access. But, in this block, All registers are mapped to Direct-Access area. Indirect-Access is used only for peripherals, RISC Core internal RAM and Work-RAM

### 4.4.1 Register map

Registers are used for SAR functions. VR4120A RISC Processor writes to these registers to control SAR functions and read from these registers to know the status, while F/W on RISC Core read from these registers to know what VR4120A RISC Processor wants and write to these registers to indicate the status of ATM Cell Processor.

In the next page, indicate the Register Map for this ATM cell Processor.

#### 4.4.1.1 Direct addressing register

(1/2)

| Address[Note] | Register | Description | Read/Write |
|---|---|---|---|
| F000H | A_GMR | General Mode Register | R/W |
| F004H | A_GSR | General Status Register | R |
| F008H | A_IMR | Interrupt Mask Register | R/W |
| F00CH | A_RQU | Receive Queue Underrunning | R |
| F010H | A_RQA | Receive Queue Alert | R |
| F014H | N/A | Reserved for future use | X |
| F018H | A_VER | Version Number | R |
| F01CH | N/A | Reserved for future use | - |
| F020H | A_CMR | Command Register | R/W |
| F024H | N/A | Reserved for future use | - |
| F028H | A_CER | Command Extension Register | R/W |
| F02CH-F04CH | N/A | Reserved for future use | - |
| F050H | A_MSA0 | Mailbox0 Start Address | R/W |
| F054H | A_MSA1 | Mailbox1 Start Address | R/W |
| F058H | A_MSA2 | Mailbox2 Start Address | R/W |
| F05CH | A_MSA3 | Mailbox3 Start Address | R/W |
| F060H | A_MBA0 | Mailbox0 Bottom Address | R/W |
| F064H | A_MBA1 | Mailbox1 Bottom Address | R/W |

**Note** VR4120A RISC Processor memory space, add 1001_0000H as Base Address.

(2/2)

| Address[Note] | Register | Description | Read/Write |
|---|---|---|---|
| F068H | A_MBA2 | Mailbox2 Bottom Address | R/W |
| F06CH | A_MBA3 | Mailbox3 Bottom Address | R/W |
| F070H | A_MTA0 | Mailbox0 Tail Address | R/W |
| F074H | A_MTA1 | Mailbox1 Tail Address | R/W |
| F078H | A_MTA2 | Mailbox2 Tail Address | R/W |
| F07CH | A_MTA3 | Mailbox3 Tail Address | R/W |
| F080H | A_MWA0 | Mailbox0 Write Address | R/W |
| F084H | A_MWA1 | Mailbox1 Write Address | R/W |
| F088H | A_MWA2 | Mailbox2 Write Address | R/W |
| F08CH | A_MWA3 | Mailbox3 Write Address | R/W |
| F090H | A_RCC | Valid Receiving Cell Counter | R |
| F094H | A_TCC | Valid Transmitting Cell Counter | R |
| F098H | A_RUEC | Receive Unprovisioned VPI/VCI Error Cell Counter | R |
| F09CH | A_RIDC | Receiving Internal Discarded Cell Counter | R |
| F0A0H-F0ACH | N/A | Reserved for future use | - |
| F0B0H | A_APR | ABR Parameter Register | R/W |
| F0B4H-F0BCH | N/A | Reserved for future use | |
| F0C0H | A_T1R | T1 Timer Register | R/W |
| F0C4H | N/A | Reserved for future use | - |
| F0C8H | A_TSR | Time Stamp Register | R/W |
| F200H-F2FFH | N/A | Can not access from VR4120A RISC Core.<br>These register use only internal DMAC. | - |
| F300H | A_IBBAR | IBUS Base Address Register | R/W |
| F304H | A_INBAR | Instruction Base Address Register | R/W |
| F308H- F31FH | N/A | Reserved for future use | - |
| F320H | A_UMCMD | UTOPIA Management Interface Command Register | R/W |
| F324H- F3FFH | N/A | Reserved for future use | - |
| F400H-F4FFH | N/A | Can not access from VR4120A RISC Core.<br>These register use only internal SAR FIFO. | - |
| F500H-FFFFH | N/A | Reserved for future use | - |

**Note** VR4120A RISC Processor memory space, add 1001_0000H as Base Address.

#### 4.4.1.2 Indirect addressing register

| Address[Note1] | Register | Description | Read/Write |
|---|---|---|---|
| FFF410H | RXLCTR | Rx Lookup Table control register | R/W |
| FFF600H | RxTBL000 | Rx Lookup Table Entry00 Halfword0 | W |
| FFF602H | RxTBL001 | Rx Lookup Table Entry00 Halfword1 | W |
| | | | |
| FFF6FCH | RxTBL3F0 | Rx Lookup Table Entry3F Halfword0 | W |
| FFF6FEH | RxTBL3F1 | Rx Lookup Table Entry3F Halfword1 | W |
| FFF700H | RxTBC00 | Rx Lookup Table Control Entry00 | R/W |
| | | | |
| FFF77EH | RXTBC3F | Rx Lookup Table Control Entry3F | R/W |

**Note** The value of the address field of indirect address command.

### 4.4.2 A_GMR (General Mode Register)

A_GMR selects operation mode of this block, enables/disables ATM SAR operations. After reset, VR4120A RISC Processor must write this register first to configure. Modification of A_GMR after starting Tx/Rx operations is prohibited. All bits of this register are writable, but the bit 31-15, 13-3 are reserved for future use. Initial value is all zero.

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| 0 | LP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | TE | RE |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

| Field | Value | Meaning | Initial value |
|---|---|---|---|
| LP | 1 | Loopback is performed at the ATM interface | 0 |
| | 0 | Loopback is not performed at the ATM interface | |
| TE | 1 | Transmit enable | 0 |
| | 0 | Transmit disable | |
| RE | 1 | Receive enable | 0 |
| | 0 | Receive disable | |

### 4.4.3 A_GSR (General Status Register)

A_GSR shows interruption status. When an incident which triggers interruption occurs, F/W on RISC Core set a bit in A_GSR corresponds to the type of incident. If Corresponding bit in A_IMR(Interrupt Mask Register) is reset and the interruption is not masked, UTOPIA Bus Controller interrupt to VR4120A RISC Processor using interrupt signal. The bit in A_GSR is reset after being read by the VR4120A RISC Processor. When the same type of incidents occurs before the bit has been read, the bit will be set again.

All bits of this register is writable by RISC Core, but the bit 27-25, 22, 20-16 are reserved for future use.
Initial value is all zero.

| PI | RQA | RQU | RD | 0 | 0 | 0 | IND | SQO | 0 | FER | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| RCR[7:0] | | | | | | | | MF[3:0] | | | | MM[3:0] | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

(1/2)

| Field | Value | Meaning | Initial value |
|---|---|---|---|
| PI | 1 | PHY layer device interruption has occurred | 0 |
| | 0 | PHY layer device interruption has not occurred | |
| RQA | 1 | Receive Queue alert has occurred. | 0 |
| | 0 | Receive Queue alert has not occurred. | |
| RQU | 1 | Receive Queue underflow has occurred. | 0 |
| | 0 | Receive Queue underflow has not occurred. | |
| RD | 1 | Receive Deactivation has occurred. | 0 |
| | 0 | Receive Deactivation has not occurred. | |
| IND | 1 | Initialization has completed. | 0 |
| | 0 | Initialization has not completed. | |
| SQO | 1 | Scheduling Queue overflow has occurred. | 0 |
| | 0 | Scheduling Queue overflow has not occurred. | |
| FER | 1 | Fatal Error has occurred. | 0 |
| | 0 | Fatal Error has not occurred. | |

(2/2)

| Field | Value | Meaning | Initial value |
|---|---|---|---|
| RCR[7:0] | 1xxxxxxx | Raw cell is in Pool No.7. | 00000000 |
| | 0xxxxxxx | Raw cell is not in Pool No.7. | |
| | x1xxxxxx | Raw cell is in Pool No.6. | |
| | x0xxxxxx | Raw cell is not in Pool No.6. | |
| | xx1xxxxx | Raw cell is in Pool No.5. | |
| | xx0xxxxx | Raw cell is not in Pool No.5. | |
| | xxx1xxxx | Raw cell is in Pool No.4. | |
| | xxx0xxxx | Raw cell is not in Pool No.4. | |
| | xxxx1xxx | Raw cell is in Pool No.3. | |
| | xxxx0xxx | Raw cell is not in Pool No.3. | |
| | xxxxx1xx | Raw cell is in Pool No.2. | |
| | xxxxx0xx | Raw cell is not in Pool No.2. | |
| | xxxxxx1x | Raw cell is in Pool No.1. | |
| | xxxxxx0x | Raw cell is not in Pool No.1. | |
| | xxxxxxx1 | Raw cell is in Pool No.0. | |
| | xxxxxxx0 | Raw cell is not in Pool No.0. | |
| MF[3:0] | 1xxx | Mailbox No.3 is full. | 0000 |
| | 0xxx | Mailbox No.3 is not full. | |
| | x1xx | Mailbox No.2 is full. | |
| | x0xx | Mailbox No.2 is not full. | |
| | xx1x | Mailbox No.1 is full. | |
| | xx0x | Mailbox No.1 is not full. | |
| | xxx1 | Mailbox No.0 is full. | |
| | xxx0 | Mailbox No.0 is not full. | |
| MM[3:0] | 1xxx | Mailbox No.3 is marked. | 0000 |
| | 0xxx | Mailbox No.3 is not marked. | |
| | x1xx | Mailbox No.2 is marked. | |
| | x0xx | Mailbox No.2 is not marked. | |
| | xx1x | Mailbox No.1 is marked. | |
| | xx0x | Mailbox No.1 is not marked. | |
| | xxx1 | Mailbox No.0 is marked. | |
| | xxx0 | Mailbox No.0 is not marked. | |

### 4.4.4 A_IMR (Interrupt Mask Register)

A_IMR masks interruption for each corresponding incident. A Mask bit, which locates in the same bit location to a corresponding bit in A_GSR, controls interruption triggered by the incident. If a bit of this register is reset to 0, the corresponding bit of the A_GSR is masked. If it is set to 1, the corresponding bit is unmasked. When the mask bit is reset and the bit in A_GSR is set, UTOPIA Bus Controller set interruption signal to interrupt VR4120A RISC Processor.

All bits of this register is writable, but the bit 27-25, 22, 20-16 are reserved for future use.

Initial value is all zero.

| PI | RQA | RQU | RD | 0 | 0 | 0 | IND | SQO | 0 | FER | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| RCR[7:0] | | | | | | | | MF[3:0] | | | | MM[3:0] | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

| Field | Value | Meaning | Initial value |
|---|---|---|---|
| PI | 0(mask)<br>1(unmask) | Mask bit for PHY layer device interruption | 0 |
| RQA | 0(mask)<br>1(unmask) | Mask bit for Receive Queue Alert | 0 |
| RQU | 0(mask)<br>1(unmask) | Mask bit for Receive Queue underflow | 0 |
| RD | 0(mask)<br>1(unmask) | Mask bit for Receive Deactivation status | 0 |
| IND | 0(mask)<br>1(unmask) | Mask bit for Initialization status | 0 |
| SQO | 0(mask)<br>1(unmask) | Mask bit for Scheduling Queue overflow | 0 |
| FER | 0(mask<br>1(unmask) | Mask bit for Fatal Error | 0 |
| RCR[7:0] | 0(mask)<br>1(unmask) | Mask bit for Raw cell reception | |
| MF[3:0] | 0(mask)<br>1(unmask) | Mask bit for Mailbox full | 0000 |
| MM[3:0] | 0(mask)<br>1(unmask) | Mask bit for Mailbox mark | 0000 |

### 4.4.5 A_RQU (Receiving Queue Underrun Register)

A_RQU shows pools which has no free buffers. When a pool has no free buffers, sets a bit which corresponds to the pool. ATM Cell Processor detects vacancy of a pool when it receives a cell and try to send the cell to buffer. Whenever one of A_RQU bits is on, A_RQU bit in A_GSR will be set. In this block, only pool7 - pool0 will be used. Bit31-bit0 corresponds to pool31-pool0 respectively. If a bit is set to “1”, corresponding pool has no free buffers. Initial value is all zero.

| A_RQU[31:0] |
|---|

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

### 4.4.6 A_RQA (Receiving Queue Alert Register)

A_RQA shows pools where the number of remaining batches is less than “ALERT LEVEL”, which is set by a user. Whenever one of A_RQA bits is on, A_RQA bit in A_GSR will be set. In this block, only pool7 - pool0 will be used. Bit31-bit0 corresponds to pool31-pool0 respectively. If a bit is set to “1”, the number of remaining batches is less than “ALERT LEVEL”. Initial value is all zero.

| A_RQA[31:0] |
|---|

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

### 4.4.7 A_VER (Version Register)

A_VER shows version number of ATM Cell Processor block. Initial value is 00000200H.

| ALL 0 | MAJOR REVISION[7:0] | MINOR REVISION[7:0] |
|---|---|---|

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

### 4.4.8 A_CMR (Command Register)

ATM Cell Processor receives command and parameter when VR4120A RISC Processor writes them in A_CMR and A_CER. It can handle only one command at a time. When ATM Cell Processor receives a command from VR4120A RISC Processor, it sets Busy Flag automatically by H/W to indicate it is busy. While Busy Flag is set, ATM Cell Processor can not receive a new command. If VR4120A RISC Processor writes a new command when Busy Flag is on, the new command will be ignored. F/W must reset Busy Flag by write-access when the command execution has been completed. Multi-host mode is not supported. Initial value is zero. Detail of this register is described in **Section 4.7 Commands**.

| BSY | A_CMR[30:0] |
|---|---|

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A_CMR[30:0](continue) | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

### 4.4.9 A_CER (Command Extension Register)

Command Extension Register. Initial value is zero. Detail of this register is described in **Section 4.7 Commands**.

| A_CER[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

### 4.4.10 A_MSA0-A_MSA3 (Mailbox Start Address Register)

A_MSA0-A_MSA3 shows start address of Receive Mailbox (Mailbox0 and Mailbox1) and Transmit Mailbox (Mailbox2 and Mailbox3) respectively. Initial value is all zero.

| A_MSA0[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| A_MSA1[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| A_MSA2[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| A_MSA3[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

### 4.4.11 A_MBA0-A_MBA3 (Mailbox Bottom Address Register)

A_MBA0-A_MBA3 shows bottom address of Receive Mailbox (Mailbox0 and Mailbox1) and Transmit mailbox (Mailbox2 and Mailbox3) respectively. Initial value is all zero.

| A_MBA0[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| A_MBA1[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| A_MBA2[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| A_MBA3[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

### 4.4.12 A_MTA0-A_MTA3 (Mailbox Tail Address Register)

A_MTA0-A_MTA3 shows tail address of Receive Mailbox (Mailbox0 and Mailbox1) and Transmit Mailbox (Mailbox2 and Mailbox3) respectively. Initial value is zero.

| A_MTA0[31:0] |
|---|
| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
| A_MTA1[31:0] |
| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
| A_MTA2[31:0] |
| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
| A_MTA3[31:0] |
| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |

### 4.4.13 A_MWA0-A_MWA3 (Mailbox Write Address Register)

A_MWA0-A_MWA3 shows write address of Receive Mailbox (Mailbox0 and Mailbox1) and Transmit Mailbox (Mailbox2 and Mailbox3) respectively. Initial value is zero.

| A_MWA0[31:0] |
|---|
| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
| A_MWA1[31:0] |
| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
| A_MWA2[31:0] |
| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
| A_MWA3[31:0] |
| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |

### 4.4.14 A_RCC (Valid Receive Cell Counter Register)

A_RCC counts the number of valid Receive cells. It is a 32-bit free-run counter. Overflow of this counter does NOT cause any interruption. Initial value is zero.

| A_RCC[31:0] |
|---|
| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |

### 4.4.15 A_TCC (Valid Transmit Cell Counter Register)

A_TCC counts the number of valid Transmit cells. It is a 32-bit free-run counter. Overflow of this counter does NOT cause any interruption. Initial value is zero.

| A_TCC[31:0] |
|---|
| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |

#### 4.4.16 A_RUEC (Receive Unprovisioned VPI/VCI Error Cell Counter Register)

A_RUEC counts the number of Receive cells with VPI/VCI Error. It is a 32-bit free-run counter. Overflow of this counter does NOT cause any interruption. Initial value is zero.

| A_RUEC[31:0] |
|---|

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

#### 4.4.17 A_RIDC (Receive Internal Dropped Cell Counter Register)

A_RIDC counts the number of Receive cells which are dropped inside. It is a 32-bit free-run counter. Overflow of this counter does NOT cause any interruption. Initial value is zero.

| A_RIDC[31:0] |
|---|

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

#### 4.4.18 A_APR (ABR Parameter Register)

A_APR contains the parameters needed for ABR parameter calculation. Initial value is 0x0H.

| Trm0 | CRM | 0 | Mrm | Nrm0 |
|---|---|---|---|---|

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| Field | Meaning | Units |
|---|---|---|
| Trm0 | Trm0 provides an upper bound on the time between forward RM-cells for an active source. | Cells |
| CRM | Missing RM-cell count. CRM limits the number of forward RM-cells which may be sent in the absence of received backward RM-cells. | Cells |
| Mrm | Mrm controls allocation of bandwidth between forward RM-cells, backward RM-cells and data cells. Mrm should be set to 2. | integer |
| Nrm0 | Nrm is the maximum number of cells a source may send for each forward RM-cells. Nrm=$2x2^{Nrm0}$ | integer |

#### 4.4.19 A_T1R (T1 Time Register)

A_T1R is a free-run hardware counter. A_T1R shows time which user allows ATM Cell Processor to spend to receive one packet. A_T1R contains MSB 16 bits of a 32-bit data which counts system clock. Initial value is “0000FFFFH”.

| All 0 | A_T1R[15:0] |
|---|---|

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

#### 4.4.20 A_TSR (Time Stamp Register)

A_TSR is a 32-bit counter which ATM Cell Processor counts system clock. It is used as a time stamp of Receive start time of T1 timer function. Initial value is “00000000”, but count up starts right after reset.

| A_TSR[31:0] |
|---|

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

### 4.4.21 A_IBBAR (IBUS Base Address Register)

A_IBBAR contains the base address for the access thorough IBUS to outside. RISC Core-space is addressed using 24-bit address, while VR4120A RISC Processor space is addressed using 32-bit address. Therefore, the extension of address is necessary when the access from the inside of this block to the outside is requested. Initial value is zero.

| A_IBBAR[31:0] |
|---|

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

### 4.4.22 A_INBAR (Instruction Base Address register)

A_INBAR contains the base address to fetch instructions. RISC Core-space is addressed using 24-bit address, while VR4120A RISC Processor space is addressed using 32-bit address. Therefore, the extension of address is necessary when the access from the inside to the outside is requested. Initial value is zero.

| A_INBAR[31:0] |
|---|

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

### 4.4.23 A_UMCMD (UTOPIA Management I/F Command Register)

A_UMCMD selects operation mode of UTOPIA Management I/F and enables Monitoring mode to show RISC Core trace data to the outside of LSI. After reset, RISC Core must write this register to configure UTOPIA Management I/F. Initial value is zero.

| NM | All 0 |
|---|---|

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| All 0 | PR | MSL[1:0] |
|---|---|---|

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| Field | Value | Meaning | Initial value |
|---|---|---|---|
| NM | 0 | RISC Core monitoring disable | 0 |
| | 1 | RISC Core monitoring enable | |
| PR | 0 | To deassert UMRSTB | 0 |
| | 1 | To assert UMRSTB to reset PHY device | |
| MSL[1:0] | 1x | Reserved | 00 |
| | 01 | UTOPIA Management acts in the Intel-compatible mode. (RD,WR,RDY style) | |
| | 00 | UTOPIA Management acts in the Motorola-compatible mode. (DS, R/W, DTACK style) | |

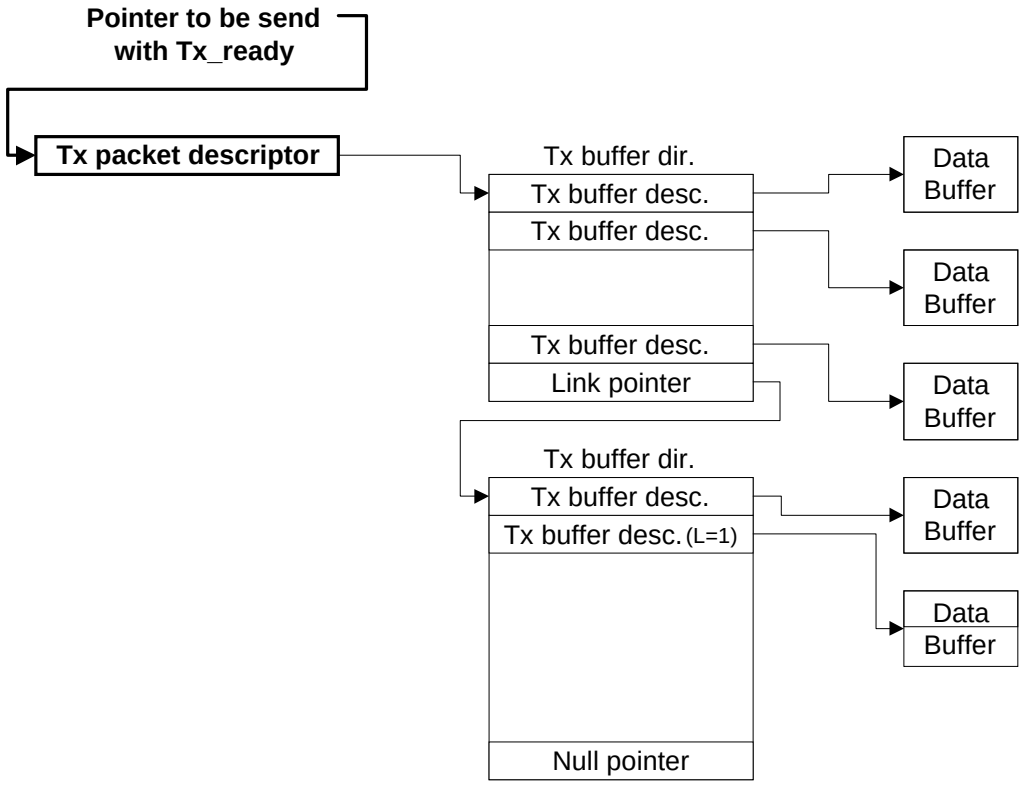
## 4.5 Data Structure

ATM Cell Processor has Tx/Rx buffer structure similar to that of the μPD98401A/μPD98405. But some names of descriptors and fields are different from the way it called in the μPD98401A/μPD98405. Basically Ethernet Controller and USB Controller will have the same buffer structure.

### 4.5.1 Tx packet

Tx Packet consists of a packet descriptor, some buffer directories, and data buffers. A received packet and a Tx packet have similar structures, so that reconstructing buffer structure is not needed when sending out a received packet.

**Figure 4-8. Tx Packet**




This block does not have Tx packet queue used in the μPD98401A/μPD98405, and requires “Tx_ready” command for each Tx packet.

**Figure 4-9. Tx Buffer Elements**

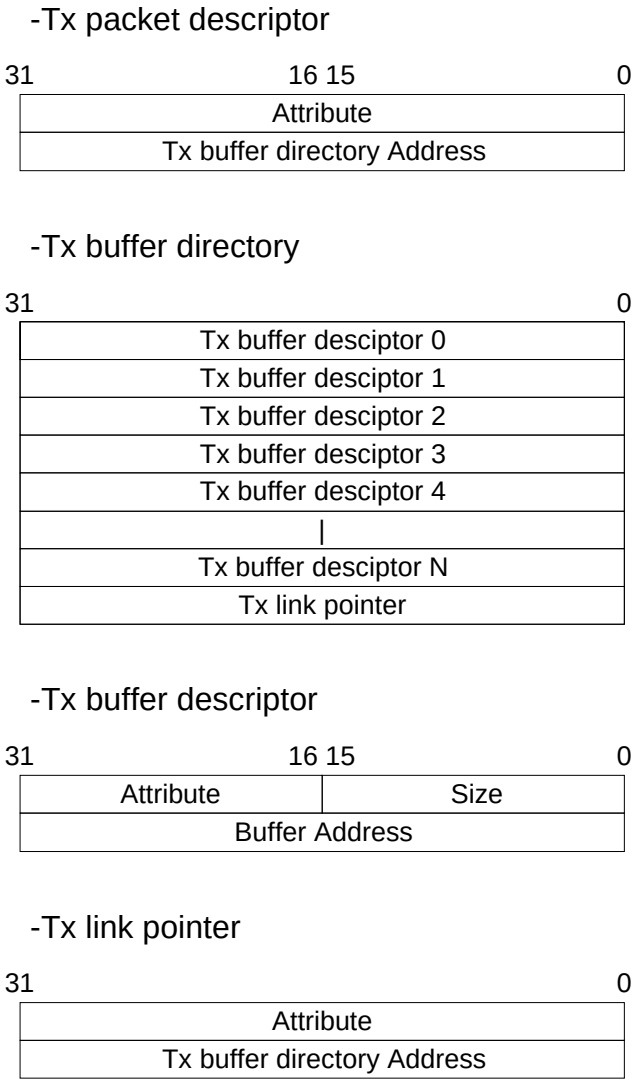



Figure 4-9 shows Tx buffer elements. Each element consists of a couple of 32-bit words of sequential address. Detail is given in following sections.

#### 4.5.1.1 Packet descriptor

A packet descriptor contains two words shown as Figure 4-10. Its address is word aligned.

**Figure 4-10. Tx Packet Descriptor**

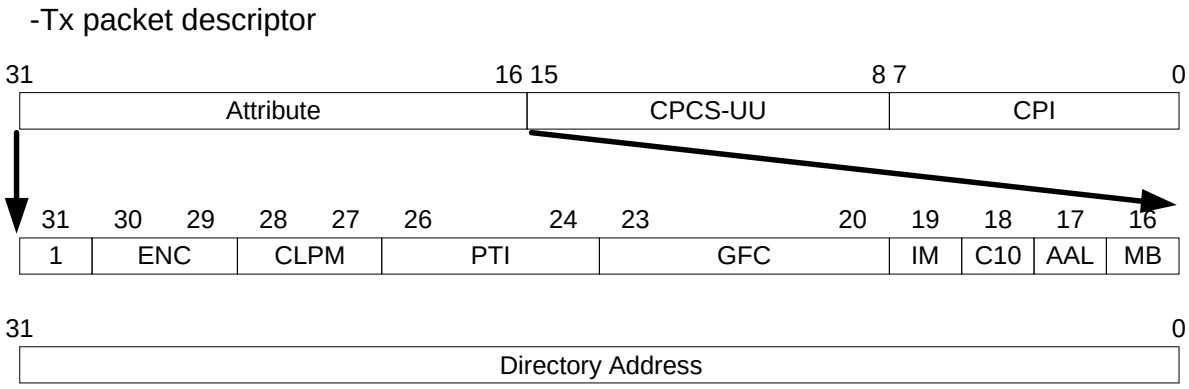



Table. 4-1 is a list of Tx packet attributes. Detail will be given in Operation chapter.

**Table 4-1. List of Tx Packet attribute**

| Field | Description |
|---|---|
| ENC | It contains Encapsulation bits to specify PPP Encapsulation modes. |
| CLPM | It contains CLP bit to be set in Tx cell header. |
| PTI | It contains PTI bit to be set in Tx cell header. |
| GFC | It contains GFC bit to be set in Tx cell header. |
| IM | It disables interruption and indication when Tx is completed. |
| C10 | It enables generation and insertion of CRC10 code. |
| AAL | It specifies type of AAL. |
| MB | It indicates the number of mailbox. |
| CPCS-UU | It contains CPCS-UU bits to be set in Tx cell trailer. |
| CPI | It contains CPI bits to be set in Tx cell trailer. |

#### 4.5.1.2 Tx buffer directory

Tx buffer directory contains some buffer descriptors, not more than 255, and a link pointer. Its address is word aligned. The end of buffer directory must be a link pointer. Buffer descriptors must be read and served from the top in a sequential manner.

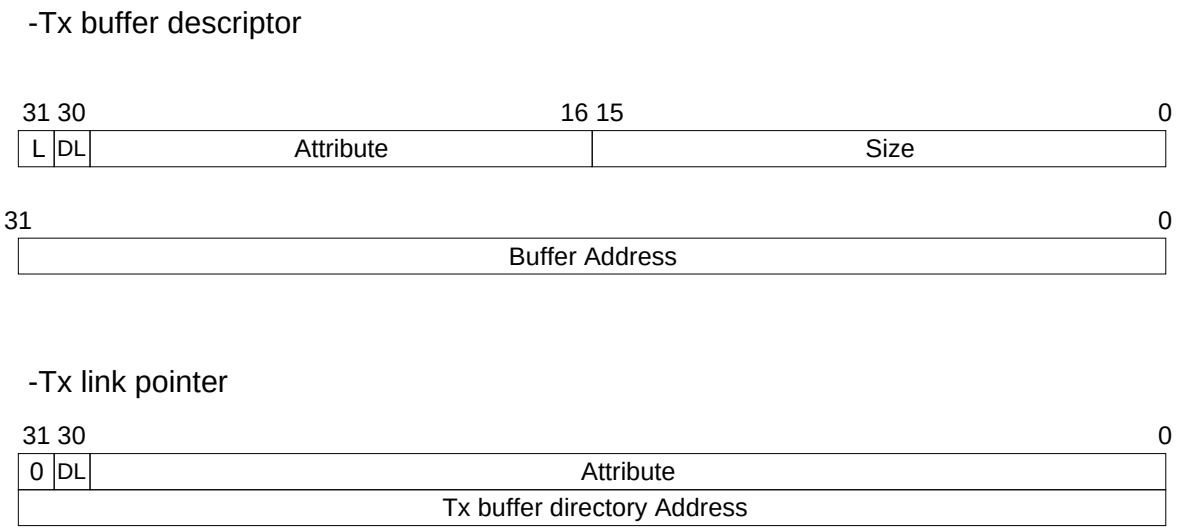
#### 4.5.1.3 Tx buffer descriptor

Both a Tx buffer descriptor and a Tx link pointer consist of 2 words. DL bit, bit 30 of the first word, indicates that these two words are a buffer descriptor (DL=1) or a link pointer (DL=0).

A Tx buffer descriptor is shown as Figure 4-11. Its address is word aligned. L bit, bit 31, indicates that the buffer pointed by this descriptor contains the last potion of a packet.

A Tx link pointer is shown as Figure 4-11. L bit, bit 31, is fixed to zero. If there is no buffer directory to be linked, directory address of link pointer must be zero, as a null pointer.

**Figure 4-11. Tx Buffer Descriptor / Link Pointer**




#### 4.5.1.4 Data buffer

Data buffer contains actual cell data and packet data. Size of a buffer can vary from 1 byte to 64k bytes. Its address is byte aligned.

### 4.5.2 Rx pool structure

Rx buffer is called pool. This block has eight pools. A pool has a chain of Rx buffer directories, which is used to be called “batch”. Each Rx buffer directory has some buffer descriptors and a link pointer. Each buffer descriptor points an actual buffer. A link pointer has an address to a next Rx buffer directory.

VR4120A RISC Processor will create up to 8 pools and give them to ATM Cell Processor.

**Figure 4-12. Rx Pool**

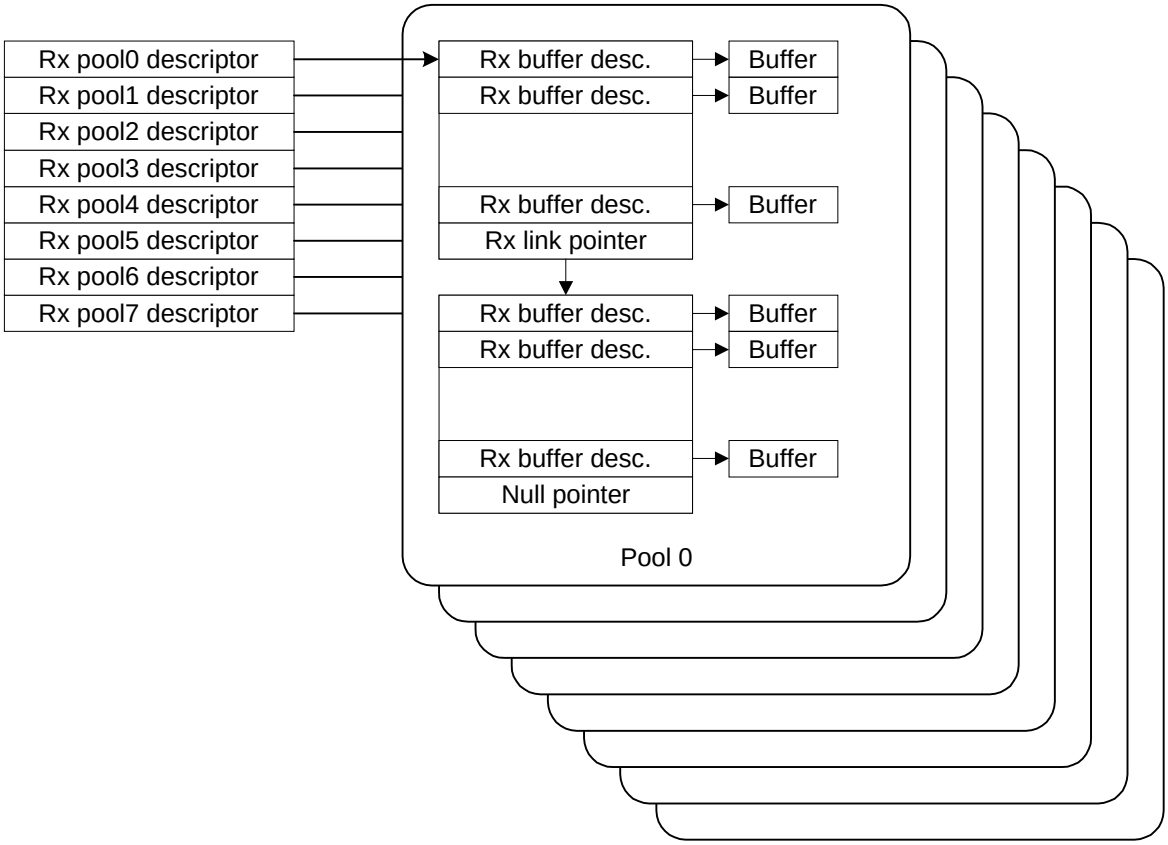


**Figure 4-13. Rx Pool Structure**

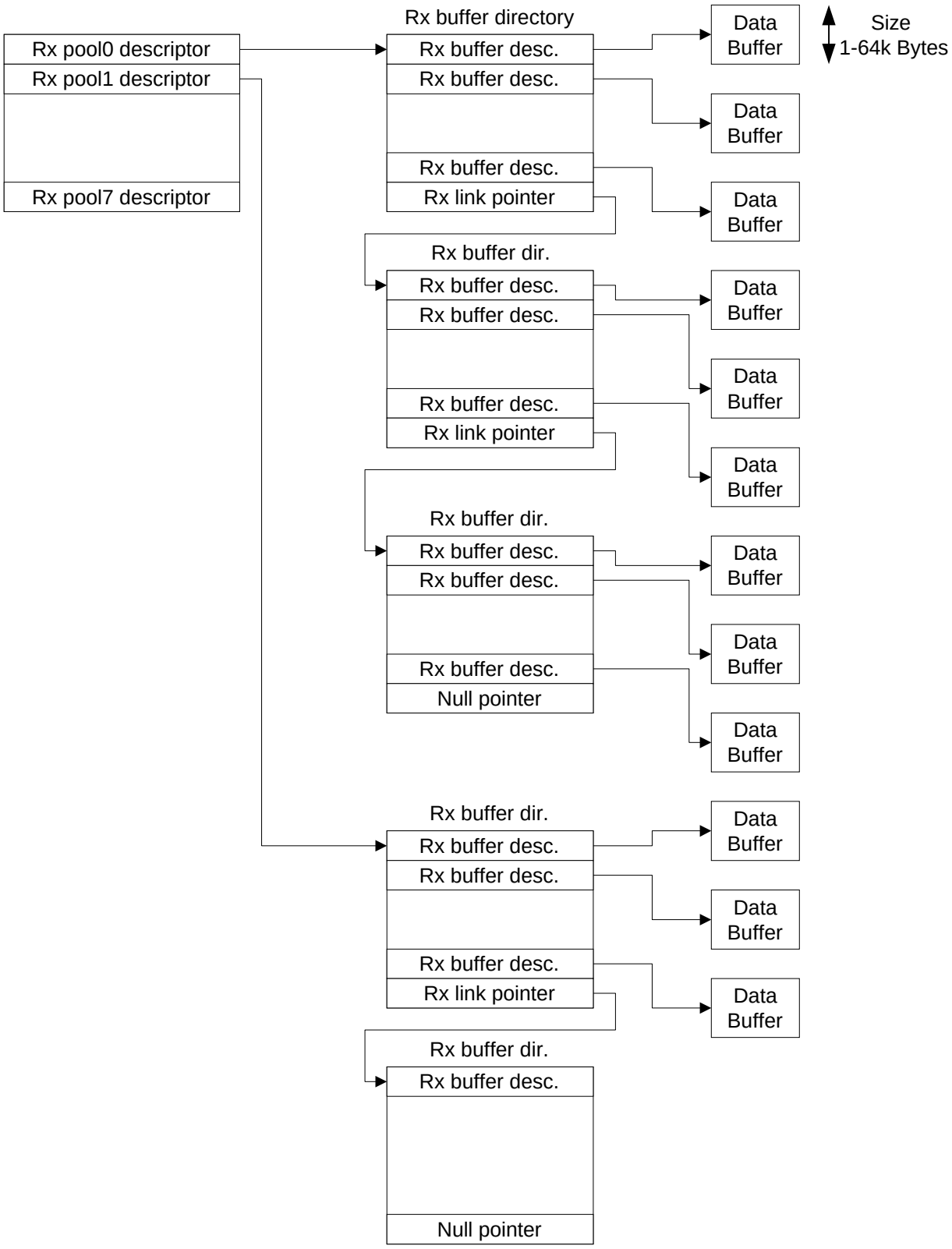


**Figure 4-14. Rx Pool Elements**

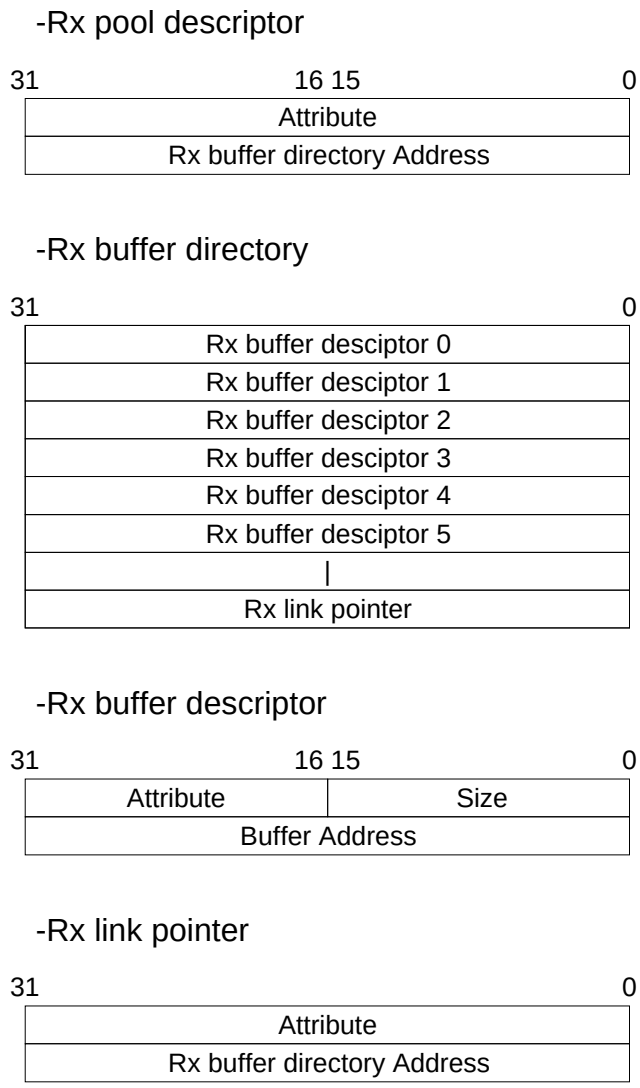



Figure 4-14 shows Rx buffer elements. Each element consists of a couple of 32-bit words of sequential address. Detail is given in following sections.

#### 4.5.2.1 Rx pool descriptor

A pool descriptor contains two words shown as Figure 4-15. Its address is word aligned.

**Figure 4-15. Rx Pool Descriptor**

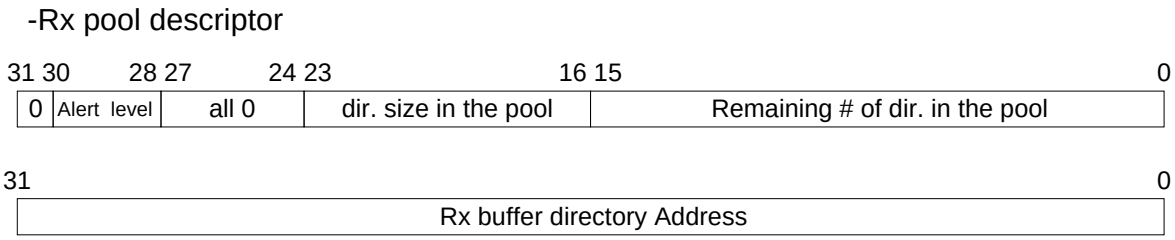



**Table 4-2. List of Rx Pool Attributes**

| Field | Note |
|---|---|
| Alert level | When the remaining number of buffer directories is lower than this number times 4, an alert interrupt will occur. |
| Dir. size in the pool | It contains the number of Rx buffer descriptors in a buffer directories in the pool. |
| Remaining number of dir. in the pool | It contains the current remaining number of buffer directories. |
| Directory Address | It contains address of the first buffer directory in the pool. |

#### 4.5.2.2 Rx buffer directory

Rx buffer directory contains some buffer descriptors, not more than 255, and a link pointer. Number of buffer descriptors in directories in one pool is identical. Number can vary in different pools.

Address of buffer directory is word aligned. The end of buffer directory must be a link pointer. Buffer descriptor must be read and used from the top in a sequential manner.

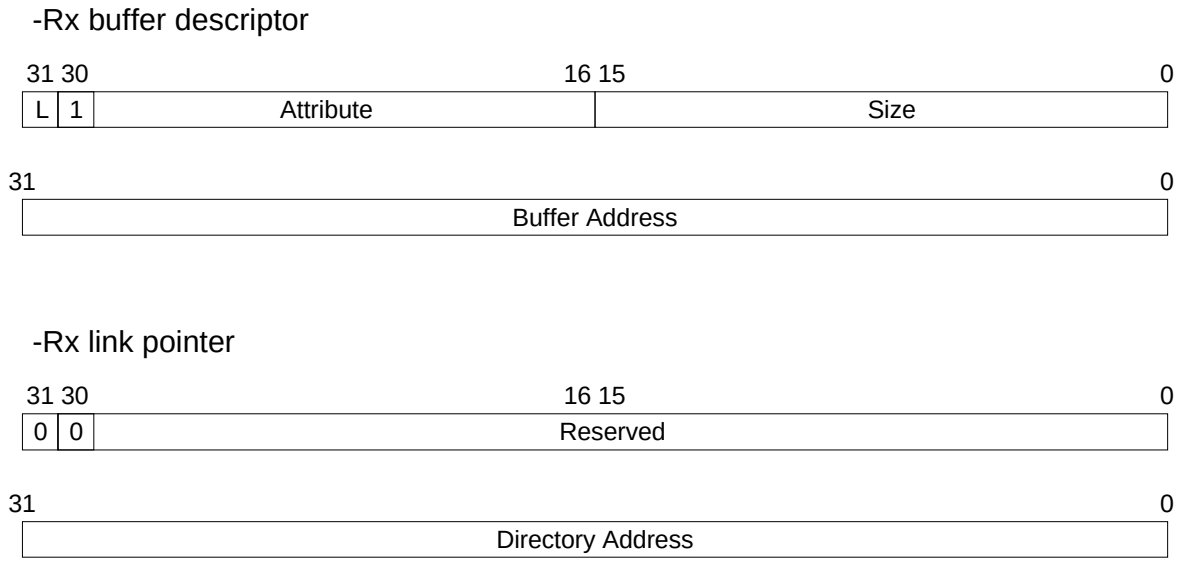
#### 4.5.2.3 Rx buffer descriptor

Both a Rx buffer descriptor and a Rx link pointer consist of 2 words. DL bit, bit 30 of the first word, indicates that these two words are a buffer descriptor (DL=1) or a link pointer (DL=0).

A Rx buffer descriptor is shown as Figure 4-16. Its address is word aligned. L bit indicates that the buffer pointed by this descriptor contains the last potion of a packet.

A Rx link pointer is shown as Figure 4-16. If there is no buffer directory to be linked, directory address of link pointer must be zero, as a null pointer.

**Figure 4-16. Rx Buffer Descriptor/ Link Pointer**




#### 4.5.2.4 Rx data buffer

Rx Data buffer contains actual cell data and packet data. Size of a buffer can vary from 1 byte to 64k bytes. Its address is byte aligned.

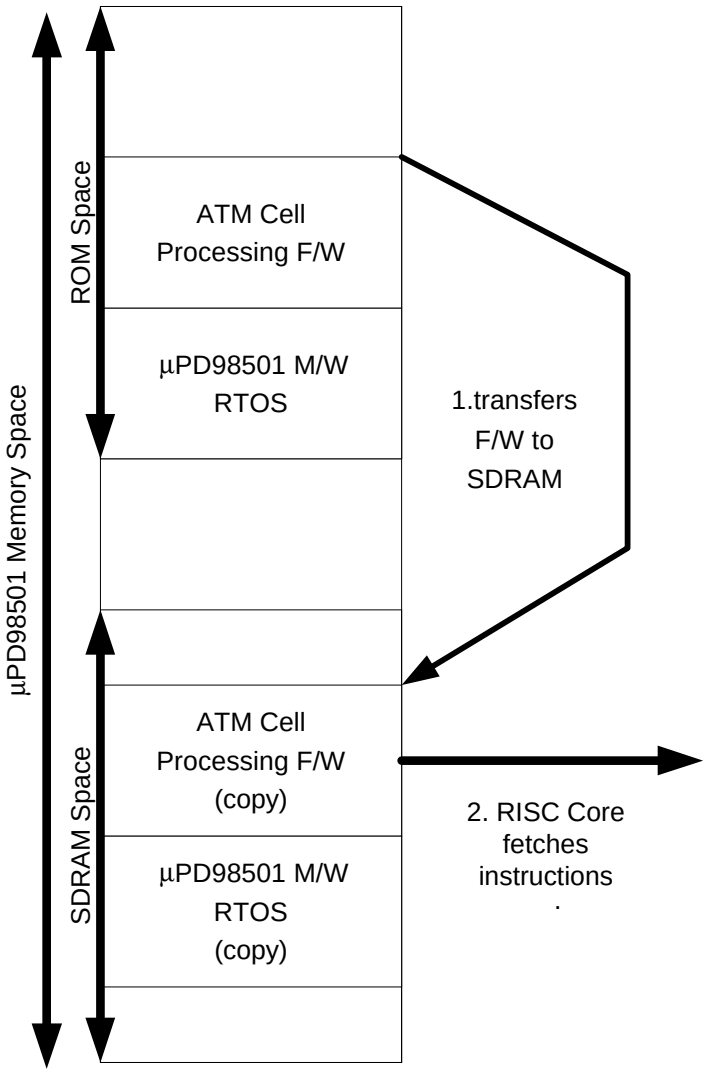
## 4.6 Initialize

This ATM Cell Processor is initialized by Firm Ware that is based RISC instruction.

### 4.6.1 Before starting RISC core

RISC Core has 1MB of Instruction space and 8KB of physical Instruction RAM and 8KB of instruction cache. The Instruction space will be mapped to VR4120A RISC Processor Memory Space. Address of instruction space will be translated into VR4120A RISC Processor memory space by adding content of A_INBAR. VR4120A RISC Processor will set A_INBAR during initializing period. Instruction memory space will be placed in SDRAM space to achieve faster instruction fetch. VR4120A RISC Processor has to transfer RISC Core F/W to the assigned SDRAM area as well as its own S/W in its initializing period, as shown in Figure 4-17. After transferring F/W, it sets base address of RISC Core F/W in A_INBAR. At the same time, base address of shared memory space has to be set in A_IBBAR. Then VR4120A RISC Processor let ATM Cell Processor to start operation.

**Figure 4-17. Transfer of F/W**

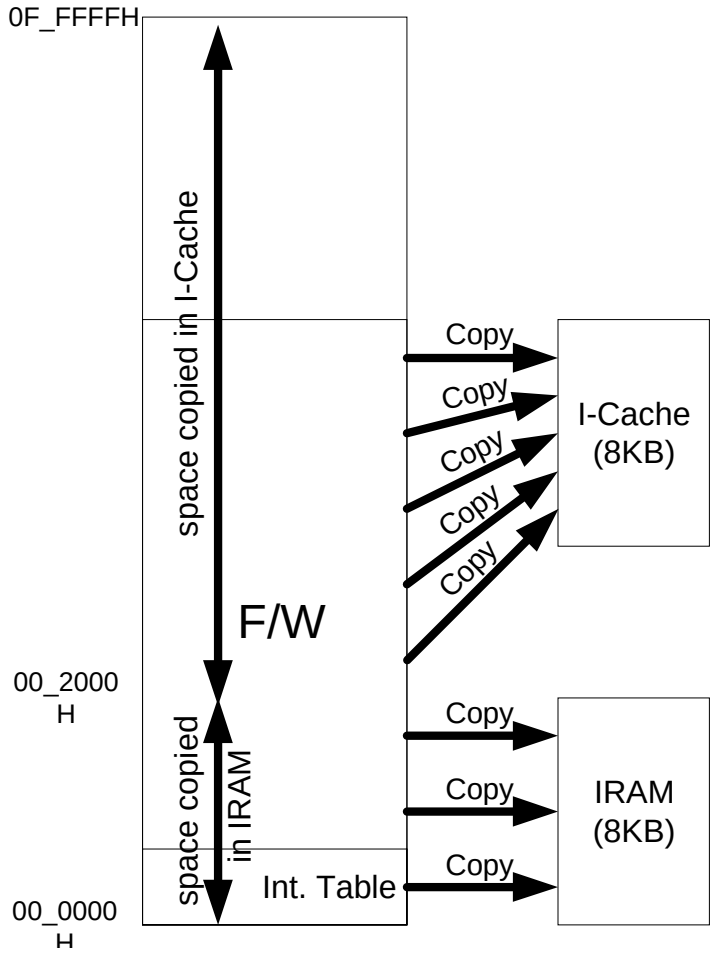


### 4.6.2 After RISC core's F/W is starting

RISC Core starts its operation from address xx00_0000H. When it starts fetching an instruction located in address xx00_0000H, a dedicated H/W will stop RISC Core and will copy a block of instructions. This copy operation will be handled in the same manner as I-cache replacement.

Lower 8K bytes of Instruction space in RISC Core will be copied on Instruction RAM because it contains interruption vector table. The other part of the space is accessed through 8K bytes of Instruction cache. In case that the total size of F/W is smaller than 16K bytes, RISC Core can run fastest because once all necessary instruction code is copied in, no cache miss will occur. The total capacity of Instruction RAM and Instruction cache may be reduced for special problem.

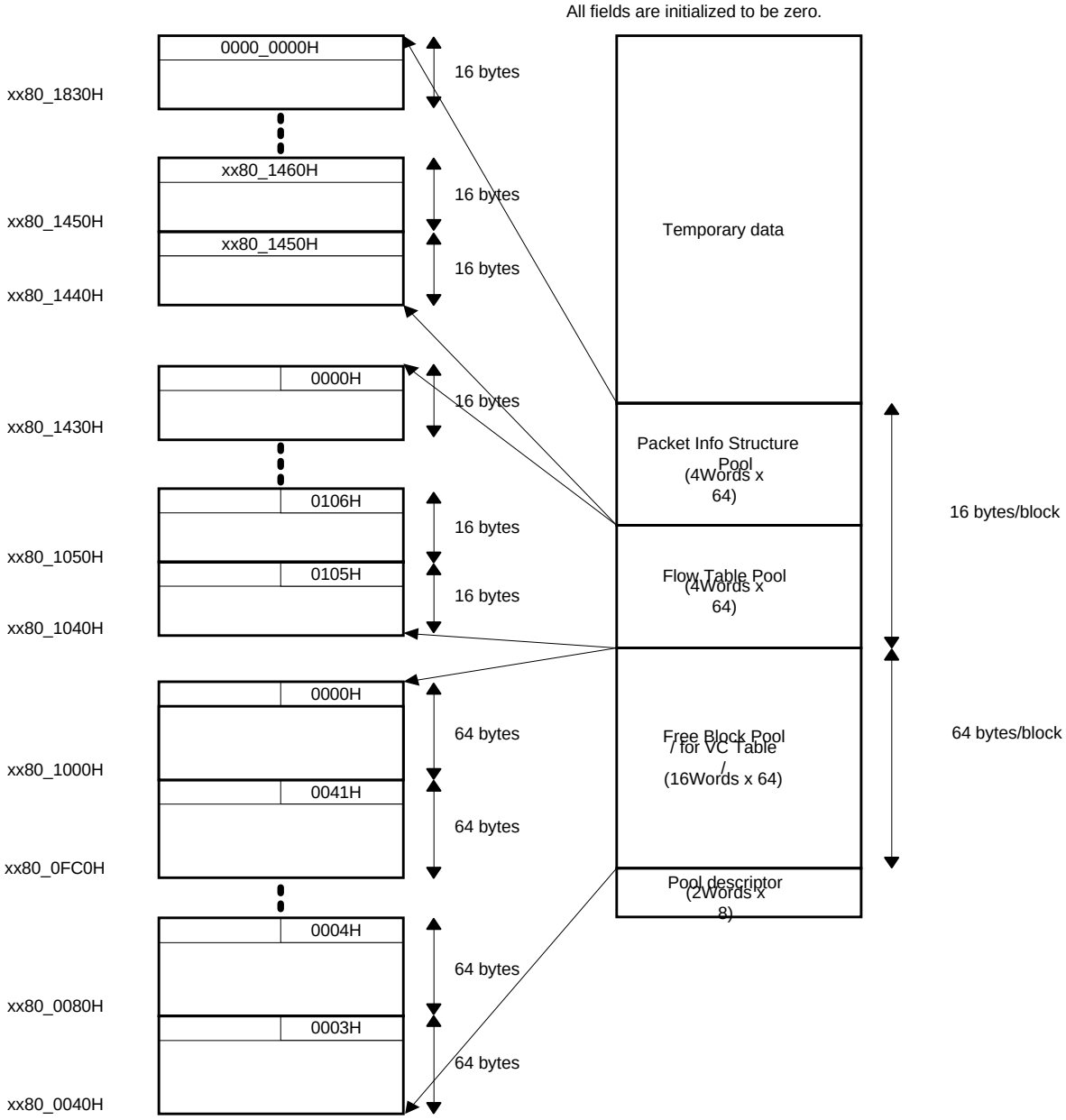
**Figure 4-18. Instruction RAM and Instruction Cache**



### 4.6.3 Work RAM initialization

When reset or software reset (write A_GMR register) is executed, RISC Core's F/W initializes Work RAM. ATM Cell Processor divides Work RAM into 16-word or 4-word blocks. It writes block numbers to the low-order 16 bits of each block or for packet info structure pool, writes next block start address. In both case, RISC Core writes 0 to the last block.

**Figure 4-19. Work RAM after Initialization**



## 4.7 Commands

ATM Cell Processor provides VR4120A RISC Processor with the following commands.

**Table 4-3. Commands**

| Command | What this block does |
|---|---|
| Set_Link_Rate | Set Phy Link Rate |
| Open_Channel | Reserves VC table area in Work RAM. |
| Close_Channel | Releases VC table area. |
| Open_IP_Channel | Reserves Flow Table area in Work RAM |
| Close_IP_Channel | Releases Flow Table area. |
| Tx_Ready | Starts transmission process. |
| IP_Flow_Ready | Starts IP flow transmission process |
| Add_Buffers | Add Buffer Directories to the indicated pool |
| Indirect_Access | Enables VR4120A RISC Processor to access Work RAM |
| Set_Rx_Congestion | (ABR) Set CI bit to 1 for all BRM cells |
| BRM_Tx | (ABR) Transmit out-of-rate BRM cell |

All commands are written in command register (A_CMR) and command extended register (A_CER) by VR4120A RISC Processor. Command register has busy flag. Since ATM Cell Processor only proceeds one command at a time, it sets the busy flag when it accepts the command. VR4120A RISC Processor cannot issue another command while this busy flag is 1. When finish the command operation, ATM Cell Processor sets the busy flag to 0. VR4120A RISC Processor has to reads the busy flag of the command register and checks if busy bit is 0, before issues new command.

(1) Commands which ATM Cell Processor returns command indication

When ATM Cell Processor receives Open_Channel, Close_Channel, Open_IP_Channel, Close_IP_Channel, Tx_Ready and Add_Buffers command, it writes command indication in command register. VR4120A RISC Processor has to read command indication, after issuing these commands. However, while busy flag is 1, ATM Cell Processor has not finished command processing yet, so that, VR4120A RISC Processor has to wait until busy flag in command register becomes 0 in order to read indication.

(2) Commands which command extended register is used

Indirect_Access command, Add_Buffers command and Tx_Ready command with IPoA mode uses command extended register.

When VR4120A RISC Processor writes these commands, VR4120A RISC Processor has to write command extended register first, and then command register. ATM Cell Processor starts command operation when command register is written. So that, unless command extended register is written first, the information in command extended register is ignored by ATM Cell Processor.

When command extended register is used for getting information from ATM Cell Processor, VR4120A RISC Processor writes command register and wait until busy flag in command register becomes 0, and reads command extended register.

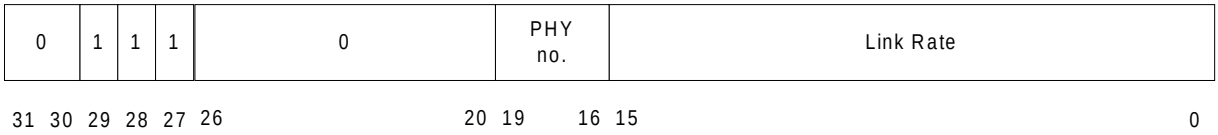
### 4.7.1 Set_Link_Rate command

This command is used to set the link rate. After initializing ATM Cell Processor, this command has to be issued once, before any packet transmission.

**Figure 4-20. Set_Link_Rate Command**

Set_Link_Rate Command

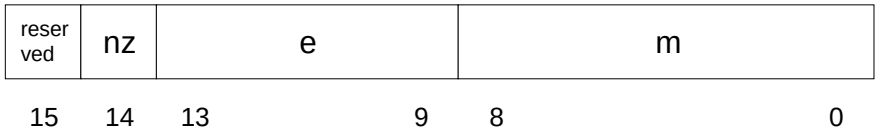
CMR




PHY No. PHY Number. However, this field is not used when ADSL DMT is connected to the μPD98501.

Link Rate Actual Link Rate is inserted in this field. The format of the rate is following.

$2^{e}(1+m/512)*nz$ cells/sec

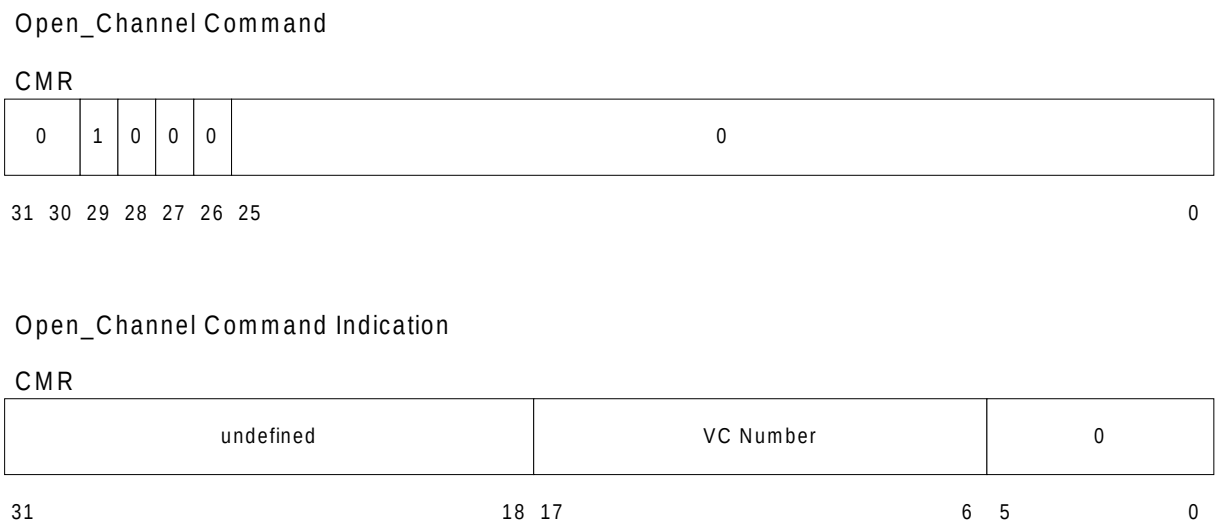


### 4.7.2 Open_Channel command

This command is used to open a new channel to be used for a send or receive operation. When the channel is opened, ATM Cell Processor reserves the area for the VC table from the VC Table pool in Work RAM and returns the VC Number to the VR4120A RISC Processor as an indication.

The indication that ATM Cell Processor returns for this command is of the following format:

**Figure 4-21. Open_Channel Command and Open_Channel Indication**




VC NUMBER VC number of the opened channel. If no more new VCs can be opened, VC NUMBER in the indication is set to 0.

### 4.7.3 Close_Channel command

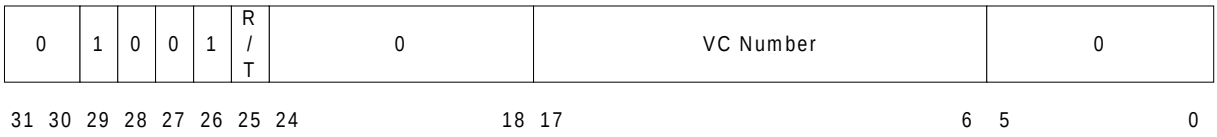
The Close_Channel command is used to close a send or receive channel. Upon accepting this command, ATM Cell Processor returns the VC table to VC Table pool.

The indication that ATM Cell Processor returns for this command has the following format:

**Figure 4-22. Close_Channel Command and Close_Channel Indication**

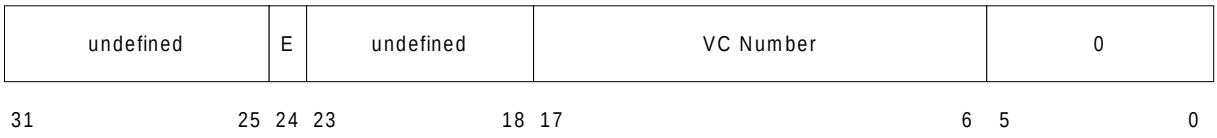
Close_Channel Command

CMR




Close_Channel Command Indication

CMR




**Close_Channel command**

| | |
|---|---|
| R/T | Indicates whether the channel to be closed is a send or receive channel.<br>1: Receive channel<br>0: Transmit channel |
| VC NUMBER | VC Number of the channel which VR4120A RISC Processor intends to close. |

**Close_Channel indication**

| | |
|---|---|
| E | Error bit. If detects an error, sets this bit to 1. When VR4120A RISC Processor issues this command, this bit has to be 0.The possible errors is invalid VC number . |
| VC NUMBER | Closed VC Number of the channel. If the channel cannot be closed, 0 is set in this area. |

### 4.7.4 Open_IP_Channel command

This command is used to open a new IP Flow to be used for a send or receive operation. When the IP Flow is opened, ATM Cell Processor reserves the area for the Flow Table from Flow Table pool in Work RAM and returns the start address of the Flow Table to the VR4120A RISC Processor as an indication.

The indication that ATM Cell Processor returns for this command is of the following format:

**Figure 4-23. Open_IP_Channel Command and Open_IP_Channel Indication**

Open_IP_Channel Command

CMR

Open_IP_Channel Command Indication

CMR

IP Flow Number — IP Flow Number of the opened IP channel. If no more new IP flow can be opened, IP flow number in the indication is set to 0.

### 4.7.5 Close_IP_Channel command

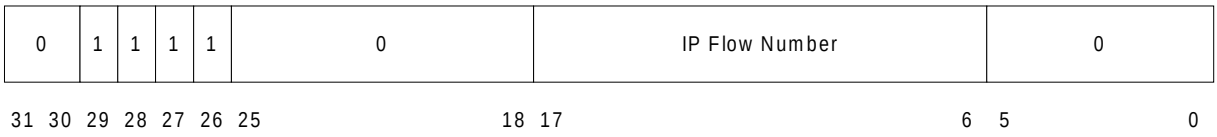
The Close_IP_Channel command is used to close a send or receive IP channel. Upon accepting this command, ATM Cell Processor returns the Flow Table to the Flow Table pool.

The indication that ATM Cell Processor returns for this command has the following format:

**Figure 4-24. Close_Channel Command and Close_Channel Indication**

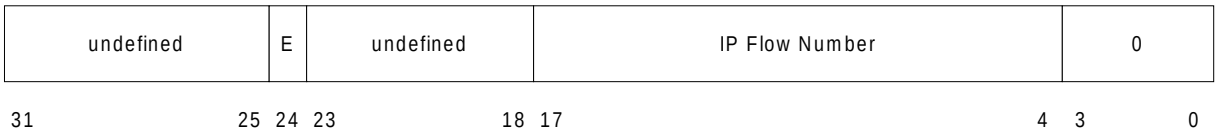
Close_IP_Channel Command

CMR




Close_IP_Channel Command Indication

CMR




**Close_IP_Channel command**

| | |
|---|---|
| IP Flow Number | IP Flow Number which VR4120A RISC Processor intends to close. |

**Close_IP_Channel indication**

| | |
|---|---|
| E | Error bit. If detects error, sets this bit to 1. When VR4120A RISC Processor issues this command, this bit has to be 0. The possible error is invalid IP Flow Number. |
| IP Flow Number | Closed IP Flow Number of the channel. If the channel cannot be closed, 0 is set in this area. |

### 4.7.6 Tx_Ready command

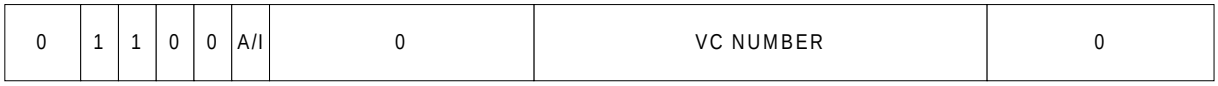
The Tx_Ready command is used by the VR4120A RISC Processor to notify ATM Cell Processor that a transmit packet has been added for a specified channel (a new packet descriptor has been added to the send queue). Upon receiving this command, ATM Cell Processor makes the scheduling table active to perform scheduling. In this command, when it detects some errors, writes E bit in A_CMR.

This command has the following format:

**Figure 4-25. Tx_Ready Command and Tx_Ready Command Indication**

Tx_Ready Command

CMR




31 30 29 28 27 26 25 24 18 17 6 5 0

CER

Tx_Ready Command Indication

CMR

31 30 29 28 27 26 25 24 23 18 17 6 5 0

**Tx_Ready command**

| | |
|---|---|
| A/I | Indicates whether to send ATM mode packet or IPoA mode packet. In ATM mode, this bit is set to 1. |
| VC NUMBER | The VC number of the channel which VR4120A RISC Processor intends to start transmission. In IPoA mode, this field is invalid and has to be set to 0. |
| Packet descriptor address | Address of the packet descriptor. |

**Tx_Ready command indication**

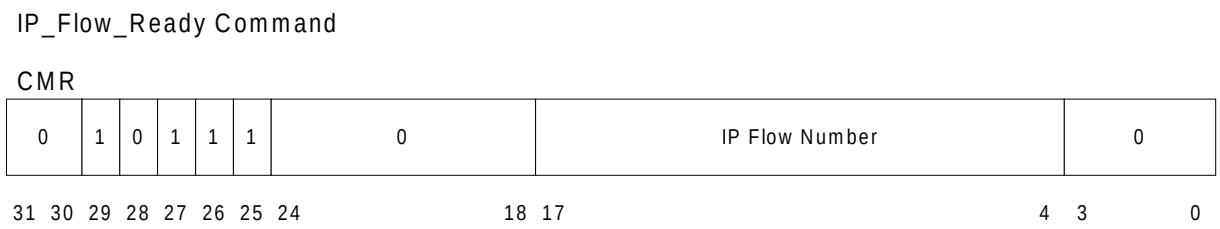
| | |
|---|---|
| A/I | Indicates whether to send ATM mode packet or IPoA mode packet. In ATM mode, this bit is set to 1. |
| E | Error bit. If detects an error, sets this bit to 1. When VR4120A RISC Processor issues this command, this bit has to be 0.The possible errors are:<br>VC number is invalid.<br>PacketInfo structure cannot be obtained.<br>The flow is used in different transmission mode. |
| VC NUMBER | The VC number of the channel which VR4120A RISC Processor intends to start transmission. |

### 4.7.7 IP_Flow_Ready command

The IP_Flow_Ready command is used by the VR4120A RISC Processor to notify ATM Cell Processor that a IP Flow information has been ready for a specified channel (a new IP Flow table has been added). Upon receiving this command, ATM Cell Processor sets the H/W IP Lookup table in order to enable IP flow search.

This command has the following format:

**Figure 4-26. IP_Flow_Ready Command**




**IP_Flow_Ready command**

| | |
|---|---|
| IP Flow Number | IP Flow Number which VR4120A RISC Processor intends to start transmission. |

### 4.7.8 Add_Buffer command

The Add_Buffers command is used to add unused buffer directories to a single receive free buffer pool.

In this command, when ATM Cell Processor detects some errors, it writes E bit in A_CMR. This command has the following format:

**Figure 4-27. Add_Buffers Command**

Add_Buffers Command

CMR

CER

Add_Buffers Command Indication

CMR

**Add_Buffers command**

| | |
|---|---|
| POOL NO. | :Number of the pool to which buffer directory are to be added. Since ATM Cell Processor supports only 8 pools(0- 7), Bit 19 and 20 should be 0. |
| NUMBER OF BUFFER DIRECTORIES | :Number of new buffer directories to be added |
| BUFFER DIRECTORY POINTER | :Start address of the first buffer directory in the list of new buffer directories to be added |

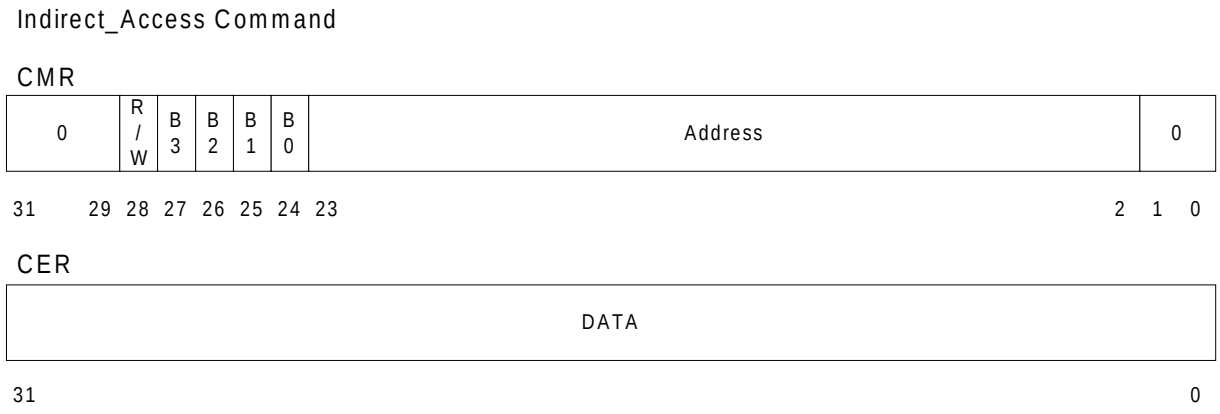
**Add_Buffers command indication**

| | |
|---|---|
| E | Error bit. If detects an error, sets this bit to 1. When VR4120A RISC Processor issues this command, this bit has to be 0.The possible errors is the number of buffer directories after this command is executed exceeds 65535. |
| POOL NO. | Number of the pool to which new buffer directories are just added |
| NUMBER OF BUFFER DIRECTORIES | Number of new buffer directories just added |

### 4.7.9 Indirect_Access command

The Indirect_Access command is used to perform read/write access to Work RAM.

**Figure 4-28. Indirect_Access Command**



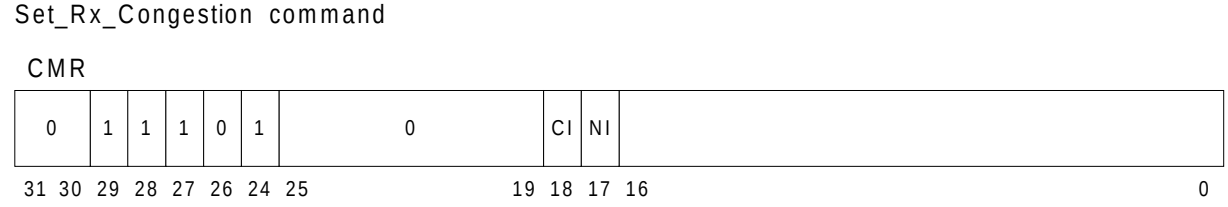

**Indirect_Access command**

| | |
|---|---|
| R/W | Specifies whether access to the target is read or write access.<br>1: Read<br>0: Write |
| B0, B1, B2, B3 | For write access, used to select bytes. |
| ADDRESS | If the address specified in this area is within control memory, the virtual address is converted to ATM Cell Processor address. Otherwise, the address is used as is. The low-order two bits of the address must be 0, that is, the address must be a word address. |

### 4.7.10 Set_Rx_Congestion command

The Set_Rx_Congestion command is used to set CI and NI bit for all Backward RM cells. When this command is issued, ATM Cell Processor takes OR of the CI and NI bit in the command and BRM cells which ATM Cell Processor sends back, and sets CI and NI bits in BRM cells. This command is valid until another Set_Rx_Congestion command with CI=NI=0 is issued.

**Figure 4-29. Set_Rx_Congestion Command**



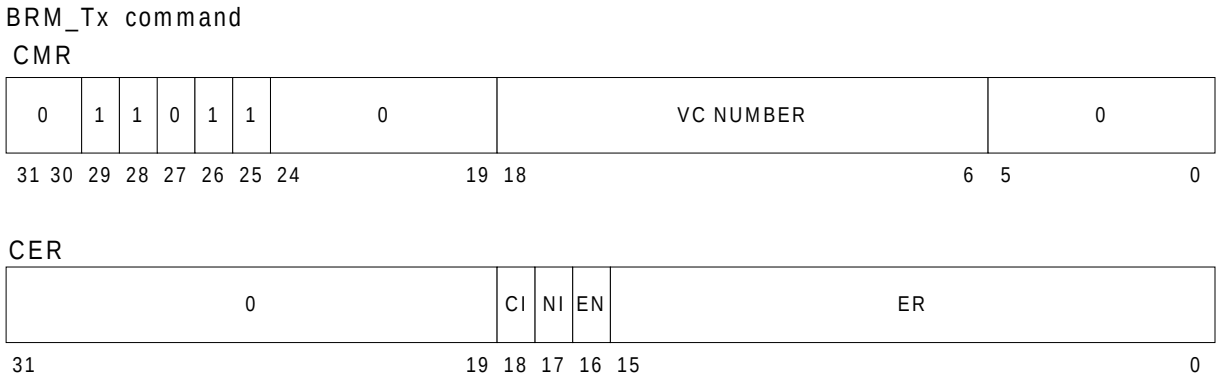

**Set_Rx_Congestion command**

| | |
|---|---|
| CI | CI (Congestion Indication) bit in BRM cell. |
| NI | NI(No Increase) bit in BRM cell. |

### 4.7.11 BRM_Tx command

BRM_Tx Command is the command for transmitting out-of-rate BRM cell. When this command is issued, ATM Cell Processor transmit one out-of-rate BRM cell with VC indicated in the command. In these BRM cells, the parameter value in A_CER are used. If EN bit is disable, the ER value for the VC which is saved in ATM Cell Processor is used, not the value in the command. If EN bit is enable, the smaller value between the saved ER value and the ER value in the command is set.

**Figure 4-30. BRM_Tx Command**




**BRM_Tx command**

| | |
|---|---|
| VC NUMBER | Rx VC Number. |
| CI | CI bit in BRM cell. |
| NI | NI bit in BRM cell. |
| EN | Enable for ER field.<br>1: Enable<br>0: Disable |
| ER | ER(Explicit Rate) field for BRM cell. |

## 4.8 Operations

In this section, functional specifications mainly SAR function is described.

### 4.8.1 Work RAM usage

The size of the Work RAM is 16 Kbytes. This memory is used for following five purposes.

(1) Temporary data

The data which are exchanged with SDRAM using DMA. The data stored in this area are transmitting and receiving indications and first cell of IP packet.

(2) Flow Table Pool

The area in which Flow Tables are stored.

(3) Packet Info structure

The area in which Packet Info structures are stored.

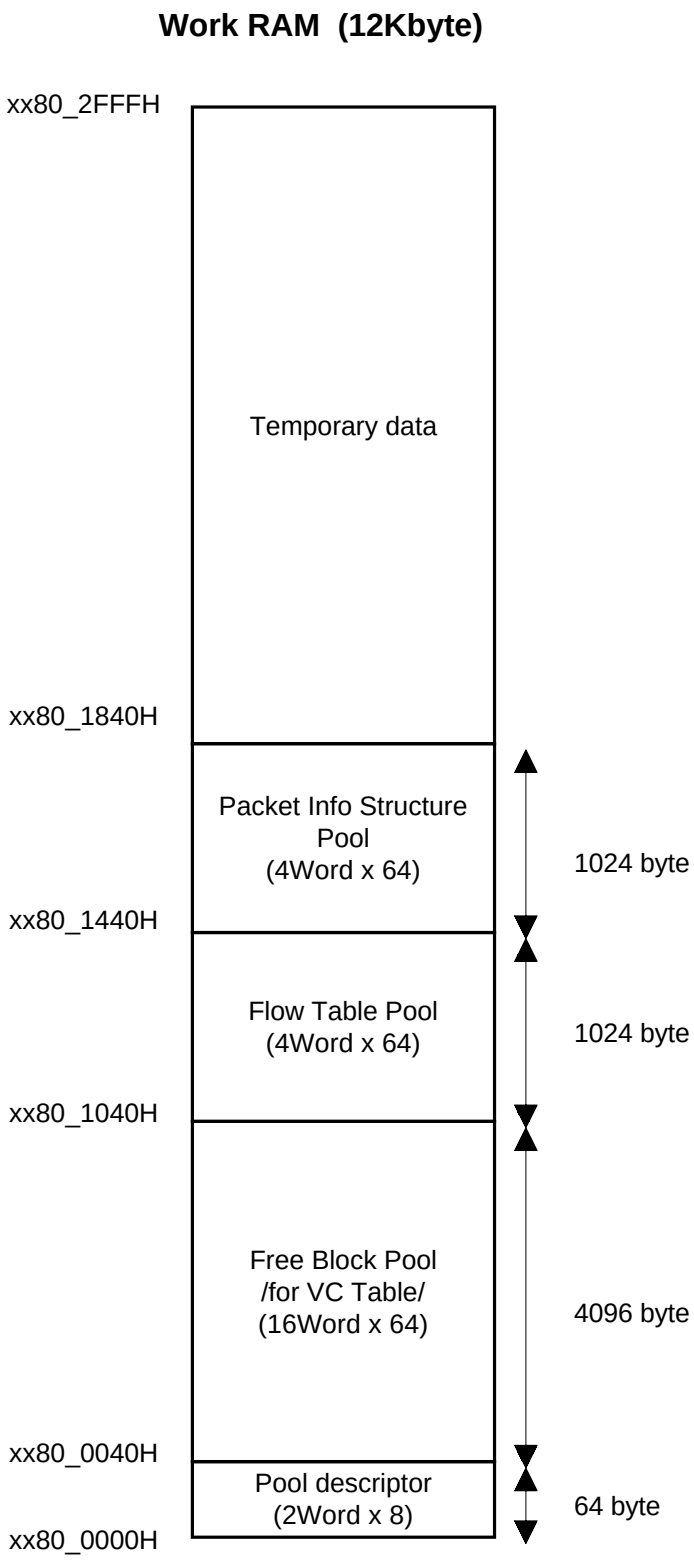
(4) Receive free buffer pool

The are in which “pool descriptors” are stored. Each pool descriptors uses 2 words.

(5) VC table pool

The area in which transmit and receive VC tables are stored. Each VC table uses 1 block (16 words).

**Figure 4-31. Work RAM Usage**



### 4.8.2 Transmission function

ATM Cell Processor has two transmission operation mode. One is ATM mode and another is IPoA mode. In ATM mode, VR4120A RISC Processor sets the VC information in VC table prior to the every packet transmission, and issue Tx_Ready command with VC number in order to transmit packet belongs to the VC. In IPoA mode, VR4120A RISC Processor also has to sets the VC information in VC table at first, and also sets the mapping between IP flow and VC. Once VR4120A RISC Processor sets those information, VR4120A RISC Processor only sets the transmitting data in SDRAM and issue Tx_Ready command with packet start address. ATM Cell Processor finds out TCP/IP header from the packet and chooses corresponding VC automatically.

The transmitting data structure is described in Section. 4.5.1.

#### 4.8.2.1 Transmission procedure

**(1) ATM mode**

(a) Setting transmitting data
Before transmitting a packet, VR4120A RISC Processor stores transmitting data in SDRAM and sets the packet descriptor.

(b) Opening the send channel
If VR4120A RISC Processor needs a new channel for transmission, VR4120A RISC Processor issues Open_Channel command. When VR4120A RISC Processor issues the command, ATM Cell Processor assigns the new block from the VC Table pool in Work RAM and reports its start address to VR4120A RISC Processor using a command indication. VR4120A RISC Processor sets the assigned block as a send VC address.

(c) Setting the send VC table
The 16-word block assigned from the VC Table pool in Work RAM is set as the send VC table for each VC.

(d) Issuing the Tx_Ready command -> making preparations for transmission of the first cell
When VR4120A RISC Processor issues Tx_Ready command, ATM Cell Processor sets the Packet Info Structure in Work RAM, fetch the packet descriptor and store it in the area. ATM Cell Processor checks Transmit Queue if any packet is waiting for transmission. If Transmit Queue is not empty, ATM Cell Processor adds the Packet Info structure at the end of the queue. If no packet is waiting in the queue, ATM Cell Processor also schedules next transmission time with “current time plus 1”.

(e) Sending a single cell (called when hit by the scheduler)
<1> Generating a header
A header is generated from Word1 in the VC table and written into SAR FIFO. "00H" is inserted into the GFC field of the header.
<2> Sending a segment data from SDRAM to SARFIFO
ATM Cell Processor reads a transmitting segment (48-byte payload data of the cell) from SDRAM and set it in SAR FIFO using Scatter/Gather DMA. The starting address of the segment is indicated by the "Buffer Read Address" field in the VC table. When the 53rd byte of the segment is written, SAR FIFO is updated.
<3> Calculating the CRC-32 value and the length
Each time a segment is read from SDRAM, the CRC-32 value is calculated for that segment and transmitted bytes are also counted. ATM Cell Processor writes those results in VC table.
<4> Updating the VC table
Updates "Buffer Read Address" and "Remaining Bytes in Current Buffer" fields.

(f) Sending the last cell

When the L flag of the current transmit buffer indicates that the buffer is the last one and the value in the field indicating the number of bytes remaining in the VC table is less than 40 bytes, the cell is the last cell of the packet.

<1> If the current cell is the last cell of the packet, and the remaining payload data is less than 40 bytes, zero padding and the 8 byte trailer are added. If the remaining payload data is more than 40 bytes and there are not enough space to add an 8-byte AAL-5 trailer, ATM Cell Processor just adds zero padding to make 48 byte payload and ATM Cell Processor sends a cell containing only a trailer and padding next.

<2> When the last segment of the AAL-5 PDU is read, the final CRC-32 value and the packet length are inserted into the trailer of the AAL-5 PDU, and the contents of the first word in the VC table are inserted into the CPCS-UU and CPI fields, thereby completing the AAL-5 trailer.

(g) ATM Cell Processor checks whether there is a subsequent packet ( checks Last Packet Info address and First Packet Info address in Tx VC table). When a subsequent packet exists, (e) and (f) are repeated.

(h) For each packet, ATM Cell Processor stores transmitting indication as a status information in the mailbox and generates an interrupt.

(i) VR4120A RISC Processor reads the mailbox and updates the read pointer of the mailbox.

**(2) IPoA mode**

(a) Setting transmitting data

Before transmitting a packet, VR4120A RISC Processor stores transmitting data in SDRAM and sets the packet descriptor. In IPoA mode, the transmitting data should be a IP packet.

(b) Opening the send channel

If VR4120A RISC Processor needs a new channel for transmission, VR4120A RISC Processor issues Open_Channel command. When this command is issued, ATM Cell Processor assigns the new block from the VC Table pool in Work RAM and reports its start address to VR4120A RISC Processor using a command indication. VR4120A RISC Processor sets the assigned block as a send VC address.

(c) Setting the send VC table

The 16-word block assigned from the VC Table pool in Work RAM is set as the send VC table for each VC.

(d) Opening the IP flow

If the IP flow has not registered yet, VR4120A RISC Processor issues Open_IP_Channel command. When this command is issued, ATM Cell Processor assigns the 3 words block for flow table and reports its start address to VR4120A RISC Processor using command indication. VR4120A RISC Processor sets the IP flow information and VC number in the flow table.

(e) Issuing the IP_Flow_Ready command

When VR4120A RISC Processor issues the IP_Flow_Ready command, ATM Cell Processor sets the H/W IP lookup table using the data which VR4120A RISC Processor sets in the IP Flow table.

(f) Issuing the Tx_Ready command -> making preparations for sending of the first cell

When VR4120A RISC Processor issues the Tx_Ready command, ATM Cell Processor sets the Packet Info structure in the Work RAM and fetches the packet descriptor using Scatter/Gather DMA and store it in the area. ATM Cell Processor also fetches the first segment to the Work RAM using Scatter/Gather DMA. Then, ATM Cell Processor searches if the IP Flow exist in IP Lookup Table using the IP header information in the first segment. If IP Flow is found, ATM Cell Processor gets the corresponding VC number.

(g) Sending a single cell

<1> Generating a header

A header is generated from Word1 in the VC table and written into SAR FIFO. "00H" is inserted into the GFC field of the header.

<2> Generate LLC or PPP header if needed

If indication bits for LLC or PPP are 1 in VC table, ATM Cell Processor generates LLC or PPP header and writes it in SAR FIFO. LLC and PPP encapsulations are exclusive.

<3> Sending a segment data from SDRAM to SARFIFO

ATM Cell Processor reads a transmitting segment (48-byte payload data of the cell) from SDRAM and set it in SAR FIFO using Scatter/Gather DMA. The starting address of the segment is indicated by the "Buffer Read Address" field in the VC table. However, if LLC or PPP encapsulation is executed, the number of bytes to fetch are 40 or 46 bytes ( if RAS PPP encapsulation is used, 44 bytes). When the 53rd byte of the segment is written, SAR FIFO is updated.

<4> Calculating the CRC-32 value and the length

Each time a segment is read from system memory, the CRC-32 value is calculated for that segment and transmitted bytes are also counted. ATM Cell Processor writes those results in VC table.

<5> Updating the VC table

Updates "Buffer Read Address" and "Remaining Bytes in Current Buffer" fields.

(h) Sending the last cell

When the L flag of the current send buffer indicates that the buffer is the last one and the value in the field indicating the number of bytes remaining in the VC table is less than 40 bytes, the cell is the last cell of the packet.

<1> If the current cell is the last cell of the packet, and the remaining payload data is less than 40 bytes, zero padding and the 8 byte trailer are added. If the remaining payload data is more than 40 bytes and there are not enough space to add an 8-byte AAL-5 trailer, ATM Cell Processor just adds zero padding to make 48 byte payload and sends a cell containing only a trailer and padding next.

<2> When the last segment of the AAL-5 PDU is read, the final CRC-32 value and the packet length are inserted into the trailer of the AAL-5 PDU, and the contents of the first word in the VC table are inserted into the CPCS-UU and CPI fields, thereby completing the AAL-5 trailer.

(i) ATM Cell Processor checks whether there is a subsequent packet ( checks Last Packet Info address and First Packet Info address in Tx VC table). When a subsequent packet exists, (f) and (g) are repeated.

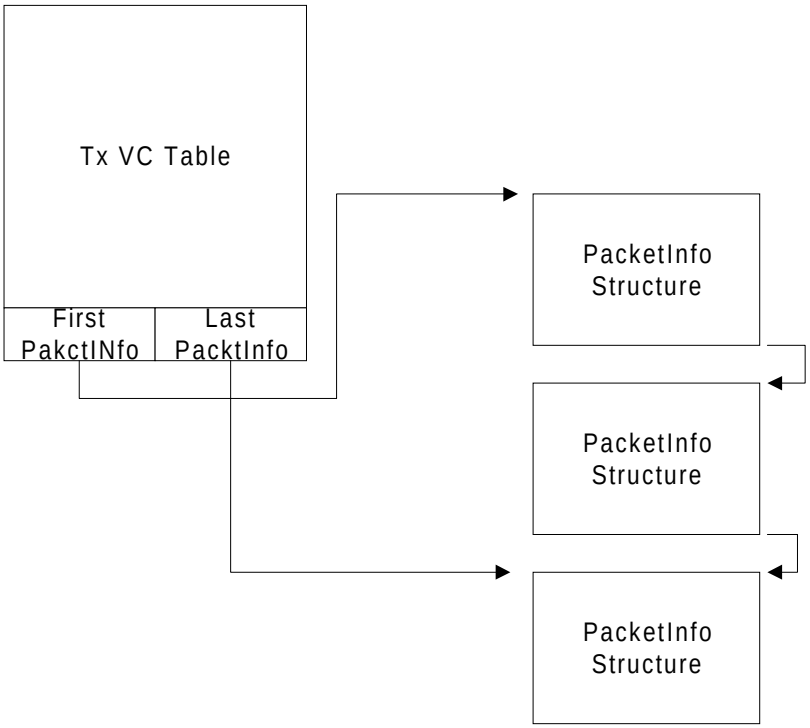
(j) For each packet, ATM Cell Processor stores send indication as status information in the mailbox and generates an interrupt. Then ATM Cell Processor releases Packet Info area.

(l) VR4120A RISC Processor reads the mailbox and updates the read pointer of the mailbox.

#### 4.8.2.2 Transmit queue

Tx_Ready command has to be issued in order to transmit a packet. However, VR4120A RISC Processor doesn't have to wait Tx indication before issuing next Tx_Ready command for the same VC. When VR4120A RISC Processor issues Tx_Ready command before completing transmission process for the previous packet, ATM Cell Processor builds Tx Queue for that VC.

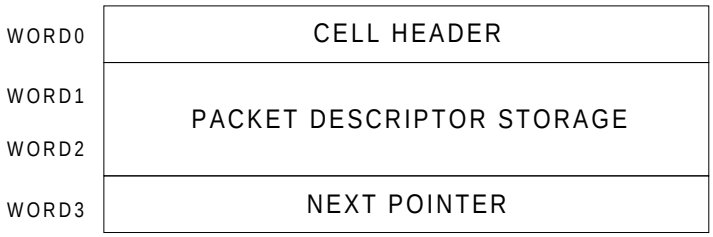
**Figure 4-32. Structure of the Transmit Queue**




**(1) Packet info structure**

The Maximum number of Packet Info Structure for each VC is 16. However, since total number of Packet Info Structure is 128, some VC may not obtain 16 Packet Info Structure depend on the queue length of other VC. When ATM Cell Processor can not obtain any Packet Info Structure, ATM Cell Processor returns error for Tx_Ready command.

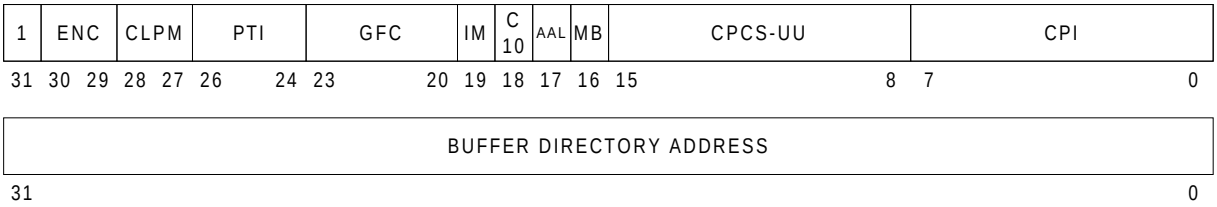
**Figure 4-33. Packet Info Structure**




| | |
|---|---|
| CELL HEADER | Cell header |
| PACKET DESCRIPTOR STORAGE | Area for temporarily storing the packet descriptor of a packet |
| NEXT POINTER | Address of the next send queue (Packet Info structure) |

**(2) Packet descriptor**

**Figure 4-34. Transmit Queue Packet Descriptor**




ENC Encapsulation mode is indicated.
- 11 PPP-NT encapsulation
- 10 PPP encapsulation
- 01 LLC encapsulation
- 00 No encapsulation

CLPM CLP bit in the packet header is indicated.
- 00 Sets CLP bit in all cells as 0.
- 11 Sets CLP bit in all cells as 1.
- 01 Sets CLP bit in all cells except last cell as 1, and only CLP bit in last cell as 0.
- 10 Not used.

PTI PTI field in the packet header is indicated. Since this field indicates cell payload type, takes different transmission procedure according to this field .
- 000 User data cell without congestion.
- 001 User data cell without congestion. Last cell in the packet.
  Prohibited to be set by user. If this value is set, treats as 000.
- 010 User data cell with congestion.
- 011 User data cell with congestion. Last cell in the packet.
  Prohibited to be set by user. If this value is set, treats as 010.
- 100 OAM F5 cell used to carry information between segments
- 101 OAM F5 cell used to carry information between end to end.
- 110 RM cell used in ABR service. Prohibited to be set by user.
- 111 Prohibited to be set by user.

GFC GFC field in the packet header is indicated.

IM Indicates if ATM Cell Processor sends the transmitting indication to VR4120A RISC Processor.
1 : Doesn't send transmitting indication to VR4120A RISC Processor.
0 : Sends transmitting indication to VR4120A RISC Processor.

C10 Indicates if CRC-10 should be inserted or not.
1 : Insert CRC-10 in every cell.
0 : Not insert CRC-10.

AAL Indicates if the transmitting packet is AAL-5 packet or not.
1 : AAL-5 packet.
0 : Not AAL-5 packet. ATM Cell Processor treats cells in this packet as Raw cell.

MB Mailbox number for storing transmitting indication for this packet is indicated.
1 : Mailbox number 3.
0 : Mailbox number 2.

CPCS-UU User information. In AAL-5 mode, ATM Cell Processor inserts the pattern in this field to the CPI field in the trailer. In Raw cell mode, this field is ignored.

CPI User information. In AAL-5 mode, ATM Cell Processor inserts the pattern in this field to the CPI field in the trailer. In Raw cell mode, this field is ignored.

### (3) Tx VC table

**Figure 4-35. Tx VC Table**

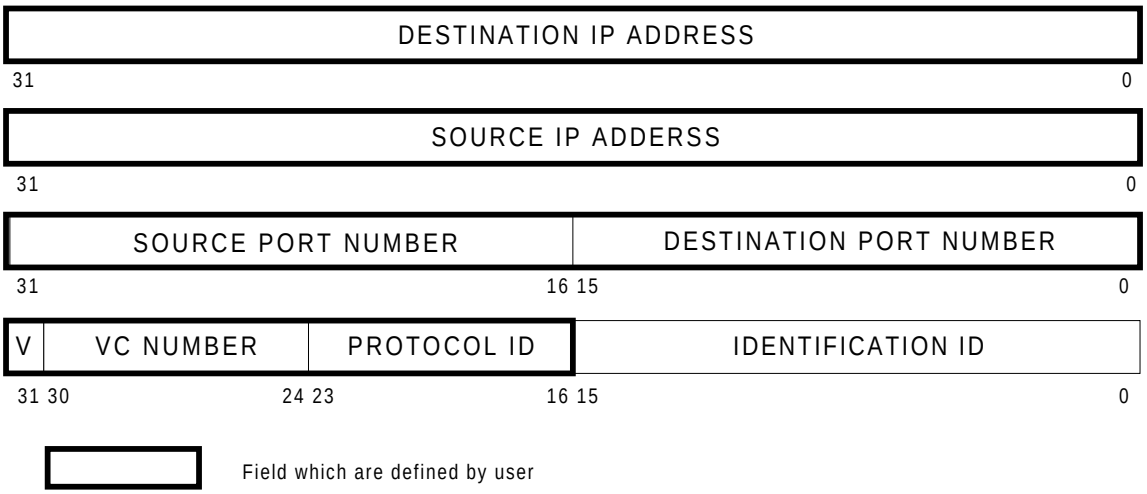
| | |
|---|---|
| Word0 | :The contents of Word0 in the packet descriptor read from SDRAM is stored as is. The initial value must be all zeros. |
| L | :This bit is used internally by SAR processing. The initial value must always be 1. |
| PRIORITY | :Specifies send priority .<br>111: CBR<br>110: VBR<br>101: ABR<br>010: UBR<br>100: GFR |
| VPI/VCI | :VPI/VCI values to be contained in the cell header |
| NO.OF BYTE TRANSMITTED IN THIS PACKET | :Number of bytes of this packet that have been sent so far. The initial value is 0. |
| REMAINING BYTES IN CURRENT BUFFER | :Number of bytes in the current buffer that have not been sent. The initial value is 0. |
| CRC-32 COUNT | :CRC-32 value calculated for those cells of this packet that have been sent. The final CRC-32 value is set in the CRC-32 field in the trailer. |
| BUFFER READ ADDRESS | :Pointer to the byte in the current buffer that is to be transferred next |
| NEXT BUFFER ADDRESS | :Pointer to the next buffer in the current packet directory |
| MBS0 | :Maximum Burst Size. The number of cells continuously transmitted with PCR. Only used in VBR mode. |
| MBS | :Internally used for counting MBS0. Initially, the same value as MBS0 has to be set. Only used in VBR mode. |
| PHY NO. | :The number of the PHY which this VC transmits to. |
| A | :"Active" bit indicating whether the VC table is in the active or idle state.<br>1: Active state<br>0: Idle state |
| Rx VC Number | :Pointer to the Rx VC Table. This field is used in ABR only. |
| MCR | :Minimum Cell Rate. In VBR and GFR mode, SCR has to be set in this field. |
| PCR | :Peak Cell Rate. |
| ICR | :Initial Cell Rate. Only used in ABR mode. |
| ACR | :Allowed Cell Rate. Initially, the same value as ICR has to be set. Only used in ABR mode. |
| WORD 10 | :These fields are used internally for scheduling use. |
| WORD 11 & WORD 12 | :These fields are used internally for ABR use. |
| ADTF0 | :The ACR Decrease Time Factor. This field is used in ABR only. The format is following:<br>ADTF(millisecond) = $10*2^{ADTF}$ |
| CTF0 | :The Cutoff Decrease factor. This field is used in ABR only. The format is following:<br>CDF0: 0-6 CDF=$1/2^{CDF}$<br>CDF0: 7 CDF=0 |
| FIRST PACKET INFO | :The start address of the first Packet Info structure in transmit queue for this VC. The initial value has to be 0. |
| LAST PACKET INFO | :The start address of the last Packet Info structure in transmit queue for this VC. The initial value has to be 0. |

#### 4.8.2.3 Flow table for IPoA mode

For sending in IPoA (IP over ATM) mode, a flow table must be set. The flow table is required to specify the send flow for the send IP packet.

**(1) Flow table**

**Figure 4-36. Table Format**




| | |
|---|---|
| DESTINATION IP ADDRESS | IP address of destination. |
| SOURCE IP ADDRESS | IP address of source. **Only uses lower 16 bits of the source IP address for IP flow matching.** |
| SOURCE PORT NUMBER | Port number of send source |
| DESTINATION PORT NUMBER | Port number of destination |
| V | Valid bit<br>0: Invalid<br>1: Valid |
| VC NUMBER | VC Number which corresponding to this IP flow |
| PROTOCOL ID | Protocol ID |
| IDENTIFICATION ID | Packet ID<br>For a divided IP packet, in order to discriminate between packets, ATM Cell Processor sets this item |

In IPoA mode, this flow table is used to match IP flow to VC channel. Multiple IP flow may be mapped into one VC channel.

#### 4.8.2.4 Non AAL-5 traffic support

**(1) OAM F5 cell transmission**

When host sets OAM F5 cell pattern(100 and 101) in the PTI field in packet descriptor, ATM Cell Processor doesn't add AAL-5 trailer. In this case, even though host sets more than 48 bytes in "SIZE" field in the packet descriptor, ATM Cell Processor only reads 48 bytes from the top of the data buffer and ignores the data after that. If host sets the bytes less than 48 byte in "SIZE" field, ATM Cell Processor adds padding. For OAM F5 cell transmission, host has to set different packet descriptor for each OAM F5 cell.

For OAM F5 cell transmission, ATM Cell Processor inserts CRC-10, if host sets "C10 bit" to 1 in packet descriptor. ATM Cell Processor calculates CRC-10 for 46 bytes and 6 bits and overwrites the result to the last 10 bits in the payload.

**(2) Raw cell transmission**

When host sends the non AAL-5 traffic packet which is not OAM F5 cell, host sets "AAL" bit in the packet descriptor to 0 and "PTI" field "0xx" which indicates user data. ATM Cell Processor doesn't calculate or add AAL-5 trailer if "AAL" bit is 0.

If host sets "C10" bit in packet descriptor to 1, ATM Cell Processor calculates and adds CRC-10 for each cell.

**Figure 4-37. Raw Cell with CRC-10**

| Header | Payload | CRC-10 |
|---|---|---|
| 5 bytes | 46 bytes and 6bits | 10 bits |

Generation polynomial of CRC-10 is following:

$G(x) = 1+x+x^4+ x^5+ x^9+ x^{10}$

#### 4.8.2.5 Transmission indication

For each packet, ATM Cell Processor writes the send indication as the transmission completion status in the mailbox. The mailbox used for transmission is mailbox 2 and 3. More specifically, ATM Cell Processor writes a send indication once all the data in the packet has been read. The issuing of a send indication does not, therefore, indicate that the sending of the packet to PMD has been completed.

Upon storing a send indication into the mailbox, ATM Cell Processor sets the corresponding MM bit of the A_GSR register to 1, and issues an interrupt if it is not masked.

The indication that ATM Cell Processor sends to the host during transmission is of the following format:

**Figure 4-38. Send Indication Format**

| 1 | VC NUMBER | A | PACKET QUEUE POINTER |
|---|---|---|---|
| 31 30 | 16 15 14 | 0 | |

| | |
|---|---|
| VC Number | VC NUMBER used by this VC |
| A (for Active) | If 0, indicates that the VC enters the idle state because the packet descriptor is invalid. |
| | If 1, indicates that the VC is kept active because the next packet descriptor is valid. |
| Packet Queue Pointer | Low-order 15 bits of the start address of the next packet descriptor in the send queue |

#### 4.8.2.6 Scheduling

ATM Cell Processor holds a scheduling table in which ATM Cell Processor sets the transmitting timings of all active channels. Transmitting timing is recalculated each time ATM Cell Processor transmits a cell. Transmitting timing is calculated using line rate and rate information which is set by host prior to the Tx_Ready command. Rate information is written in Tx VC table. For ABR, rate information is dynamically changed. ABR scheduling is described in Section. 4.8.5.

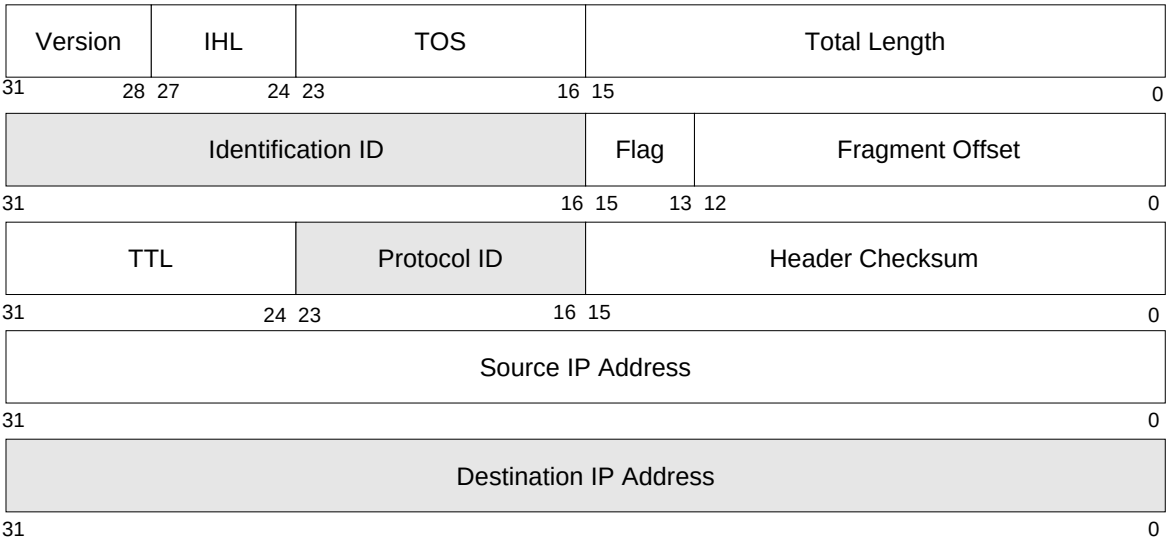
#### 4.8.2.7 IP support

ATM Cell Processor dynamically maps IP flow into ATM flow. Once VR4120A RISC Processor sets the IP flow to ATM flow mapping, VR4120A RISC Processor doesn't have to care ATM flow afterwards. VR4120A RISC Processor just prepares the packet and informs start address of the packet data with Tx_Ready command.

**(1) IP header**

ATM Cell Processor compares Flow table and IP header of transmitting IP packet. If the Fragment is 0, that means the first fragment, TCP header is also compared. When Flow table has same information as TCP/IP headers have, ATM Cell Processor transmits a cell using a specific VC. IP header and TCP header are following. Here, the fields darkened are used for comparison between flow table.

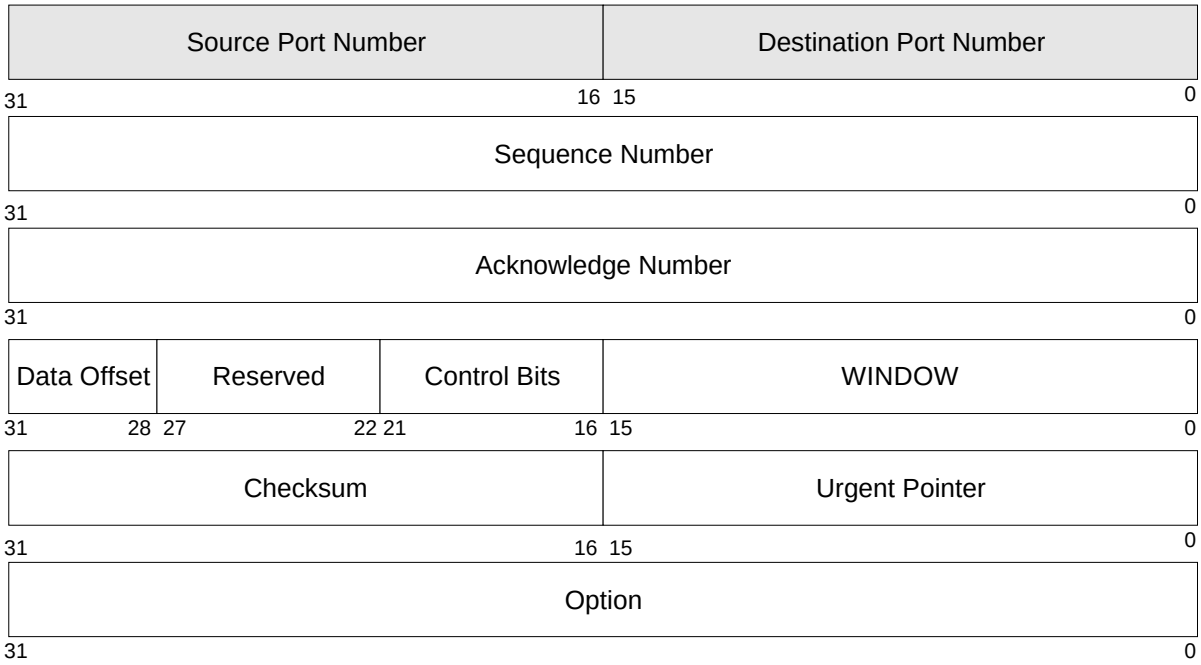
**Figure 4-39. IP Header**




(a) Version: Indicating the IP header version.
(b) IHL (Internet Header Length) : Indicating the length of the IP header in word.
(c) TOS (Type of Service) : Indicating QoS (Quality of Service) for the datagram.
(d) Total Length: Indicating datagram size in octet. The maximum size of IP datagram is 65545 octets.
(e) Identification : Indication for each datagram used to reconstruct divided datagrams.
(f) Flag : indicating the divide operation.
(g) Fragment Offset: Indicating where this fragment is located in original datagram. The unit is 8 octets.
(h) Time to Live (TTL): Indicating how long the datagram can exists in the network. The unit is a second.
(i) Protocol: Indicating the protocol in higher layer.
(j) Header Checksum. Indicating the checksum of IP header.
(k) Source Address:
(l) Destination Address.

**(2) TCP header**

**Figure 4-40. TCP Header**



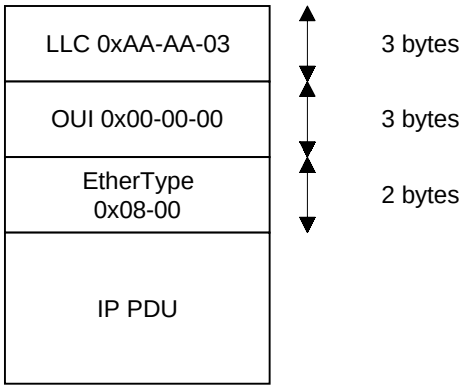

(a) Source Port Number:
(b) Destination Port Number:
(c) Sequence Number: Indicating the sequence number of the segment if SYN flag is not ON. If SYN flag is on, Initial sequence number is indicated.
(d) Acknowledgement Number: If ACK flag is ON, the sequence number of the next data is indicated.
(e) Data Offset: The length of the TCP header.
(f) Reserved: The field for future use.
(g) Control bits:
URG: Urgent pointer
ACK: ACK field
PSH: Push function is executed.
RST: Reset the connection
SYN: Used for initial sequence number procedure
FIN: Inform to the destination that there is no data to send.
(h) Window: The number of octets destination can receive.
(i) Checksum:
(j) Urgent Pointer: Valid when URG bit is ON.

ATM Cell Processor doesn't recognize Option field in TCP header. ATM Cell Processor treats this fields as data.

#### 4.8.2.8 LLC encapsulation

If LLC encapsulation is indicated in Tx VC table, ATM Cell Processor adds the LLC header to the top of the IP packet which is sent using IPoA mode Tx_Ready command. ATM Cell Processor always encapsulates CPCS-PDU as Internet IP PDU.

**Figure 4-41. LLC Encapsulation Format**




#### 4.8.2.9 PPP encapsulation

If PPP encapsulation is indicated, ATM Cell Processor adds the PPP header to the top of the IP packet which is sent using IPoA mode Tx_Ready command. ATM Cell Processor always encapsulates CPCS-PDU as IP-PDU. ATM Cell Processor adds 0x0021 if it is normal PPP or 0xff03-0021 if it is NT-PPP at the top of the CPCS-PDU.

### 4.8.3 Receiving function

Receiving data structure is described in Section. 4.5.2.

#### 4.8.3.1 Receiving procedure

(a) Setting up a receive pool

Before receiving a packet, VR4120A RISC Processor prepares the receive pool used to store receive data and sets the information about the pool in the pool descriptor in Work.

(b) Opening the receive channel

If VR4120A RISC Processor is going to open a new connection, VR4120A RISC Processor issues Open_Channel command. If Open_Channel command is issued, ATM Cell Processor assigns VC Table from VC Table pool in Work RAM and reports its start address to VR4120A RISC Processor using a command indication. VR4120A RISC Processor sets the assigned block as a VC table.

(c) Setting the receive VC table

VR4120A RISC Processor sets the 16-word blocks assigned from VC Table pool in Work RAM as the receive VC table for each VC.

(d) Setting up the reception lookup table

The VPI/VCI of the receive cell are set in the receive lookup table.

(e) Receiving the first cell of the packet

Upon ATM Cell Processor receiving a cell, it checks if the VPI/VCI of the received cell has been registered in receive lookup table. If not registered, the cell is discarded. If registered, ATM Cell Processor gets the VC number from receive lookup table. If it is a first cell of a packet, ATM Cell Processor assigns the new buffer directory. The VC is added to the T1 timer list.

(f) Receiving a cell

ATM Cell Processor transfers the segment data from SAR FIFO to SDRAM using Scatter/Gather DMA. In the same way as in transmission, the CRC-32 value calculated for each segment is stored into the "CRC-32" field of the VC table. The "Current Count of Segments and "Remaining words in current buffer" fields in the VC table are updated.

(g) Receiving the last cell

<1> The trailer information is checked for errors.

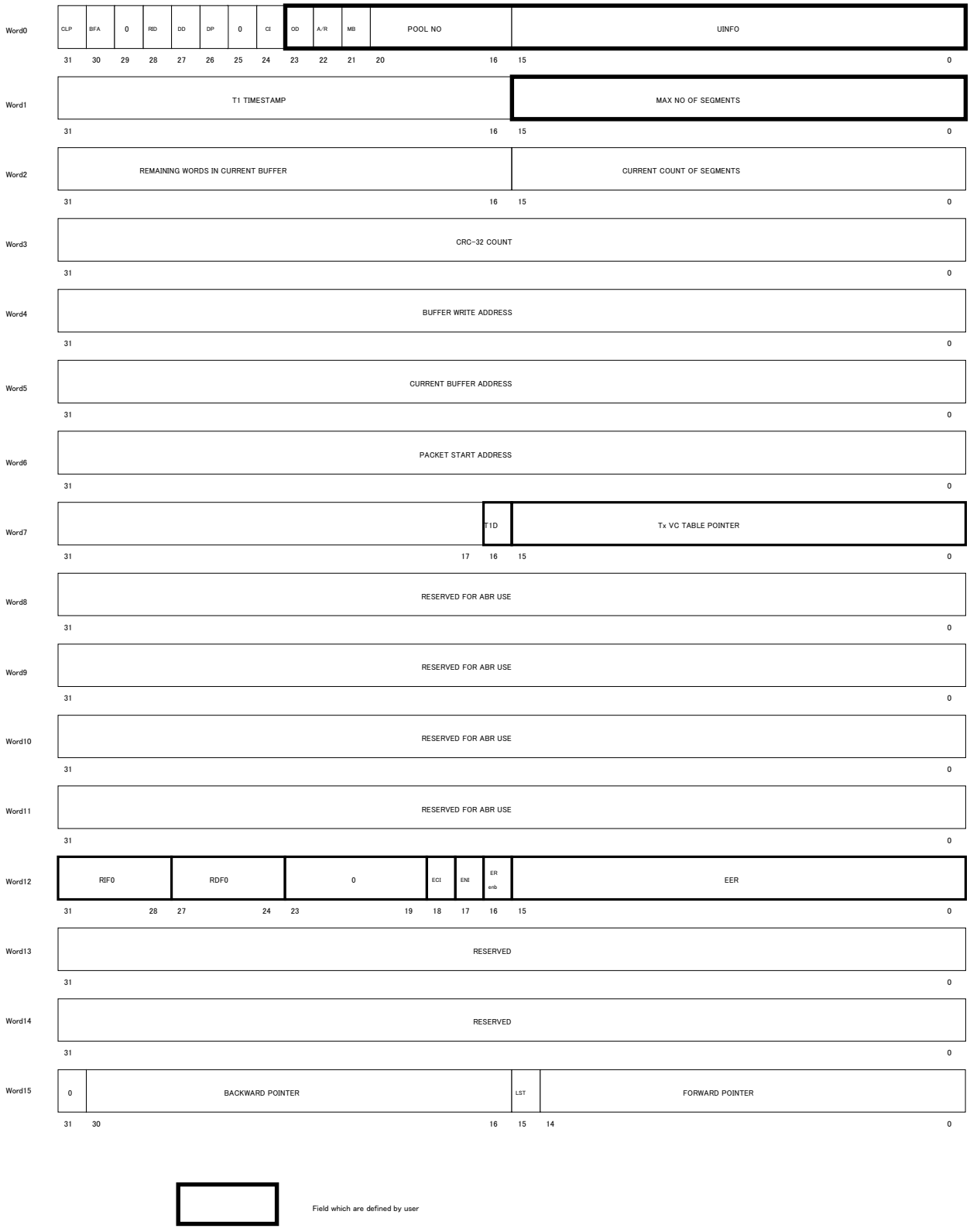
<2> A receive indication is stored into the mailbox and reported with an interrupt.

(h) Reading the receive indication

VR4120A RISC Processor reads the receive indication, updates the read pointer in the mailbox, and extract receive data from the pool.

## (1) Rx VC table

**Figure 4-42. Receive VC Table**



| | |
|---|---|
| CLP | :Set to 1 if the CLP in the header of at least one cell of the packet being received is equal to 1. |
| BFA | :Set to 1 if the free buffer assigned to this VC exists. |
| RID | :Set to 1 if an error occurs while a packet is being received. Then, the subsequent cells of the packet, including the last cell, are discarded. |
| DD | :If a free buffer is not assigned, the cell is discarded and this field is set to 1. |
| DP | :Always 0 because it is not used by ATM Cell Processor. |
| MB | :Mailbox number. |
| POOL NO | :Pool number. (ATM Cell Processor supports pool numbers 0 to 7.) |
| UINFO | :User information |
| T1 TIMESTAMP | :Area used for T1 timer calculation |
| CI | :Congestion notification. |
| OD | :OAM cell reception/discard indication bit. |
| A/R | :AAL-5/RAW cell reception indication bit. |
| MAX.NO.OF SEGMENTS | :Maximum allowable number of segments in one packet |
| REMAINING WORDS IN CURRENT BUFFER | :Number of words of the free area remaining in the current buffer |
| CURRENT COUNT OF SEGMENTS | :Number of segments of the current packet that have been received so far |
| CRC-32 COUNT | :Used for CRC-32 value calculation |
| BUFFER WRITE ADDRESS | :Next address in the currently assigned free buffer that is used for storage |
| CURRENT BUFFER POINTER | :Start address of the free buffer to be assigned next |
| PACKET START ADDRESS | :Start address of the packet (start address of the batch assigned first) |
| Tx VC TABLE POINTER | :Pointer to the Tx VC Table. This field is used in ABR only. |
| WORD 6 to WORD 12 | :These fields are used internally for ABR service. |
| BACKWARD POINTER | :VC Number of the VC linked before this VC in the T1 link list |
| LST | :Set to 1 if the VC is the last one linked in the T1 link list. |
| FORWARD POINTER | :VC Number of the VC linked after this VC in the T1 link list |

#### 4.8.3.2 Non AAL-5 traffic support

Every time ATM Cell Processor receives raw cell, it makes a raw cell data with 53 byte raw cell and 11 byte indication and stores it to the appropriate Rx pool. Then, ATM Cell Processor sets corresponding bits in A_GSR register and generate interruption if not masked.

Since ATM Cell Processor treats raw cells as a unit of cell not a packet, it doesn't stores rx indications in Rx mailbox. When ATM Cell Processor receives raw cells, CRC-10 verify function is always enable. If ATM Cell Processor detects CRC-10 error, sets error bit in raw cell data.

**(1) OAM F5 cell**

When OD bit in VC table is 1 and PTI field in received ATM cell header is "1xx", ATM Cell Processor generates Raw cell data and stores it in pool 0. It sets PCR0=1 in A_GSR register and generates interruption to VR4120A RISC Processor, if not masked. ATM Cell Processor always stores these data in Pool 0. If OD bit is set to 1, Pool 0 has to be set for Raw cell data.

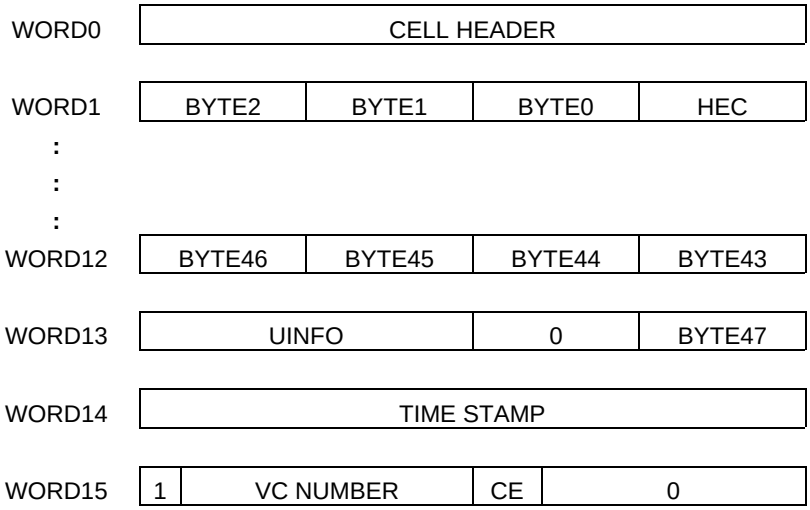
**(2) Non AAL-5 traffic**

When A/R bit in VC table is 0, ATM Cell Processor treats cells belongs to the VC as raw cells. If receives raw cells, ATM Cell Processor has to assign a pool to raw cell data.

**(3) Raw cell data**

The following is the Raw cell data format in little endian mode.

**Figure 4-43. Raw Cell Data Format**




| | |
|---|---|
| Cell Header | :Header of the cell without HEC. |
| HEC | :HEC field pattern of the cell. |
| BYTE0 – BYTE47 | :Payload data of the cell. |
| UINFO | :User information. The pattern which user set in UINFO field in VC table. |
| TIME STAMP | :A_TSR register value when ATM Cell Processor received the cell. |
| VC NUMBER | :VC Number of the cell. |
| CE | :CRC-10 check result<br>0 : No error.<br>1 : CRC-10 error is detected. |

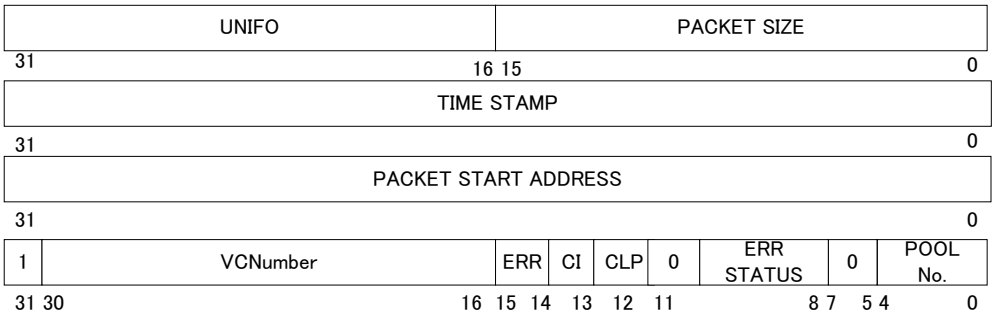
#### 4.8.3.3 Receiving indication

For each packet, ATM Cell Processor writes a receive indication as the reception completion status in the mailbox. The mailbox used for reception is mailbox 0 and 1. More specifically, ATM Cell Processor writes a receive indication once all data (including the trailer) has been stored into the receive buffer if the packet has been received normally; when all data has been stored into the receive buffer if the packet received contains a "CRC-32" or "Length" error; and when an error is detected if the error is other than a "CRC-32" or "Length" error.

A receive indication contains the start address and size of the batch that ATM Cell Processor uses to store the packet, and other information. By reading the receive indication, the host can process the received packet.

The format of a receive indication is as follows:

**Figure 4-44. Receive Indication Format**




| | |
|---|---|
| UINFO | :Pattern set by the host in the UINFO field in the VC table |
| PACKET SIZE | :Size of the receive packet in cell units |
| TIME STAMP | :Value of A_TSR when this indication is issued |
| PACKET START ADDRESS | :Start address of the batch used |
| CHANNEL NUMBER | :VC NUMBER used by this VC |
| ERR | :If 1, indicates that an error occurred while the packet was being received.<br>If 0, indicates that the packet was received normally. |
| CI | :The PTI field in the header of at least one cell of the packet indicated congestion. |
| CLP | :The CLP field in the header of at least one cell of the packet was equal to 1. |
| ERROR STATUS | :Status of the error that occurred |
| | 0000 No error |
| | 0001 Free buffer underflow |
| | 0011 MAX NUMBER OF SEGMENT violation |
| | 0100 CRC-32 error |
| | 0101 User abort |
| | 0110 Length error |
| | 0111 T1 time-out |
| POOL NUMBER | :Number of the pool used |

#### 4.8.3.4 Receive errors

ATM Cell Processor checks for receiving errors while a packet is being received and using trailer information when the packet has been received. Upon detecting an error, it reports the type of error and the start address and the amount of data that had been transferred to SDRAM prior to the error being detected, using the receive indication to be stored in the mailbox.

Upon receiving a receive indication containing the error status, the host takes appropriate action and discards the packet that caused the error. The types of reception errors are described below:

**(1) Free buffer underflow**

This error occurs if the free area in the free buffer in SDRAM is 48 bytes or less when a packet is received. When this error occurs, an A_RQU interrupt is generated. If the discarded cell is an intermediate cell or the last cell of the packet, reception of the packet is suspended, a receive indication reporting the free buffer underflow error is issued, and the RID bit in the VC table is set. The remaining cells of the packet, including the last cell, are discarded, and the RID bit is referenced. When the last cell is received, the RID bit is reset. If the cell discarded due to this error is the first cell of the packet, an A_RQU interrupt is generated and the RID bit is set. No receive indication is issued, however.

**(2) Max No. of segments violation**

This error occurs if the last cell of a packet has not been received when the number of cells received has reached the user-specified "Max. no. of segments." When the next cell is received, the RID bit is set and a receive indication is issued. The subsequent cells of the packet, including the last cell, are discarded.

**(3) CRC-32 error**

This error occurs if the CRC-32 value does not match the CRC-32 value contained in the receive trailer when all the data in the packet has been transferred to SDRAM. If a check made on the trailer reveals that the received packet has both CRC-32 and "length" errors, the CRC-32 error is reported as the error status in the receive indication.

**(4) "length" error**

This error is reported if the "length" field contained in the receive trailer and the calculated packet length satisfy either of the following conditions:

("Number of received cells x 48 bytes" - ""length" value in the trailer") > 55 bytes

("Number of received cells x 48 bytes" - ""length" value in the trailer") < 8 bytes

**(5) T1 time-out**

This error occurs if the last cell of a packet has not been received even after the user-specified A_T1R time has elapsed since the first cell was received. When this error occurs, the RID bit is set and the remaining cells of the packet that caused this error, including the last cell, are discarded.

The table below lists the errors that can occur with any of the first cell, the immediate cells, and the last cell of a packet.

**Table 4-4. Reception Errors That Can Occur During Packet Reception**

| Error | When a cell is discarded | When a receive indication is issued | Handling of other cells after the error occurred |
|---|---|---|---|
| Free buffer underflow | The cell received when the size of the free buffer is insufficient is discarded. | Time at which transfer becomes impossible during segment transfer | The subsequent cells of the packet, including the last cell, are discarded. |
| MAX No. of Segments error | The cell received after MAX No. of Segments was reached is discarded. | Time at which the next cell is received after MAX No. of Segments was reached | The subsequent cells of the packet, including the last cell, are discarded. |
| T1 error | The cell received after the T1 time has elapsed is discarded. | Time at which T1 time has elapsed. | The subsequent cells of the packet, including the last cell, are discarded. |

Two or more errors may occur at the same time; however, only one is reported with a receive indication. The table below lists the error reporting priorities.

**Table 4-5. Error Reporting Priorities**

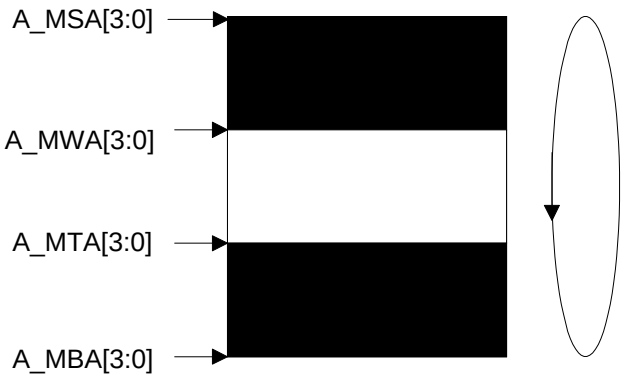
| | Underflow | MAX error | CRC error | Length | T1 error |
|---|---|---|---|---|---|
| Underflow | - | Underflow | Underflow | Underflow | Underflow |
| MAX error | - | - | MAX error | MAX error | MAX error |
| CRC error | - | - | - | CRC error | CRC error |
| Length | - | - | - | - | Length |
| T1 error | - | - | - | - | - |

### 4.8.4 Mailbox

ATM Cell Processor uses mailboxes as ring buffers in SDRAM. The structure of a mailbox and the defined addresses are as follows.

Mailbox start address (A_MSA[3:0]) :The start address of the mailbox
Mailbox bottom address (A_MBA[3:0]) :The bottom address of the mailbox (address following the last address)
Mailbox write address (A_MWA[3:0]) :The write pointer
Mailbox tail address (A_MTA[3:0]) :The tail address that has been read by the host and which is to be updated

**Figure 4-45. Mailbox Structure**




Upon writing an indication, increments the write pointer (A_MWA[3:0]), sets the MM bit for the corresponding mailbox in the A_GSR register, and issues an interrupt if it is not masked. When updating the write pointer (A_MWA[3:0]), ATM Cell Processor causes A_MWA[3:0] to jump to the start address (A_MSA[3:0]) if A_MWA[3:0] has reached the bottom address (A_MBA[3:0]). To read an indication, VR4120A RISC Processor uses the read pointer (A_MTA[3:0]). A_MTA[3:0] is managed by the host: Each time VR4120A RISC Processor reads an indication from the mailbox, it writes the address of the next indication to the read pointer (A_MTA[3:0]). If the write pointer (A_MWA[3:0]) points to the same address as that pointed to by the read pointer (A_MTA[3:0]), sets the MF bit of the A_GSR register that corresponds to the mailbox to indicate that the mailbox is full (the MF state), and issues an interrupt if it is not masked. Once the mailbox enters the MF state, ATM Cell Processor does not execute any commands.

### 4.8.5 ABR function

ATM Cell Processor supports ABR service which is defined in ATM Forum “Traffic Management Specification Ver.4.0.”.

#### 4.8.5.1 Source / destination behavior

**(1) Source behavior**

(a) The value of ACR never exceeds PCR, nor be less than MCR. ATM Cell Processor never sends in-rate cells at a rate exceeding ACR. It always sends in-rate cells at a rate less than or equal to ACR.

(b) Before ATM Cell Processor sends the first cell after connection setup, it sets ACR to ICR. The first in-rate cell to be sent is a forward RM-cell.

(c) After the first in-rate forward RM-cell, ATM Cell Processor sends in-rate in the following order.

<1> The next in-rate cell is a forward RM-cell if and only if, since the last in-rate forward RM-cell was sent, either: at least Mrm in-rate cells have been sent and at least Trm time has elapsed, or Nrm-1 in-rate cells have been sent.

<2> The next in-rate cell is a backward RM-cell if condition <1> above is not met, if a backward RM-cell is waiting for transmission, and if either; no in-rate backward RM-cell has been sent since the last in-rate forward RM-cell, or no data cell is waiting for transmission.

<3> The next in-rate cell to be sent is a data cell if neither condition <1> nor condition <2> above is met, and if a data cell is waiting for transmission.

(d) Cells sent in accordance with procedure #1, #2, and #3 have CLP=0.

(e) Before sending a in-rate forward RM-cell, if ACR>ICR and the time T that has elapsed since the last in-rate forward RM-cell was sent is greater than ADTF, then ATM Cell Processor reduces ACR to ICR.

(f) Before sending an in-rate forward RM-cell and after following procedure #5 above, ATM Cell Processor adjusts ACR again. If at least CRM in-rate forward RM-cells have been sent since the last backward RM-cell with BN=0 was received, then ATM Cell Processor reduces ACR by at least ACR*CDF. If this reduction makes ACR below MCR, it sets ACR to MCR.

(g) After following procedure #5 and #6 above, ATM Cell Processor places the new ACR value in the CCR field of the outgoing forward RM-cell.

(h) When a backward RM-cell is received with CI=1, then ATM Cell Processor reduces ACR by ACR*RDF. If this reduction makes ACR below MCR, ATM Cell Processor sets ACR to MCR. If the backward RM-cell has both CI=0 and NI=0, then ATM Cell Processor increases ACR by RIF*PCR. ATM Cell Processor sets ACR to PCR if the result of the calculation above exceeds PCR.

(i) When a backward RM-cell is received (in-rate or out-of-rate), and after ACR is adjusted according to procedure #8, ATM Cell Processor sets ACR to the minimum of ACR as computed in procedure #8, and the ER field. But ATM Cell Processor doesn’t sets ACR less than MCR. If the backward RM-cell has NI=1, the ATM Cell Processor doesn’t increase ACR.

(j) When ATM Cell Processor generates a forward RM-cell (in-rate or out-of-rate), it assigns values to the various RM-cell fields as specified for source-generated cells in Section. 4.8.4.2

(k) Forward RM-cells may be sent out-of-rate (i.e. not conforming to the current ACR). Out-of-rate forward RM-cells shall not be sent at a rate greater than TCR.

(l) ATM Cell Processor resets EFCI on every data cell it sends.

**(2) Destination behavior**

(a) When ATM Cell Processor receives a data cell, it saves EFCI indicator as the EFCI state of the connection.

(b) When ATM Cell Processor receives a forward RM-cell, it turns around the cell to return to the source. ATM Cell Processor changes the DIR bit in the RM-cell from "forward" to "backward", sets BN to zero, and leaves CCR, MCR, ER, CI, and NI fields in the RM-cell unchanged except:

<1> If a saved EFCI state is set, then ATM Cell Processor sets CI=1 in the RM-cell, and resets the saved EFCI state.

<2> When the number of cells in Rx FIFO exceeds the threshold, ATM Cell Processor sets CI=1 in all backward RM-cell.

<3> When Rx pool becomes empty, ATM Cell Processor sets CI=1 in all backward RM-cell of VC which uses this Rx pool.

(c) If ATM Cell Processor receives a forward RM-cell while another turn-around RM-cell (on the same connection) is scheduled for in-rate transmission:

<1> ATM Cell Processor overwrites the contents of the old cell with new cell.

<2> If there is no RM-cell waiting for out-of-rate transmission, ATM Cell Processor sends the new backward RM cell out-of-rate.

<3> If there is already an out-of-rate RM cell waiting, ATM Cell Processor replaces it with new RM cell.

#### 4.8.5.2 RM cell format

**Figure 4-46. RM Cell Format**

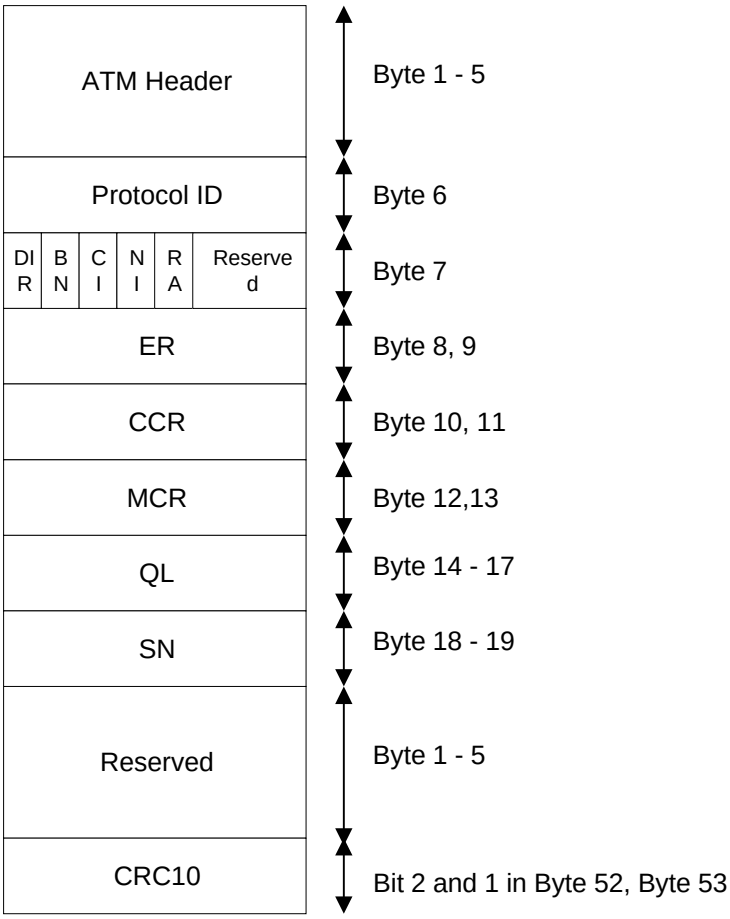


**Table 4-6. RM Cell Initial Value**

| Field | Function | FRM cell | BRM cell* |
|---|---|---|---|
| Header | ATM header. | PTI=110 | PTI=110 |
| ID | The Protocol ID. | 00000001 | 00000001 |
| DIR | Direction. | 0 | 1 |
| BN | Backwards Explicit Congestion Notification(BECN) | 0 | 1 |
| CI | Congestion Indication<br>1: Congestion<br>0: No congestion | 0 | 0 or 1 |
| NI | No Increase bit.<br>1: Increase is inhibited.<br>0: Increase is permitted. | 0 | 0 or 1 |
| RA | Not used for ATM Forum ABR.<br>(Request / Acknowledge bit) | 0 | 0 |
| Reserved in Octet7 | Reserved field. | 0 | 0 |
| ER | Explicit Rate. | PCR<br>(VC table) | Any value |
| CCR | Current Cell Rate. | ACR<br>(VC table) | 0 |
| MCR | Minimum Cell Rate. | MCR<br>(VC table) | 0 |
| QL | Not used for ATM Forum ABR.<br>(Queue Length) | 0 | 0 |
| SN | Not used for ATM Forum ABR.<br>( Sequence Number) | 0 | 0 |
| Reserved in Octet 22-51 | Reserved field. | 6AH for each octet | 6AH for each octet |
| Reserved in octet 52 | Reserved field | 0 | 0 |
| CRC-10 | CRC-10 | CRC-10 | CRC-10 |

This BRM cell is a destination *generated* cell. The BRM cell information which destination only turns around is described in Section. 4.8.4.1.

#### 4.8.5.3 ABR parameter

**Table 4-7. ABR Parameter**

| Label | Description | Units and Range |
|---|---|---|
| PCR | The Peak Cell Rate. | In Cells/Sec, See **Note** for Range |
| MCR | The Minimum Cell Rate. | In Cells/Sec, See **Note** for Range |
| ICR | The Initial Cell Rate. | In Cells/Sec, See **Note** for Range |
| RIF | Rate Increase Factor, RIF, controls the amount by which the cell transmission rate may increase upon receipt of an RM-cell. | RIF is power of two, ranging from 1/32768 to 1. |
| Nrm | Nrm is the maximum number of cells a source may send for each forward RM-cell. | Power of 2<br>Range: 2 to 256 cells |
| Mrm | Mrm controls allocation of bandwidth between forward RM-cells, backward RM-cells, and data cells. | Power of 2<br>range: 1 to 8 cells |
| RDF | The Rate Decrease Factor, RDF, controls the decrease in the cell transmission rate. | RDF is from 2 to 1/32,768 |
| ACR | The Allowed Cell Rate, ACR, is the current rate at which a source is allowed to send. | In Cells/Sec, See **Note** for Range |
| CRM | Missing RM-cell count. CRM limits the number of forward RM-cells which may be sent in the absence of received backward RM-cells. | Units: cells.<br>Integer. |
| ADTF | The ACR Decrease Time Factor is the time permitted between sending RM-cells before the rate is decreased to ICR. | Units: seconds<br>ADTF range: .01 to 10.23sec with granularity of 10ms. |
| Trm | Trm provides an upper bound on the time between forward RM-cells for an active source. | Trm is 100 times a power of two range: $100*2^{-7}$ to $100*2^{0}$ |
| FRTT | The Fixed Round Trip Time, FRTT, is the sum of the fixed and propagation delays from the source to the furthest destination and back. | Units 10 microseconds<br>Range: 0 to 167 seconds |
| TBE | Transient Buffer Exposure, TBE, is the negotiated number of cells that the network would like to limit the source to sending during idle startup period, before the first RM-cell returns. | Units: Cells<br>Range: 0 to 16,777,215 |
| CDF | The Cutoff Decrease Factor, CDF, controls the decrease in ACR associated with CRM. | CDF is zero, or power of two in the range 1/64 to 1. |
| TDF, ADDF | The Timeout Decrease Factor, TDF, controls the decrease in ACR associated with TOF. TDF is signaled as ADDF(ACR Decay Decrease Factor), where TDF = ADDF times the smallest power of 2 in cells/second greater than or equal to ICR in cells/second. | TDF is in units of 1/seconds. ADDF is either zero or a power of two in the range $2^{-5}$ to $2^{-19}$ |
| PNI | The Permit Next Increase flag, PNI. PNI=1 disables source behavior 5b. PNI=0 enables source behavior 5b. | 0 or 1. |
| TCR | The Tagged Cell Rate, TCR, limits the rate at which a source may send out-of-rate forward RM cells. | TCR is a constant fixed at 10 cells/second |

**Note** Rate field is expressed as below.

Rate = $2^{e}*(1+m/512)*nz$ cells/sec

**Figure 4-47. Rate Field ABR Parameter**

#### 4.8.5.4 ABR service operation

(1) Sets registers for ABR service

(2) Open_Channel

In ABR service, since ATM Cell Processor uses both Tx VC table and Rx VC table, VR4120A RISC Processor has to open two channel. In ABR service, although ATM Cell Processor only transmits data and not receives any data, receiving VC is required to receive BRM cell. The other way, although ATM Cell Processor only receives data, transmitting VC is required to transmit FRM cell.

(3) Setting VC tables

As described above, in ABR service, both transmit and receive channels are always used together. So that, both transmit and receive VC tables have to be set before transmitting ABR cells or receiving ABR cells. There are special parameters for ABR service. Most of these parameters are obtained from connection negotiation.

(4) Setting Receive lookup table

In ABR service, although ATM Cell Processor only transmits cells and not receives, it has to receive BRM cells. If ATM Cell Processor only receives BRM cells and not receives any data cells, VR4120A RISC Processor doesn't have to reserve receiving pool, but if ATM Cell Processor receives data cells, VR4120A RISC Processor has to reserve receiving pool.

(5) Tx_Ready command

The operation for this command is as same as the ones for other service classes.

(6) Transmit and Receive indications

The operation for this command is as same as the ones for other service classes.

(7) Close_Channel

In ABR service, both transmit and receive channels have to be closed at a time. VR4120A RISC Processor has to close transmit channel first and then receive channel.

#### 4.8.5.5 RM cell operation

ATM Cell Processor manages RM cells internally. VR4120A RISC Processor doesn't have to do anything on RM cells. ATM Cell Processor manages RM cells according to source/destination behavior.

- RM cell operation at the source side

  Transmit FRM cells periodically. ATM Cell Processor also transmit out-of-rate FRM cells with TCR. If ATM Cell Processor receives BRM cell, it recalculate ACR value using information in BRM cell. VR4120A RISC Processor doesn't have to reserve FRM cell data in transmit buffer in SDRAM.

- RM cell operation at the destination side

When ATM Cell Processor receives FRM cell, it automatically send back BRM cell. ATM Cell Processor follows destination behavior to send back BRM cell. Received FRM cell is not stored in receiving pool in SDRAM. ATM Cell Processor processes it internally.

**[MEMO]**

# CHAPTER 5 ETHERNET CONTROLLER

## 5.1 Overview

This section describe Ethernet communication block. This Ethernet Controller block comprises of a 10/100Mbps Ethernet MAC (Media Access Control), data transmit/receive FIFOs, DMA and NEC original high speed bus interface.

The $\mu$PD98501 implements 2-channel Ethernet Controller.

### 5.1.1 Features

- IEEE802.3/802.3u/802.3x Compliant:
  - 10/100Mbps Ethernet MAC
  - Media Independent Interface (MII)
  - Flow Control
- Full duplex operation for 10Mbps and 100Mbps
- Address Filtering:
  - unicast
  - multicast
  - broadcast
- Statistics counters for management information
  - RMON
  - SNMP MIB
- Large independent receive and transmit FIFOs
- Direct Memory Access (DMA) with programmable burst size providing for low CPU utilization
- Internal Bus interface (IBUS): 32bit @66MHz

### 5.1.2 Block diagram of Ethernet controller block

The following list describes this block hardware components, and Figure 5-1 shows a block diagram of this block:

**IBUS Interface** - Data transfer is done via this IBUS interface between CPU and Ethernet Controller or memory and Ethernet Controller. This performance is approximately 2Gbps (32 bit x 66MHz). Also in this block, IBUS protocol operation is done [retry, disconnect and so on].

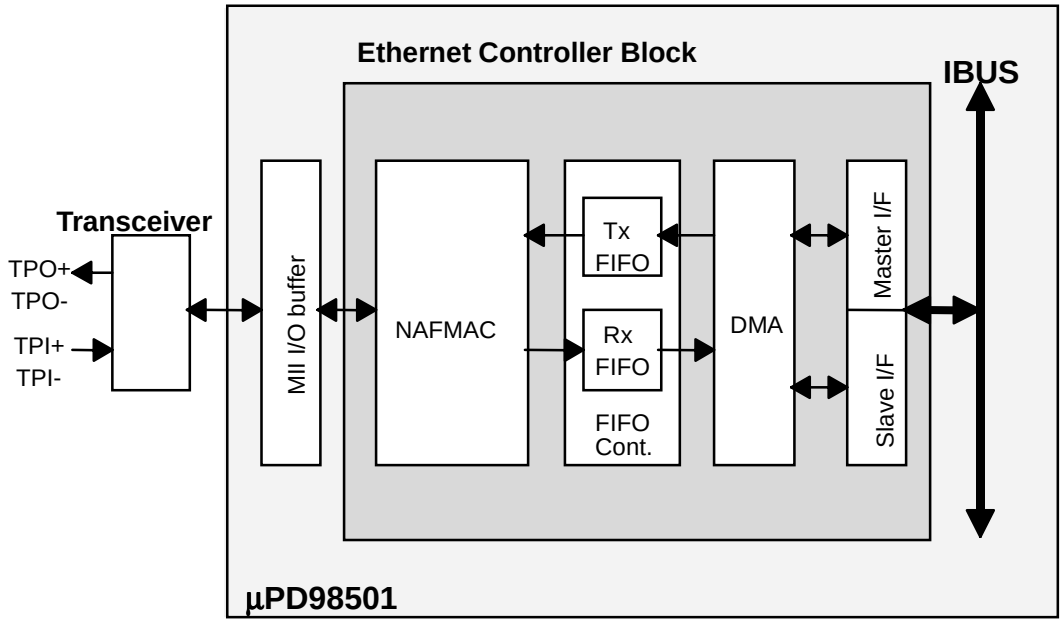
**DMA Controller** - DMA controller contains dual reception and transmission controller, which handle data transfers between memory and FIFOs. This DMA controller supports a byte alignment.

**FIFO Controller** - FIFO controller contains two independent FIFOs for reception and transmission. This controller supports automatic packet deletion on receive (after a collision or address filtering) and packet retransmission after a collision on transmit.

**MAC** - MAC is fully compliant with IEEE802.3, IEEE802.3u and IEEE802.3x.

**MII I/O Buffer** - MII I/O buffers are compliant IEEE802.3u and are connect to standard PHY device.

**Figure 5-1. Block Diagram of Ethernet Controller**



## 5.2 Register

Register of this block has following four category. Please show in Table 5-1.

VR4120A RISC Core controls following Register sets via System Controller and IBUS.

The μPD98501 has 2-channel Ethernet Controller, #1 controller's base address is 1000_2000H, #2 controller's base address is 1000_3000H.

**Table 5-1. Ethernet Controller's Register Map**

| Offset Address | Descriptions |
|---|---|
| 00H – 13FH | MAC Control Registers |
| 140H – 1FFH | Statistics Counter Registers |
| 200H – 233H | DMA and FIFO Management Registers |
| 234H – 23FH | Interrupt and Configuration Registers |

MAC Control Registers map is shown in Table 5-2.

**Table 5-2. MAC Control Register Map**

| Address | Register | Description | R/W | Default |
|---|---|---|---|---|
| 00H | En_MACC1 | MAC configuration register 1 | R/W | 0000_0000H |
| 04H | En_MACC2 | MAC configuration register 2 | R/W | 0000_0000H |
| 08H | En_IPGT | Back-to-Back IPG register | R/W | 0000_0013H |
| 0CH | En_IPGR | Non Back-to-Back IPG register | R/W | 0000_0E13H |
| 10H | En_CLRT | Collision register | R/W | 0000_370FH |
| 14H | En_LMAX | Max packet length register | R/W | 0000_0600H |
| 18H-1CH | N/A | Reserved for future use | - | - |
| 20H | En_RETX | Retry count register | R/W | 0000_0000H |
| 24H-50H | N/A | Reserved for future use | - | - |
| 54H | En_LSA2 | Station Address register 2 | R/W | 0000_0000H |
| 58H | En_LSA1 | Station Address register 1 | R/W | 0000_0000H |
| 5CH | En_PTVR | Pause timer value read register | R | 0000_0000H |
| 60H | N/A | Reserved for future use | - | - |
| 64H | En_VLTP | VLAN type register | R/W | 0000_0000H |
| 80H | En_MIIC | MII configuration register | R/W | 0000_0000H |
| 84H-90H | N/A | Reserved for future use | - | - |
| 94H | En_MCMD | MII command register | W | 0000_0000H |
| 98H | En_MADR | MII address register | R/W | 0000_0000H |
| 9CH | En_MWTD | MII write data register | R/W | 0000_0000H |
| A0H | En_MRDD | MII read data register | R | 0000_0000H |
| A4H | En_MIND | MII indicator register | R | 0000_0000H |
| A8H-C4H | N/A | Reserved for future use | - | - |
| C8H | En_AFR | Address Filtering register | R/W | 0000_0000H |
| CCH | En_HT1 | Hash table register 1 | R/W | 0000_0000H |
| D0H | En_HT2 | Hash table register 2 | R/W | 0000_0000H |
| D4H-D8H | N/A | Reserved for future use | - | - |
| DCH | En_CAR1 | Carry register 1 | R/W | 0000_0000H |
| E0H | En_CAR2 | Carry register 2 | R/W | 0000_0000H |
| E4H-12CH | N/A | Reserved for future use | - | - |
| 130H | En_CAM1 | Carry mask register 1 | R/W | 0000_0000H |
| 134H | En_CAM2 | Carry mask register 2 | R/W | 0000_0000H |
| 138H-13CH | N/A | Reserved for future use | - | - |

**Remarks 1.** These reserve registers must not be accessed.

**2.** n =1, 2

n = 1: Ethernet Controller #1,

n = 2: Ethernet Controller #2

### 5.2.1 Statistics counter registers

MAC gathers statistical information about the receive and transmit operations from the statistics counters.

Each counter consists of 32 bits counter. When a counter overflow condition occurs, the corresponding bit of the En_CAR1 register or En_CAR2 register is set to 1, and it can generate an interrupt. The interrupt can be masked by setting the bits of En_CAM1 register or En_CAM2 register.

Some time is required to complete the updating of all statistics counters after the change of the status vector (TSV or RSV) because of the count operation.

**Table 5-3. Statistics Counter Register Map**

(1/2)

| Address | Register | Description | R/W |
|---|---|---|---|
| 140H | En_RBYT | Receive Byte Counter | R/W |
| 144H | En_RPKT | Receive Packet Counter | R/W |
| 148H | En_RFCS | Receive FCS Error Counter | R/W |
| 14CH | En_RMCA | Receive Multicast Packet Counter | R/W |
| 150H | En_RBCA | Receive Broadcast Packet Counter | R/W |
| 154H | En_RXCF | Receive Control Frame Packet Counter | R/W |
| 158H | En_RXPF | Receive PAUSE Frame Packet Counter | R/W |
| 15CH | En_RXUO | Receive Unknown OP code Counter | R/W |
| 160H | En_RALN | Receive Alignment Error Counter | R/W |
| 164H | En_RFLR | Receive Frame Length Out of Range Counter | R/W |
| 168H | En_RCDE | Receive Code Error Counter | R/W |
| 16CH | En_RFCR | Receive False Carrier Counter | R/W |
| 170H | En_RUND | Receive Undersize Packet Counter<br>This counter is incremented each time a frame is received which is less than 64 bytes in length and contains a valid FCS. | R/W |
| 174H | En_ROVR | Receive Oversize Packet Counter<br>This counter is incremented each time a frame is received which exceeds 1518 bytes length and contains a valid FCS. | R/W |
| 178H | En_RFRG | Receive Error Undersize Packet Counter<br>This counter is incremented each time a frame is received which is less than 64 bytes in length and contains a invalid FCS, includes alignment error. | R/W |
| 17CH | En_RJBR | Receive Error Oversize Packet Counter<br>This counter is incremented each time a frame is received which exceeds 1518 bytes length and contains an invalid FCS or includes an alignment error. | R/W |
| 180H | En_R64 | Receive 64 Byte Frame Counter<br>This counter is incremented for each good or bad frame received which is 64 bytes in length. | R/W |
| 184H | En_R127 | Receive 65 to 127 Byte Frame Counter<br>This counter is incremented for each good or bad frame received which is 65 to 127 bytes in length. | R/W |

**Remark** n =1, 2

n = 1: Ethernet Controller #1,

n = 2: Ethernet Controller #2

(2/2)

| Address | Register | Description | R/W |
|---|---|---|---|
| 188H | En_R255 | Receive 128 to 255 Byte Frame Counter<br>This counter is incremented for each good or bad frame received which is 128 to 255 bytes in length. | R/W |
| 18CH | En_R511 | Receive 256 to 511 Byte Frame Counter<br>This counter is incremented for each good or bad frame received which is 256 to 511 bytes in length. | R/W |
| 190H | En_R1K | Receive 512 to 1023 Byte Frame Counter<br>This counter is incremented for each good or bad frame received which is 512 to 1023 bytes in length. | R/W |
| 194H | En_RMAX | Receive Over 1023 Byte Frame Counter<br>This counter is incremented for each good or bad frame received which is over 1023 bytes in length. | R/W |
| 198H | En_RVBT | Receive Valid Byte Counter<br>This counter is incremented by the byte count of each valid packet received. | R/W |
| 1C0H | En_TBYT | Transmit Byte Counter | R/W |
| 1C4H | En_TPCT | Transmit Packet Counter | R/W |
| 1C8H | En_TFCS | Transmit CRC Error Packet Counter | R/W |
| 1CCH | En_TMCA | Transmit Multicast Packet Counter | R/W |
| 1D0H | En_TBCA | Transmit Broadcast Packet Counter | R/W |
| 1D4H | En_TUCA | Transmit Unicast Packet Counter | R/W |
| 1D8H | En_TXPF | Transmit PAUSE control Frame Counter | R/W |
| 1DCH | En_TDFR | Transmit Single Deferral Packet Counter | R/W |
| 1E0H | En_TXDF | Transmit Excessive Deferral Packet Counter | R/W |
| 1E4H | En_TSCL | Transmit Single Collision Packet Counter | R/W |
| 1E8H | En_TMCL | Transmit Multiple collision Packet Counter | R/W |
| 1ECH | En_TLCL | Transmit Late Collision Packet Counter | R/W |
| 1F0H | En_TXCL | Transmit Excessive Collision Packet Counter | R/W |
| 1F4H | En_TNCL | Transmit Total Collision Counter | R/W |
| 1F8H | En_TCSE | Transmit Carrier Sense Error Counter<br>This counter is incremented for each frame transmitted which carrier sense error occurs during transmission. | R/W |
| 1FCH | En_TIME | Transmit Internal MAC Error Counter<br>This counter is incremented for each frame transmitted in which an internal MAC error occurs during transmission. | R/W |

**Remark** n =1, 2

n = 1: Ethernet Controller #1,

n = 2: Ethernet Controller #2

### 5.2.2 DMA and FIFO management registers

These registers control to transfer receive and transmit data by internal DMAC of this block.

**Table 5-4. DMA and FIFO Management Registers Map**

| Address | Register | Description | R/W | Default |
|---|---|---|---|---|
| 200H | En_TXCR | Transmit Configuration Register | R/W | 0000_0000H |
| 204H | En_TXFCR | Transmit FIFO Control Register | R/W | FFFF_40C0H |
| 208H | En_TXDTR | Transmit Data Register | W | 0000_0000H |
| 20CH | En_TXSR | Transmit Status Register | R | 0000_0000H |
| 210H | N/A | Reserved for future use | - | - |
| 214H | En_TXDPR | Transmit Descriptor Register | R/W | 0000_0000H |
| 218H | En_RXCR | Receive Configuration Register | R/W | 0000_0000H |
| 21CH | En_RXFCR | Receive FIFO Control Register | R/W | C040_0040H |
| 220H | En_RXDTR | Receive Data Register | R | 0000_0000H |
| 224H | En_RXSR | Receive Status Register | R | 0000_0000H |
| 228H | N/A | Reserved for future use | - | - |
| 22CH | En_RXDPR | Receive Descriptor Register | R/W | 0000_0000H |
| 230H | En_RXPDR | Receive Pool Descriptor Register | R/W | 0000_0000H |

**Remark** n =1, 2

n = 1: Ethernet Controller #1,

n = 2: Ethernet Controller #2

### 5.2.3 Interrupt and configuration registers

These register control interrupt occur and configuration for this block.

**Table 5-5. Interrupt and Configuration Registers Map**

| Address | Register | Description | R/W | Default |
|---|---|---|---|---|
| 234H | En_CCR | Configuration Register | R/W | 0000_0000H |
| 238H | En_ISR | Interrupt Service Register | R | 0000_0000H |
| 23CH | En_MSR | Mask Serves Register | R/W | 0000_0000H |

**Remark** n =1, 2

n = 1: Ethernet Controller #1,

n = 2: Ethernet Controller #2

### 5.2.4 Detail of MAC control registers

#### 5.2.4.1 En_MACC1-MAC Configuration Register 1 (00H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:15 | - | Reserved for future use. Write as 0 | - |
| 14 | MACLB | MAC loopback:<br>Setting this bit to 1 switches into loopback mode which routes from Transmit Function Block to Receive Function Block. | 0 |
| 13:12 | - | Reserved for future use. Write as 0 | - |
| 11 | TXFC | Transmit flow control enable:<br>Setting this bit to 1 enables to transmit the pause control frame. | 0 |
| 10 | RXFC | Receive flow control enable:<br>Setting this bit to 0 enables MAC to execute PAUSE operation for the pause time on the setting of the pause timer. The setting of the pause timer must be updated every time a valid PAUSE control frame is received regardless of the setting of this bit. | 0 |
| 9 | SRXEN | Receive enable:<br>Setting this bit to 1 enables the function of the MAC Address field filtering. | 0 |
| 8 | PARF | Control packet pass:<br>Setting this bit to 1 allows MAC to check for all received packets including control frames. When this bit is set to 0, MAC does not check the reception of a control frame. | 0 |
| 7 | PUREP | Pure preamble:<br>Setting this bit to 1 allows the recognition of start of a new packet only when a pure preamble ( “0101” only ) is seen. | 0 |
| 6 | FLCHT | Length field check:<br>When this bit is set to 1, MAC compares the length field value in the packet to the actual packet length, and indicates the result in the status vector. When this bit is set 0, MAC does not check the Frame length field. | 0 |
| 5 | NOBO | No Back Off:<br>When this bit is set to 1, MAC always transmits the packet with no backoff operation. | 0 |
| 4 | - | Reserved for future use. Write as 0 | - |
| 3 | CRCEN | CRC append enable:<br>Setting this bit to 1 enables MAC to append calculated CRC code to the end of transmitted packet automatically. | 0 |
| 2 | PADEN | PAD append enable:<br>Setting this bit to 1 enables MAC to append PAD when the transmitted packet length is less than 64 octets because the input data (TPD) length is less than 60 bytes before appending the CRC. | 0 |
| 1 | FULLD | Full duplex enable:<br>Setting this bit to 1 enables the full duplex operation. | 0 |
| 0 | HUGEN | Large packet enable:<br>Setting this bit to 1 enable MAC to transmit and receive a packet of which length is over the value of En_LMAX register. | 0 |

### 5.2.4.2 En_MACC2-MAC Configuration Register 2 (04H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:11 | - | Reserved for future use. Write as 0 | - |
| 10 | MCRST | MAC Control Block software reset:<br>Setting this bit to 1 forces MAC Control Block to a software reset operation.<br>In order to complete the software reset state, this bit needs to be cleared. | 0 |
| 9 | RFRST | Receive Function Block software reset:<br>Setting this bit to 1 forces the Receive Function Block to a software reset operation. In order to complete the software reset state, this bit needs to be cleared. | 0 |
| 8 | TFRST | Transmit Function Block software reset:<br>Setting this bit to 1 forces Transmit Function Block to a software reset operation. In order to complete the software reset state, this bit needs to be cleared. | 0 |
| 7 | - | Reserved for future use. Write as 0. | - |
| 6 | BPNB | Back Pressure No Back Off:<br>When this bit is set to 1, MAC transmits the packet with no backoff operation after the back pressure operation only. | 0 |
| 5 | APD | Auto VLAN PAD:<br>When this bit is set to 1, if the ETPID field in the current transmit packet matches the value in the En_VLTP register , MAC pads the current packet as a VLAN packet when the current packet should be padded . | 0 |
| 4 | VPD | VLAN PAD mode:<br>When this bit is set to 1, MAC always pads the current packet as a VLAN packet when the current packet should be padded. | 0 |
| 3:2 | - | Reserved for future use. Write as 0. | - |

### 5.2.4.3 En_IPGT-Back to Back IPG Register (08H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:7 | - | Reserved for future use. Write as 0 | - |
| 6:0 | IPGT | Back-To-Back IPG :<br>This field sets IPG when transmit operation is executed with Back-to-back mode. The formula for the Back-to-Back IPG is:<br>IPG = (5 + IPGT) × 4 bits time | 13H |

#### 5.2.4.4 En_IPGR-Non Back to Back IPG Register (0CH R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:15 | - | Reserved for future use. Write as 0 | - |
| 14:8 | IPGR1 | Carrier sense time:<br>This field sets the carrier sense time when the transmit operation is executed in non Back-to-Back mode. The formula for the carrier sense time is:<br>Carrier sense time = (2 + IPGR1) × 4 bits time<br>The carrier sense time defined in this field is included in the non Back-to-Back IPG defined in the IPGR2 field. | 0EH |
| 7 | - | Reserved for future use. Write as 0 | - |
| 6:0 | IPGR2 | Non Back-To-Back IPG:<br>This field sets IPG when transmit operation is executed with non Back-to-Back mode. The formula for the non Back-to-Back IPG is:<br>IPG = (5 + IPGR2) × 4 bits time | 13H |

#### 5.2.4.5 En_CLRT-Collision Register (10H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:14 | - | Reserved for future use. Write as 0 | - |
| 13:8 | LCOL | Late collision window:<br>This fields sets collision window size.<br>The formula for the collision window size is:<br>collision window size = (LCOL + 8) × 8 bits time | 38H |
| 7:4 | - | Reserved for future use. Write as 0 | - |
| 3:0 | RETRY | Maximum number of retry:<br>This field sets the maximum number of retries during the transmission. If a retry occurs beyond this value, the current transmission is aborted. | 0FH |

#### 5.2.4.6 En_LMAX-Maximum Packet Length Register (14H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:16 | - | Reserved for future use. Write as 0 | - |
| 15:0 | MAXF | Maximum packet length:<br>This field sets the maximum the length of transmit and receive packet by the octet when En_MACC1 register: HUGEN bit is set to 1.<br>Receive: If the current receive packet length is greater than the value<br>of MAXF, MAC terminates the reception.<br>Transmit: If the current transmit packet length is greater than the value<br>of MAXF, MAC aborts the transmission. | 600H |

#### 5.2.4.7 En_RETX-Retry Counter Register (20H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:4 | - | Reserved for future use. Write as 0 | - |
| 3:0 | RETX | Retry counter:<br>This field indicates the number of the retries during the current transmission. | 0 |

#### 5.2.4.8 En_LSA2-Station Address Register (54H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:16 | - | Reserved for future use. Write as 0 | - |
| 15:0 | LSA2 | Station address SA (47:32) | 0 |

#### 5.2.4.9 En_LSA1-Station Address Register (58H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:0 | LSA1 | Station address SA(31:0):<br>This field sets the station address. The station address: SA(47:0) is used by the MAC Control Block as the source address in the PAUSE control frame, and used for the comparison with the destination address of the receive packet. | 0 |

#### 5.2.4.10 En_PTVR-Pause Timer read Register (5CH R)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:16 | - | Reserved for future use. | - |
| 15:0 | PTCT | Pause timer counter:<br>This field indicates the current pause timer value. | 0 |

#### 5.2.4.11 En_VLTP-VLAN Type Register (64H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:16 | - | Reserved for future use. Write as 0 | - |
| 15:0 | VLTP | VLAN type: This field sets ETPID value.<br>Receive : Detects a VLAN frame by comparing this field with the<br>ETPID field in the received packet.<br>Transmit : If En_MACC2 register: APD bit is set to 1 and the ETPID<br>field in the current transmit packet matches this field, MAC pads as a VLAN packet when the current packet should be padded | 0 |

### 5.2.4.12 En_MIIC-MII Configuration Register (80H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:16 | - | Reserved for future use. Write as 0 | - |
| 15 | MIRST | MII Management Interface Block software reset:<br>Setting this bit to 1 forces the MII Management Interface Block to a software reset operation.<br>In order to complete the software reset state, this bit needs to be cleared. | 0 |
| 14:4 | - | Reserved for future use. Write as 0 | - |
| 3:2 | CLKS | Select frequency range:<br>This field sets the frequency range of HCLK input.<br>MDC is generated by dividing down the HCLK and the clock division is set by this bits. The settings of this bits are:<br>00: HCLK is equal to 25 MHz<br>01: HCLK is less than or equal to 33 MHz<br>10: HCLK is less than or equal to 50 MHz<br>11: HCLK is less than or equal to 66 MHz<br>MDC is set less than or equal to 2.5MHz. by the setting of this bits. | 0 |
| 1:0 | - | Reserved for future use. Write as 0 | - |

### 5.2.4.13 En_MCMD-MII Command Register (94H W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:2 | - | Reserved for future use. Write as 0 | - |
| 1 | SCANC | SCAN command:<br>When this bit is set to 1, MAC executes the SCAN command. | 0 |
| 0 | RSTAT | MII management read:<br>When this bit is set to 1, MAC executes the read access through MII management interface. | 0 |

### 5.2.4.14 En_MADR-MII Address Register (98H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:13 | - | Reserved for future use. Write as 0 | - |
| 12:8 | FIAD | MII PHY address:<br>This field sets PHY address to be selected during this operation. | 0 |
| 7:5 | - | Reserved for future use. Write as 0 | - |
| 4:0 | RGAD | MII register address:<br>This field sets register address to be accessed during this operation. | 0 |

### 5.2.4.15 En_MWTD-MII Write Data Register (9CH R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:16 | - | Reserved for future use. Write as 0 | - |
| 15:0 | CTLD | MII write data:<br>This field sets the MII management write data for write access through the MII management interface. | 0 |

### 5.2.4.16 En_MRDD-MII Read Data Register (A0H R)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:16 | - | Reserved for future use. | - |
| 15:0 | CTLD | MII write data:<br>This field sets the MII management write data for write access through the MII management interface. | 0 |

### 5.2.4.17 En_MIND-MII Indicate Register (A4H R)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:3 | - | Reserved for future use. | - |
| 2 | NVALID | SCAN command start status:<br>This bit is set to 1 when En_MCMD register: SCANC bit is set to 1. When the first read access using the SCAN command is completed, this bit is cleared. | 0 |
| 1 | SCANA | SCAN command active:<br>This bit is set to 1 while En_MCMD register: SCANC bit is set to 1. | 0 |
| 0 | BUSY | BUSY:<br>This bit indicates that the MAC is executing a management access to the external PHY device through the MII management interface. This bit is set to 1 during the management access. | 0 |

### 5.2.4.18 En_AFR-Address Filtering Register (C8H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:4 | - | Reserved for future use. Write as 0 | - |
| 3 | PRO | Promiscuous mode:<br>When this bit is set to 1, all receive packets are accepted. | 0 |
| 2 | PRM | Accept Multicast:<br>When this bit is set to 1, all multicast packets are accepted. | 0 |
| 1 | AMC | Accept Multicast ( qualified ):<br>When this bit is set to 1, multicast packets that match the conditions using hash table are accepted. | 0 |
| 0 | ABC | Accept Broadcast:<br>When this bit is set to 1, all broadcast packets are accepted. | 0 |

| PRO | PRM | AMC | ABC | Unicast | Multicast | Broadcast |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Yes (SA) | No | No |
| 0 | 0 | 0 | 1 | Yes (SA) | No | Pass |
| 0 | X | 1 | 1 | Yes (SA) | Pass | Pass |
| 0 | 1 | 0 | 1 | Yes (SA) | Yes (Hash) | Pass |
| 0 | X | 1 | 0 | Yes (SA) | Pass | No |
| 0 | 1 | 0 | 0 | Yes (SA) | Yes (Hash) | No |
| 1 | X | X | X | Pass | Pass | Pass |
| SRXEN = 0 | | | | Disable | Disable | Disable |

### 5.2.4.19 En_HT1-Hash Table Register 1 (CCH R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:0 | HT1 | Hash table 1:<br>This register is used with the HT2 register as the hash table for detection of qualified multicast packets. This register sets the upper 32bits of the hash table. HT(63:32) | 0 |

### 5.2.4.20 En_HT2-Hash Table Register 2 (D0H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:0 | HT2 | Hash table 2:<br>This register is used with the HT1 register as the hash table for detection of qualified multicast packets. This register sets the lower 32bits of the hash table. HT(31:0) | 0 |

#### 5.2.4.21 En_CAR1-Carry Register 1 (DCH R/W)

The bits of this register indicate that an overflow event has occurred in a statistics counter. Each bit corresponds to a counter, and the bit is set to 1 when the corresponding statistics counter overflow event occurs.

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:16 | - | Reserved for future use. Write as 0 | - |
| 15 | C1VT | En_RVBT counter carry bit | 0 |
| 14 | C1UT | En_TUCA counter carry bit | 0 |
| 13 | C1BT | En_TBCA counter carry bit | 0 |
| 12 | C1MT | En_TMCA counter carry bit | 0 |
| 11 | C1PT | En_TPCT counter carry bit | 0 |
| 10 | C1TB | En_TBYT counter carry bit | 0 |
| 9 | C1MX | En_RMAX counter carry bit | 0 |
| 8 | C11K | En_R1K counter carry bit | 0 |
| 7 | C1FE | En_R511 counter carry bit | 0 |
| 6 | C1TF | En_R255 counter carry bit | 0 |
| 5 | C1OT | En_R127 counter carry bit | 0 |
| 4 | C1SF | En_R64 counter carry bit | 0 |
| 3 | C1BR | En_RBCA counter carry bit | 0 |
| 2 | C1MR | En_RMCA counter carry bit | 0 |
| 1 | C1PR | En_RPKT counter carry bit | 0 |
| 0 | C1RB | En_RBYT counter carry bit | 0 |

#### 5.2.4.22 En_CAR2-CARRY Register 2 (E0H R/W)

The bits of this register indicate that an overflow event has occurred in a statistics counter. Each bit corresponds to a counter, and the bit is set to 1 when the corresponding statistics counter overflow event occurs.

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31 | C2XD | Status vector overrun bit | 0 |
| 30:23 | - | Reserved for future use. Write as 0 | - |
| 22 | C2IM | En_TIME counter carry bit | 0 |
| 21 | C2CS | En_TCSE counter carry bit | 0 |
| 20 | C2BC | En_TNCL counter carry bit | 0 |
| 19 | C2XC | En_TXCL counter carry bit | 0 |
| 18 | C2LC | En_TLCL counter carry bit | 0 |
| 17 | C2MC | En_TMCL counter carry bit | 0 |
| 16 | C2SC | En_TSCL counter carry bit | 0 |
| 15 | C2XD | En_TXDF counter carry bit | 0 |
| 14 | C2DF | En_TDFR counter carry bit | 0 |
| 13 | C2XF | En_TXPF counter carry bit | 0 |
| 12 | C2TE | En_TFCS counter carry bit | 0 |
| 11 | C2JB | En_RJBR counter carry bit | 0 |
| 10 | C2FG | En_RFRG counter carry bit | 0 |
| 9 | C2OV | En_ROVR counter carry bit | 0 |
| 8 | C2UN | En_RUND counter carry bit | 0 |
| 7 | C2FC | En_RFCR counter carry bit | 0 |
| 6 | C2CD | En_RCDE counter carry bit | 0 |
| 5 | C2FO | En_RFLR counter carry bit | 0 |
| 4 | C2AL | En_RALN counter carry bit | 0 |
| 3 | C2UO | En_RXUO counter carry bit | 0 |
| 2 | C2PF | En_RXPF counter carry bit | 0 |
| 1 | C2CF | En_RXCF counter carry bit | 0 |
| 0 | C2RE | En_RFCS counter carry bit | 0 |

#### 5.2.4.23 En_CAM1-CARRY Register 1 Mask Register (130H R/W)

This register masks Interrupt that is generated from the setting of the bits in the En_CAR1 register.

Each mask bit can be enabled independently.

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:16 | - | Reserved for future use. Write as 0 | - |
| 15 | M1VT | En_RVBT counter carry mask bit | 0 |
| 14 | M1UT | En_TUCA counter carry mask bit | 0 |
| 13 | M1BT | En_TBCA counter carry mask bit | 0 |
| 12 | M1MT | En_TMCA counter carry mask bit | 0 |
| 11 | M1PT | En_TPCT counter carry mask bit | 0 |
| 10 | M1TB | En_TBYT counter carry mask bit | 0 |
| 9 | M1MX | En_RMAX counter carry mask bit | 0 |
| 8 | M11K | En_R1K counter carry mask bit | 0 |
| 7 | M1FE | En_R511 counter carry mask bit | 0 |
| 6 | M1TF | En_R255 counter carry mask bit | 0 |
| 5 | M1OT | En_R127 counter carry mask bit | 0 |
| 4 | M1SF | En_R64 counter carry mask bit | 0 |
| 3 | M1BR | En_RBCA counter carry mask bit | 0 |
| 2 | M1MR | En_RMCA counter carry mask bit | 0 |
| 1 | M1PR | En_RPKT counter carry mask bit | 0 |
| 0 | M1RB | En_RBYT counter carry mask bit | 0 |

#### 5.2.4.24 En_CAM2-CARRY Register 2 Mask Register (134H R/W)

This register masks the Interrupt that is generated from the setting of the bits in the En_CAR2 register.

Each mask bit can be enabled independently.

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31 | M2XD | Status vector overrun mask bit | 0 |
| 30:23 | - | Reserved for future use. Write as 0 | - |
| 22 | M2IM | En_TIME counter carry mask bit | 0 |
| 21 | M2CS | En_TCSE counter carry mask bit | 0 |
| 20 | M2BC | En_TNCL counter carry mask bit | 0 |
| 19 | M2XC | En_TXCL counter carry mask bit | 0 |
| 18 | M2LC | En_TLCL counter carry mask bit | 0 |
| 17 | M2MC | En_TMCL counter carry mask bit | 0 |
| 16 | M2SC | En_TSCL counter carry mask bit | 0 |
| 15 | M2XD | En_TXDF counter carry mask bit | 0 |
| 14 | M2DF | En_TDFR counter carry mask bit | 0 |
| 13 | M2XF | En_TXPF counter carry mask bit | 0 |
| 12 | M2TE | En_TFCS counter carry mask bit | 0 |
| 11 | M2JB | En_RJBR counter carry mask bit | 0 |
| 10 | M2FG | En_RFRG counter carry mask bit | 0 |
| 9 | M2OV | En_ROVR counter carry mask bit | 0 |
| 8 | M2UN | En_RUND counter carry mask bit | 0 |
| 7 | M2FC | En_RFCR counter carry mask bit | 0 |
| 6 | M2CD | En_RCDE counter carry mask bit | 0 |
| 5 | M2FO | En_RFLR counter carry mask bit | 0 |
| 4 | M2AL | En_RALN counter carry mask bit | 0 |
| 3 | M2UO | En_RXUO counter carry mask bit | 0 |
| 2 | M2PF | En_RXPF counter carry mask bit | 0 |
| 1 | M2CF | En_RXCF counter carry mask bit | 0 |
| 0 | M2RE | En_RFCS counter carry mask bit | 0 |

### 5.2.5 Detail of DMA and FIFO management registers

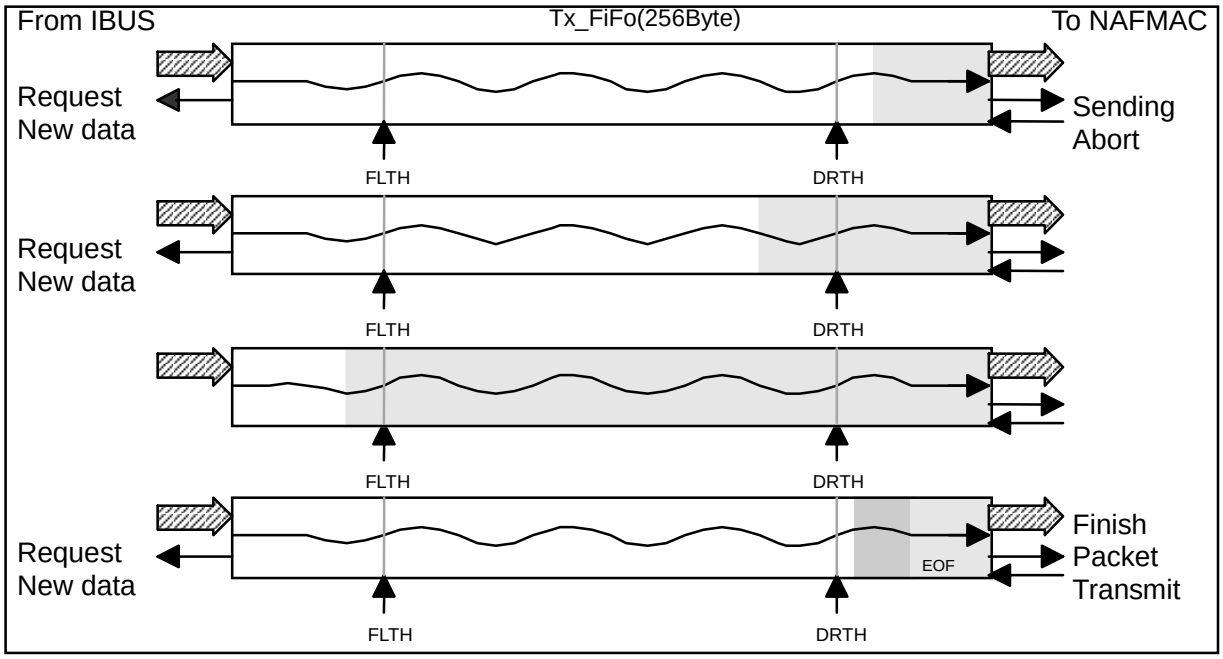
#### 5.2.5.1 En_TXCR-Transmit Configuration Register (200H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31 | TXE | Transmit Enable:<br>0: Disable<br>1: Enable | 0 |
| 30:19 | - | Reserved for future use. Write as 0 | - |
| 18:16 | DTBS<br>[2:0] | DMA Transmit Burst Size:<br>000: 1 Word (4 bytes)<br>001: 2 Word (8 bytes)<br>010: 4 Word (16 bytes)<br>011: 8 Word (32 bytes)<br>100: 16 Word (64 bytes)<br>101: 32 Word (128 bytes)<br>110: 64 Word (256 bytes)<br>111: Reserved for future use | 0 |
| 15:1 | - | Reserved for future use. Write as 0 | - |
| 0 | AFCE | Auto Flow Control Enable:<br>0: Disable<br>1: Enable | 0 |

### 5.2.5.2 En_TXFCR-Transmit FIFO Control Register (204H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:16 | TPTV | Transmit Pause Timer Value: | FFFFH |
| 15:10 | TX_DRTH | Transmit Drain Threshold Level:<br>This threshold is enable to the transmit data to the MAC form the FIFO. If the transfer data is not finish by DMAC and the buffer empty pointer exceed this pointer, this Ethernet sends Abort Packet. Please see the Figure 5-2. This is a word pointer. | 10H |
| 9:8 | - | Reserved for future use. Write as 0 | - |
| 7:2 | TX_FRTH | Transmit Fill Threshold Level:<br>This threshold is enable to transmit data to the FIFO from IBUS via internal DMAC of this block. Please see the Figure 5-2. This is a word pointer. | 03H |
| 1:0 | - | Reserved for future use. Write as 0 | - |

**Figure 5-2. Tx FIFO Control Mechanism**




### 5.2.5.3 En_TXDTR-Transmit Data Register (208H W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 32:0 | XMT_D<br>[31:0] | Transmit Data | 0 |

### 5.2.5.4 En_TXSR-Transmit Status Register (20CH R)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31 | CSE | Carrier lost was detected during the transmission | 0 |
| 30 | TBP | Back pressure occurred when the packet was received | 0 |
| 29 | TPP | A packet request during the PAUSE operation was transmitted Note1 | 0 |
| 28 | TPCF | A PAUSE control frame was transmitted | 0 |
| 27 | TCFR | A control frame was transmitted | 0 |
| 26 | TUDR | The TPUR pin was set high and aborted. Note 2 | 0 |
| 25 | TGNT | A packet greater than the value of the En_LMAX register in length was transmitted and aborted. Note 3 | 0 |
| 24 | LCOL | Collision occurred out of the collision window and the transmission was aborted. | 0 |
| 23 | ECOL | The number of collisions was greater than the value of the En_RETX register and the transmission was aborted. | 0 |
| 22 | TEDFR | The transmission was deferred beyond 24888 bits time and aborted. | 0 |
| 21 | TDFR | Transmission deferral occurred at least one time but less than an excessive deferral. | 0 |
| 20 | TBRO | A broadcast packet was transmitted. | 0 |
| 19 | TMUL | A multicast packet was transmitted. | 0 |
| 18 | TDONE | The transmission was completed. | 0 |
| 17 | TFLOR | The value of the length field in the transmitted packet was over 1518 bytes.Note4 | 0 |
| 16 | TFLER | The value of the length field in the transmitted packet didn't match the actual data byte length. Note 4, 5 | 0 |
| 15 | TCRCE | The attached CRC in the data byte stream didn't match the internal generated CRC when the En_MACC1 register: CRCEN bit was cleared to 0. | 0 |
| 14:11 | TCBC | The number of collisions for the previous transmission only. | 0 |
| 10:0 | TBYT | The number of the transmitted bytes not including collided bytes.Note6 | 0 |

**Notes 1.** This status is not reported if the packet requested during the PAUSE operation was a control frame requested by TPCF pin.

**2.** This status is reported only when no collision events occurred.

**3.** This status is reported only when the En_MACC1 register: HUGEN bit is cleared.

**4.** This status is not reported if En_MACC1 register: FLCHT bit is cleared.

**5.** When the value of the length field was over 1518 bytes, it is reported only as TFLOR status, not as TFLER status.

**6.** When the transmission was aborted, this value is not correct.

### 5.2.5.5 En_TXDPR-Transmit Descriptor Pointer(214H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:2 | XMTDP | Transmit Descriptor<br>Please see the Section 5.3.2.1. | 0H |
| 1:0 | - | Reserved for future use. Write as 0 | - |

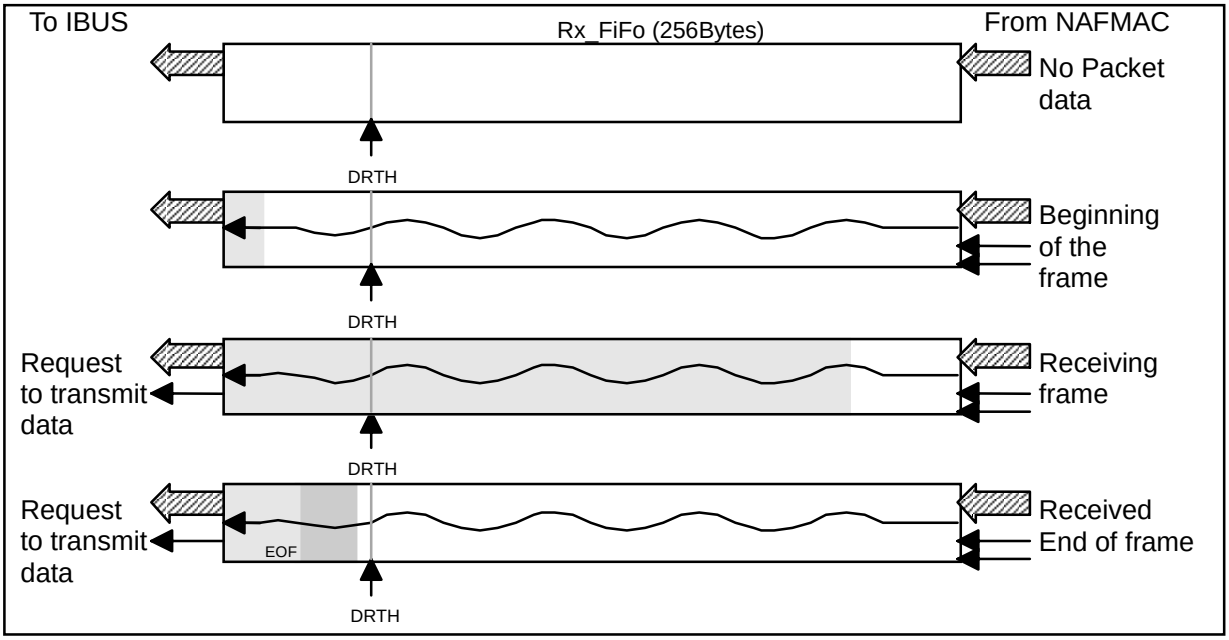
### 5.2.5.6 En_RXCR-Receive Configuration Register (218H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31 | RXE | Receive Enable:<br>0: Disable<br>1: Enable | 0 |
| 30:19 | - | Reserved for future use. Write as 0 | - |
| 18:16 | DRBS<br>[2:0] | DMA Transmit Burst Size:<br>000: 1 Word (4 bytes)<br>001: 2 Word (8 bytes)<br>010: 4 Word (16 bytes)<br>011: 8 Word (32 bytes)<br>100: 16 Word (64 bytes)<br>101: 32 Word (128 bytes)<br>110: 64 Word (256 bytes)<br>111: Reserved for future use | 0 |
| 15:0 | - | Reserved for future use. Write as 0 | - |

### 5.2.5.7 En_RXFCR-Receive FIFO Control Register (21CH R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:26 | UWM [7:2] | Upper Water Mark:<br>This pointer is used with Auto Flow Control Enable in En_TXCR. When the receiving data fill level exceeds this pointer, the transmit module generates a flow control frame automatically. | 30H |
| 25:24 | - | Reserved for future use. Write as 0 | - |
| 23:18 | LWN [7:2] | Lower Water Mark: | 10H |
| 17:8 | - | Reserved for future use. Write as 0 | - |
| 7:2 | RX_DRTH | Receive Drain Threshold Level<br>This threshold is enable to the transmit data to the IBUS via internal DMAC form the FIFO. Please see the Figure 5-3. This pointer is a word pointer. | 10H |
| 1:0 | - | Reserved for future use. Write as 0 | - |

**Figure 5-3. Rx FIFO Control Mechanism**




### 5.2.5.8 En_RXDTR-Receive Data Register (220H R)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 32:0 | RCV_D [31:0] | Receive Data | 0 |

### 5.2.5.9 En_RXSR-Receive Status Register (224H R)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31 | RLENE | A packet which was less than 64 octets or larger than 1518 octets was received.(If a VLAN packet, the packet was less than 68 bytes or larger than 1522 bytes) | 0 |
| 30 | VLAN | A packet in which the Ethernet-encoded TPID field matched the value of the En_VLTP register was received.[Note1] | 0 |
| 29 | USOP | A control frame containing an unknown OP code was received. [Note 2] | 0 |
| 28 | PRCF | A control frame containing the PAUSE OP code was received. [Note 2] | 0 |
| 27 | RCFR | A frame containing a valid type field designating it as a control frame was received. [Note 2] | 0 |
| 26 | DBNB | An alignment error occurred. | 0 |
| 25 | RBRO | A broadcast packet was received. | 0 |
| 24 | RMUL | A multicast packet was received. | 0 |
| 23 | RXOK | A packet which had a valid CRC and no RXER was received. [Note 1] | 0 |
| 22 | RLOR | The value of the length field in the received packet was over 1518 bytes. [Note 3] | 0 |
| 21 | RLER | The value of the length field in the received packet didn't match the actual data byte length. [Note 3, Note 4] | 0 |
| 20 | RCRCE | A CRC error occurred. | 0 |
| 19 | RCV | RXER was detected. | 0 |
| 18 | CEPS | A False Carrier was detected. | 0 |
| 17 | REPS | A packet which had a preamble and SFD only or one data nibble only was received. | 0 |
| 16 | PAIG | The previous packet with the following status was received and<br>• The carrier length was 6072 nibble (3036 octets)<br>• Short IPG<br>• When En_MACC1 register: PUREP bit was set to 1, an invalid preamble or invalid SFD was detected. | 0 |
| 15:0 | RBYT | The received byte count | 0 |

**Note 1.** This status is not reported when a CRC error or RXER occurs.

**2.** This status is not reported when a CRC error occurs.

**3.** This status is not reported if En_MACC1 register: FLCHT bit is cleared.

**4.** When the value of the length field is over 1518 bytes, it is reported only as RLOR status, not as RLER status.

### 5.2.5.10 En_RXDPR-Receive Descriptor Pointer (22CH R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:2 | RCVDP | Receive Descriptor Pointer:<br>Please see the Section 5.3.2.1. | 0H |
| 1:0 | - | Reserved for future use. Write as 0 | - |

#### 5.2.5.11 En_RXPDR-Receive Pool Descriptor Pointer (230H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31 | - | Reserved for future use. Write as 0 | - |
| 30:28 | AL[2:0] | Alert Level | 0H |
| 27:16 | - | Reserved for future use. Write as 0 | - |
| 15:0 | RNOD [15:0] | Remaining Number of Descriptor | 0H |

### 5.2.6 Detail of interrupt and configuration registers

#### 5.2.6.1 En_CCR-Configuration Register (240H R/W)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:1 | - | Reserved for future use. Write as 0 | - |
| 0 | SRT | Software Reset | 0H |

#### 5.2.6.2 En_ISR-Interrupt Serves Register (244H R with clear)

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:16 | - | Reserved for future use. Write as 0 | - |
| 15 | XMTDN | Transmit Done | 0 |
| 14 | TBDR | Transmit Buffer Descriptor Request at Null | 0 |
| 13 | TFLE | Transmit Frame Length Exceed | 0 |
| 12 | UR | Underrun | 0 |
| 11 | TABR | Transmit Aborted | 0 |
| 9:8 | - | Reserved for future use. Write as 0 | - |
| 10 | TCFRI | Control Frame Transmit | 0 |
| 7 | RCVDN | Receive Done | 0 |
| 6 | RBDRS | Receive Buffer Descriptor Request at alert level | 0 |
| 5 | RBDRU | Receive Buffer Descriptor Request at zero | 0 |
| 4 | OF | Overflow | 0 |
| 3 | LFAL | Link Failed | 0 |
| 2:1 | - | Reserved for future use. Write as 0 | - |
| 0 | CARRY | Carry Flag:<br>CARRY indicates an overflow of the statistics counters | 0 |

#### 5.2.6.3 En_MSR-Mask Serves Register (23CH R/W)

Each interrupt source is maskable. En_MSR register shows which interrupts are enable.

Default value is all “0” which means all interrupt sources are disable.

| Bit | Field | Functions | Default |
|---|---|---|---|
| 31:16 | - | Reserved for future use. Write as 0 | - |
| 15 | XMTDN | Transmit Done | 0 |
| 14 | TBDR | Transmit Buffer Descriptor Request at Null | 0 |
| 13 | TFLE | Transmit Frame Length Exceed | 0 |
| 12 | UR | Underrun | 0 |
| 11 | TABR | Transmit Aborted | 0 |
| 9:8 | - | Reserved for future use. Write as 0 | - |
| 10 | TCFRI | Control Frame Transmit | 0 |
| 7 | RCVDN | Receive Done | 0 |
| 6 | RBDRS | Receive Buffer Descriptor Request at alert level | 0 |
| 5 | RBDRU | Receive Buffer Descriptor Request at zero | 0 |
| 4 | OF | Overflow | 0 |
| 3 | LFAL | Link Failed | 0 |
| 2:1 | - | Reserved for future use. Write as 0 | - |
| 0 | CARRY | Carry Flag:<br>CARRY indicates an overflow of the statistics counters | 0 |

## 5.3 Operation

### 5.3.1 Initialization

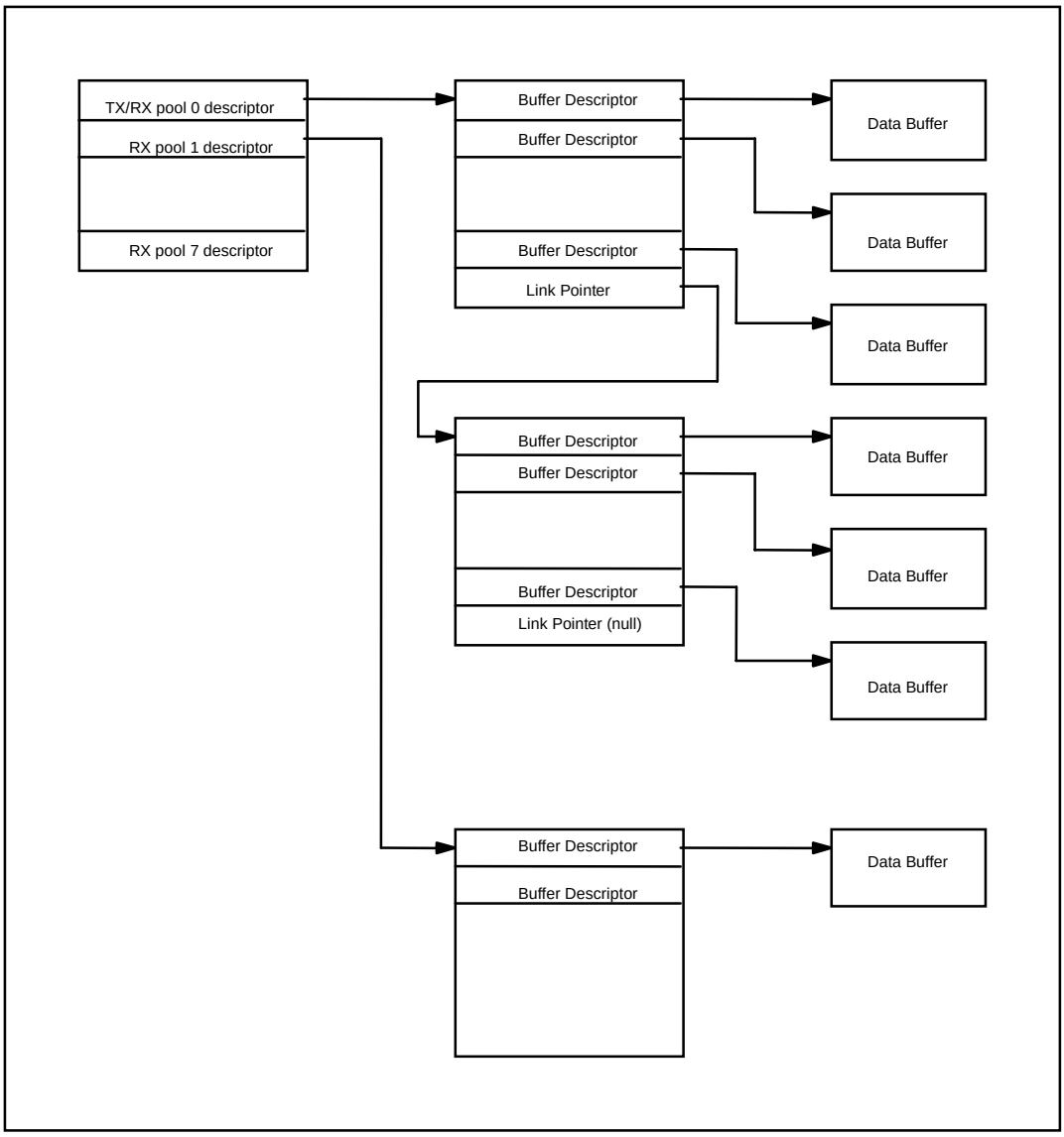
After a power on or a software reset , VR4120A RISC Core has to set the following registers:

i) Interrupt Mask Registers
ii) Configuration Registers
iii) MII Management Registers
iv) Pool / Buffer Descriptor Registers

### 5.3.2 Buffer structure for Ethernet block

The data buffer structure for Ethernet Controller is shown in Figure 5-4.

**Figure 5-4. Buffer Structure for Ethernet Block**



### 5.3.3 Buffer descriptor format

The Transmit Descriptor format is shown in Figure 5-5 and the description is shown in Table 5-6.

**Figure 5-5. Transmit Descriptor Format**

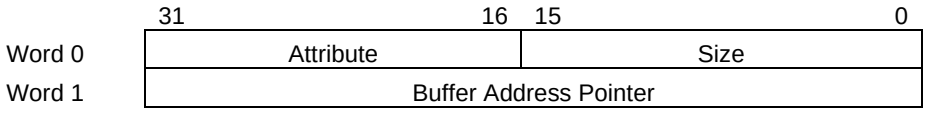



**Table 5-6. Attribute for Transmit Descriptor**

| Attribute | Bit Name | Status |
|---|---|---|
| 31 | L | Last Descriptor |
| 30 | D/L | Data Buffer / Link Pointer |
| 29 | OWN | Owner 1:Ethernet Controller 0: VR4120A |
| 28 | DBRE | Data Buffer Read Error |
| 27 | TUDR | Transmit Underrun Error |
| 26 | CSE | Carrier Sense Lost Error |
| 25 | LCOL | Late Collision |
| 24 | ECOL | Excessive Collision |
| 23 | EDFR | Excessive Deferral |
| 22:19 | - | Reserved for future use. Write as 0. |
| 18 | TGNT | Transmit Giant Frame |
| 17 | HBF | Heart Beat Fail for ENDEC mode |
| 16 | TOK | Transmit OK |
| 15:0 | SIZE | Transmit Byte Count |

The Receive Descriptor format is shown in Figure 5-6 and the description is shown in Table 5-7.

**Figure 5-6. Receive Descriptor Format**

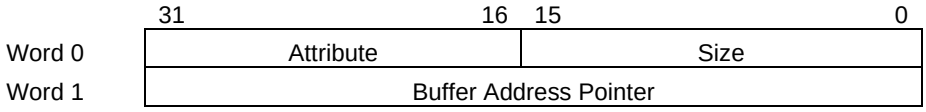


**Table 5-7. Attribute for Receive Descriptor**

| Attribute | Bit Name | Status |
|---|---|---|
| 31 | L | Last Frame |
| 30 | D/L | Data Buffer / Link Pointer |
| 29 | OWN | Owner bit 1:Ethernet Controller 0: VR4120A |
| 28 | DBWE | Data Buffer Write Error |
| 27:25 | FTYP | Frame Type[2:0]:<br>000 Broadcast Frame<br>001 Multicast Frame<br>010 Unicast Frame<br>011 VLAN Frame<br>100 PAUSE control frame<br>101 Control Frame (except pause)<br>11x Reserved for future use |
| 24 | OVRN | Overrun Error |
| 23 | RUNT | Runt packet |
| 22 | FRGE | Fragment Error |
| 21 | RCV | Detects RXER |
| 20 | FC | False Carrier |
| 19 | CRCE | CRC Error |
| 18 | FAE | Frame Alignment Error |
| 17 | RFLE | Receive Frame Length Error |
| 16 | RXOK | Receive OK |
| 15:0 | SIZE | Receive Byte Count |

**Remark** RUNT packet: less than 64 bytes packet with a good FCS
FRAGMENT packet: less than 64 bytes packet with either a bad FCS or a bad FCS with an alignment error

### 5.3.4 Frame transmission

The transmitter is designed to work with almost no intervention from the host processor. Once the driver enables the transmitter by setting the Transmit Descriptor Register (XMTDP) and the Transmit Enable (TXE), Ethernet Controller fetches the first Transmit Data Buffer from Buffer Descriptor.

When the drain threshold level of the transmit FIFO was over, the MAC transmit logic will assert TX_EN and start transmitting the preamble sequence, the start frame delimiter, and then the frame information. However, the controller defers the transmission if the line is busy (carrier sense is active). Before transmitting, the controller has to wait for carrier sense to become inactive. Once carrier sense is inactive, the controller determines if carrier sense stays inactive for IPGR1 bit time in En_IPGR register. If so, then the transmission begins after waiting an additional IPGR2 – IPGR1 bit times (i.e., IPG is generally 96 bit times).

If a collision occurs during the transmit frame, Ethernet Controller follows the specified back-off procedures and attempts to re-transmit the frame until the retry limit threshold is reached (RETRY in En_CLRT register). Ethernet Controller holds the first 64 bytes of the transmit frame in the transmit FIFO, so that Ethernet Controller does not have to be retrieved from system memory in case of a collision. This improves bus utilization and latency.

When Ethernet Controller reads the Transmit Data Buffer, and it shows the end of data buffer “L bit is active”, Ethernet Controller adds the FCS after the end of data if CRCEN in En_MACC1 register is enable.

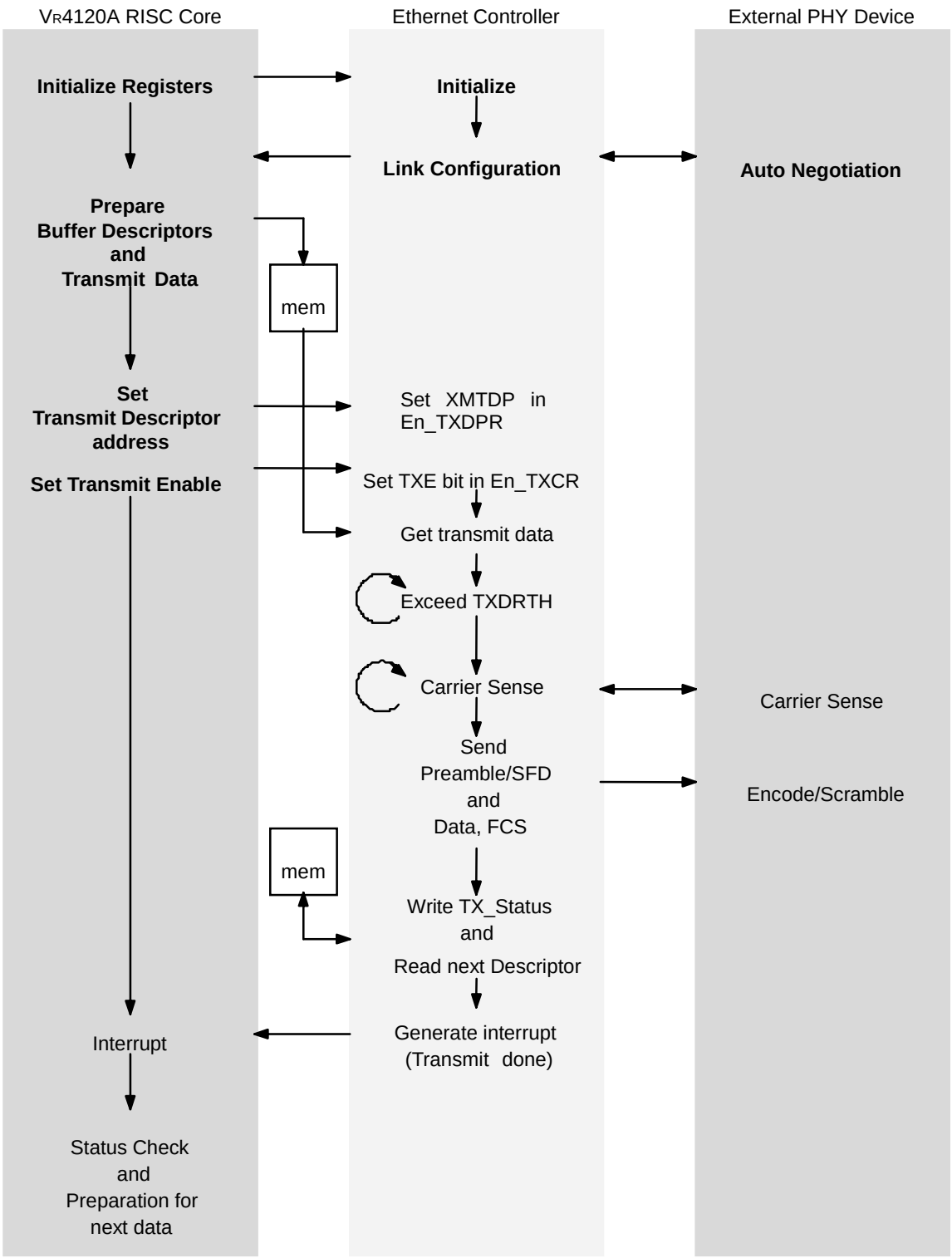
Short frames are automatically padded by the transmit logic if PADEN bit in En_MACC1 register is set. If the transmit frame length exceeds 1518 bytes, Ethernet Controller will assert an interrupt. However, the entire frame will be transmitted (no truncation).

If the current descriptor does not contain the end of frame, Ethernet Controller reads next buffer descriptor, and then reads the continuous data from the data buffer. After Ethernet Controller sent out the whole packet,. Ethernet Controller writes the transmission status into the last descriptor (L=1), and generates an interrupt to indicates the end of transmission.

When Ethernet Controller received the pause control frame and if it is active, Ethernet Controller transmitter stops immediately if no transmission is in progress or continues transmission until the current frame either finishes or terminates with a collision. When the pause timer was expired or Ethernet Controller received a zero value of pause control frame, Ethernet Controller resumes transmission with the next frame.

Transmit procedure is as follows: (Figure 5-7)

**Figure 5-7. Transmit Procedure**



**Operation flow for transmit packet**

i) Prepares transmit data in data buffer
ii) Initializes registers (En_TXDPR, En_TXCR)
iii) Reads buffer descriptor for transmission from SDRAM
iv) Reads transmit data from data buffer by using master DMA burst operation
v) Waits for exceeding of transmit drain threshold (TXDRTH)
Senses carrier
Transmits data (Preamble. SFD, data)
vi) Reads continuous data?
If the current buffer descriptor does not show a last packet (L=0), it reads continuous data.
Increments current Transmit Descriptor Pointer
vii) Reads next buffer descriptor
viii) Reads continuous data from data buffer again
ix) Stores the transmit status in the buffer descriptor
x) Generates an interrupt
xi) Reads next buffer descriptor and data, if available

### 5.3.5 Frame reception

The receiver is designed to work with almost no intervention from the host processor and can perform address recognition, CRC checking and maximum frame length checking.

When the driver enables the receiver by setting Receive Descriptor Pointer Register (En_RXDPR) and Receive Enable (RXE), it will immediately start processing receive frames. The receiver will first check for a valid preamble (PA) / start frame delimiter (SFD) header at the beginning packet. If the PA/SFD is valid, it will be stripped and the frame will be processed by the receiver. If a valid PA/SFD is not found the frame will be ignored.

Once a collision window (64 bytes) of data has been received and if address recognition has not rejected the frame, Ethernet Controller starts transferring the incoming frame to the receive data buffer. If the frame is a runt (due to collision) or is rejected by address recognition, no receive buffers are filled. Thus, no collision frames are presented to the user except late collisions, which indicate serious LAN problems.

It has no matter since after the reception it writes the receive status into the descriptor even if the received data were gone out to SDRAM.

If the incoming frame exceeds the length of the data buffer, Ethernet Controller fetches the next Receive Descriptor Buffer in the table and, if it is empty, continues transferring the rest of the frame to this data buffer.
If the remaining number of descriptors is under four times of the alert level, Ethernet Controller generates an interrupt to request new additional descriptors.

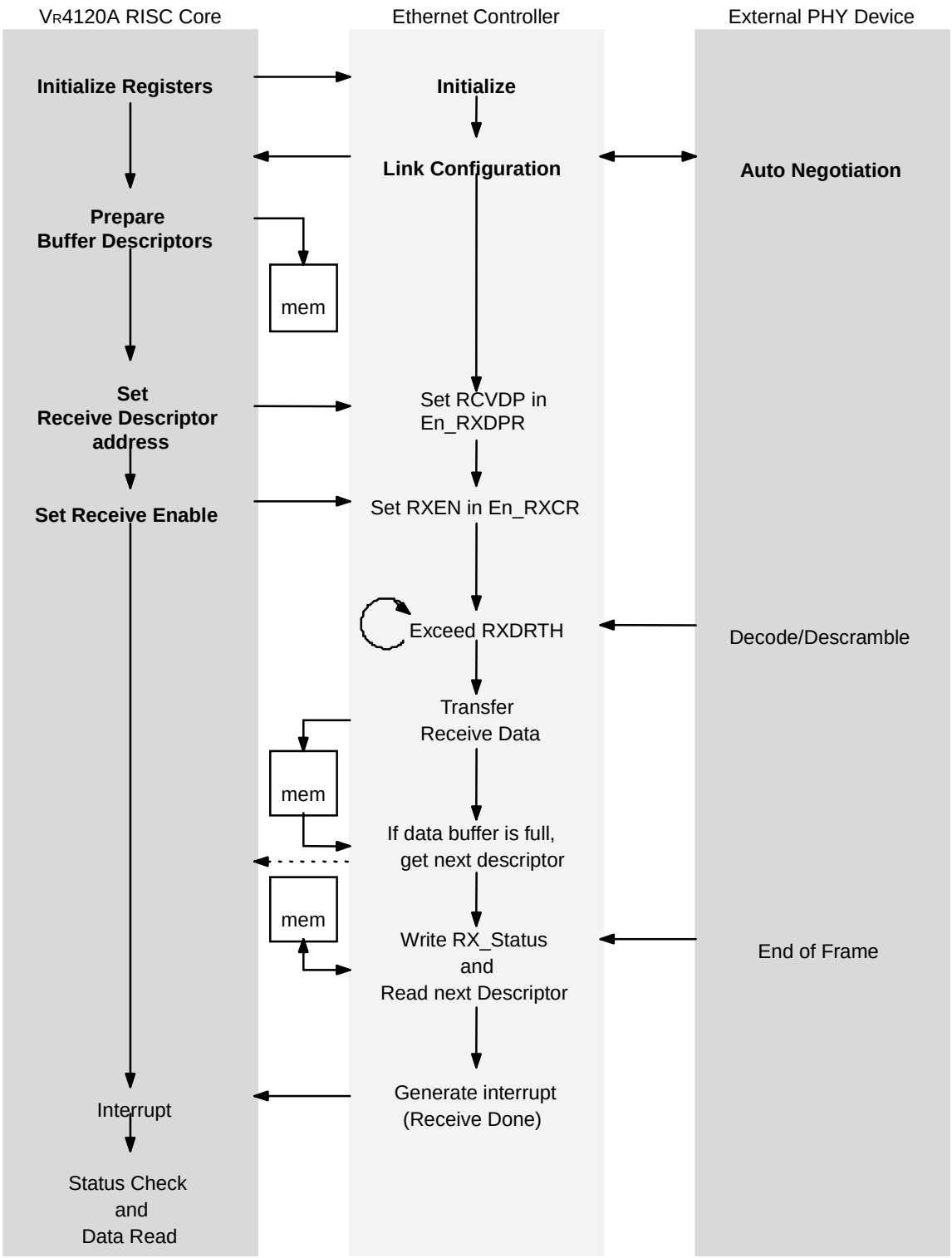
During reception, Ethernet Controller checks for a frame that is either too short or too long. When the frame ends (carrier sense is negated), the receive CRC field is checked out and written to the data buffer. The data length written to the last data Buffer in the Ethernet frame is the length of the entire frame. Frames that are less then 64 bytes in length are not DMA'd (transferred) and, are rejected in hardware with no impact on system bus utilization if the data is in the Rx FIFO.

**Caution Recommend a high (over 16 words) drain threshold.**

When the receive frame is complete, Ethernet Controller sets the L-bit in the Receive Descriptor, writes the frame status bits into the Receive Descriptor, and sets the OWN-bit. Ethernet Controller generates a maskable interrupt, indicating that a frame has been received and is in memory. Ethernet Controller then waits for a new frame.

Receive procedure is as follows:(Figure 5-8)

**Figure 5-8. Receive Procedure**



**Operation flow for receive packet**

i) Prepares the receive buffer descriptors
ii) Initializes registers (En_RXDPR, En_RXCR, En_MSR)
iii) Reads the receive buffer descriptor
iv) Waits for exceeding of receive drain threshold (RXDRTH)
v) Writes receive data to data buffer by using master DMA burst operation
vi) Increments the Receive Descriptor Pointer if the current data buffer is full
vii) Check out the RNOD
If the remaining number of descriptors is less than four times of the alert level, generates an interrupt to request an adding descriptor.
viii) Reads the next buffer descriptor
ix) Stores the received data
x) Stores the receive status in Receive Descriptor
xi) Generates an interrupt for the end of reception
xii) Reads next Receive Descriptor if available

**How to add the receive buffer descriptors**

i) Prepares the receive buffer descriptors
ii) Sets the number of buffer descriptors in En_RXPDR as well as the alert level
iii) Generates an interrupt by this Ethernet Controller
iv) Adds the receive buffer descriptors in the memory
v) Sets the number of buffer descriptors in En_RXPDR as well as the alert level

# CHAPTER 6 USB CONTROLLER

## 6.1 Overview

This block is built into the μPD98501 Access Network Controller and, in cooperation with the USB, performs the processing required for transferring data. The following lists the features of USB Controller.

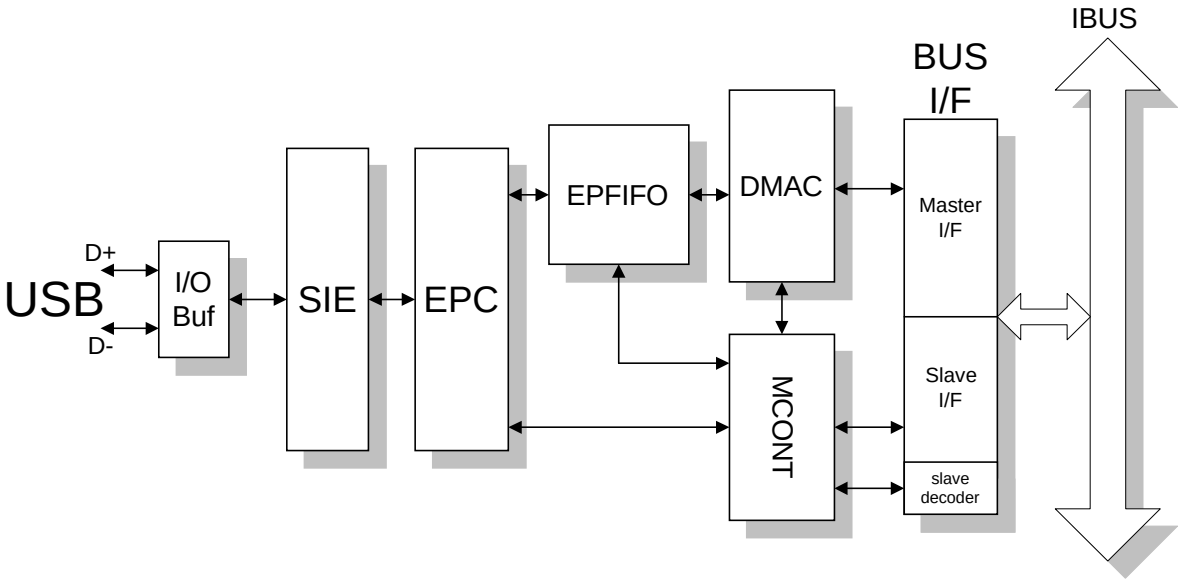
### 6.1.1 Features

- Conforms to Universal Serial Bus Specification Rev 1.1
- Supports operation conforming to the USB Communication Device Class Specification
- Supports data transfer at full speed (12 Mbps)
- In addition to the control Endpoint, a further six Endpoints are built in
  (Interrupt in/out, Isochronous in/out and Bulk in/out)
- For Control transfer, a built-in 64-byte FIFO is provided for send
- For Isochronous transfer, a built-in 128-byte FIFO is provided for send
- For Bulk transfer, a built-in 128-byte FIFO is provided for send
- For Interrupt transfer, a built-in 64-byte FIFO is provided for send
- For Control/Isochronous/Bulk/Interrupt transfer, a built-in 128-byte shared FIFO is provided for receive
- A DMA function for transferring send/receive data is built in
- A Control/Status registers are built in
- Compatible with the Suspend and Resume signaling issued from the Host PC
  (Processing by the VR4120A RISC Processor is required)
- Remote Wake-up is supported (Processing by the VR4120A RISC Processor is required)
- Directly connect to Internal BUS (IBUS) Master and Slave Interface block
- All the counters required to indicate the USB status are built in

### 6.1.2 Internal block diagram

USB Controller internal block diagram is as shown below.

**Figure 6-1. USB Controller Internal Configuration**




USB Controller's configuration features the following blocks.

| | | |
|---|---|---|
| SIE (Serial Interface Engine) | : | Performs Serial/Parallel conversion, NRZI encode/decode, CRC calculation, etc. |
| EPC (EndPoints Controller) | : | Performs data send/receive for each Endpoint. |
| EPFIFO (RX FIFO) | : | FIFO for Send and Receive |
| MCONT (Main Controller) | : | Block for controlling sending and receiving. |
| DMAC (DMA Controller) | : | Block for controlling DMA transfer. |
| Master_if (Master Interface) | : | Master section of the Internal BUS interface. |
| Slave_if (Slave Interface) | : | Slave section of the Internal BUS interface. |
| I/O Buf (I/O Buffer) | : | I/O buffer that satisfies the electrical specifications of the USB. |
| Internal BUS | : | The internal bus of the μPD98501. |

## 6.2 Register Set

This section explains the mapping of those registers that can be accessed from IBUS.

### 6.2.1 Register map

| Offset Address | Register | Description | R/W | Default Value |
|---|---|---|---|---|
| 00H | U_GMR | USB General Mode Register | R/W | 00001800H |
| 04H | U_VER | USB Frame number/Version Register | R | 01000000H |
| 08H | N/A | Reserved for future use | R/W | 00000000H |
| 0CH | N/A | Reserved for future use | R | 00000000H |
| 10H | U_GSR1 | USB General Status Register 1 | R | 00000000H |
| 14H | U_IMR1 | USB Interrupt Mask Register 1 | R/W | 00000000H |
| 18H | U_GSR2 | USB General Status Resister 2 | R | 00000000H |
| 1CH | U_IMR2 | USB Interrupt Mask Register 2 | R/W | 00000000H |
| 20H | U_EP0CR | USB EP0 Control Register | R/W | 00000000H |
| 24H | U_EP1CR | USB EP1 Control Register | R/W | 00000000H |
| 28H | U_EP2CR | USB EP2 Control Register | R/W | 00000000H |
| 2CH | U_EP3CR | USB EP3 Control Register | R/W | 00000000H |
| 30H | U_EP4CR | USB EP4 Control Register | R/W | 00000000H |
| 34H | U_EP5CR | USB EP5 Control Register | R/W | 00000000H |
| 38H | U_EP6CR | USB EP6 Control Register | R/W | 00000000H |
| 3CH | N/A | Reserved for future use | - | - |
| 40H | U_CMR | USB Command Register | R/W | 00000000H |
| 44H | U_CA | USB Command Address Register | R/W | 00000000H |
| 48H | U_TEPSR | USB Tx EndPoint Status Register | R/W | 00000000H |
| 4CH | N/A | Reserved for future use | - | - |
| 50H | U_RP0IR | USB Rx Pool0 Information Register | R/W | 00000000H |
| 54H | U_RP0AR | USB Rx Pool0 Address Register | R | 00000000H |
| 58H | U_RP1IR | USB Rx Pool1 Information Register | R/W | 00000000H |
| 5CH | U_RP1AR | USB Rx Pool1 Address Register | R | 00000000H |
| 60H | U_RP2IR | USB Rx Pool2 Information Register | R/W | 00000000H |
| 64H | U_RP2AR | USB Rx Pool2 Address Register | R | 00000000H |
| 68H | N/A | Reserved for future use | - | - |
| 6CH | N/A | Reserved for future use | - | - |
| 70H | U_TMSA | USB Tx MailBox Start Address Register | R/W | 00000000H |
| 74H | U_TMBA | USB Tx MailBox Bottom Address Register | R/W | 00000000H |
| 78H | U_TMRA | USB Tx MailBox Read Address Register | R/W | 00000000H |
| 7CH | U_TMWA | USB Tx MailBox Write Address Register | R | 00000000H |
| 80H | U_RMSA | USB Rx MailBox Start Address Register | R/W | 00000000H |
| 84H | U_RMBA | USB Rx MailBox Bottom Address Register | R/W | 00000000H |
| 88H | U_RMRA | USB Rx MailBox Read Address Register | R/W | 00000000H |
| 8CH | U_RMWA | USB Rx MailBox Write Address Register | R | 00000000H |
| 90-FFH | N/A | Reserved for future use | - | - |
| 100H | U_TDN | USB EP0 Tx Data Phase NAK Counter | R | 00000000H |
| 104H | U_TDS | USB EP0 Tx Data Phase STALL Counter | R | 00000000H |
| 108H | N/A | Reserved for future use | - | - |
| 10CH | N/A | Reserved for future use | - | - |
| 110H | U_THT | USB EP0 Tx Handshake Phase Timeout Counter | R | 00000000H |
| 114H | N/A | Reserved for future use | - | - |

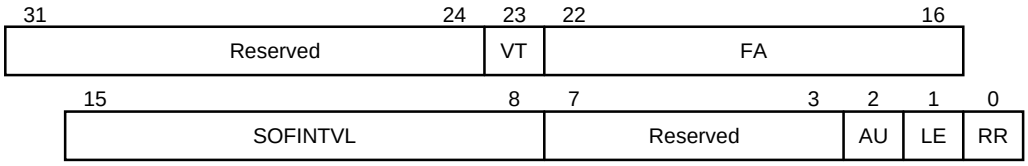
| Offset Address | Register | Description | R/W | Default Value |
|---|---|---|---|---|
| 118H | N/A | Reserved for future use | - | - |
| 11CH | N/A | Reserved for future use | - | - |
| 120H | U_RDT | USB EP0 Rx Data Phase Timeout Counter | R | 00000000H |
| 124H | U_RDCER | USB EP0 Rx Data Phase CRC Error Counter | R | 00000000H |
| 128H | U_RDBER | USB EP0 Rx Data Phase Bitstuff Error Counter | R | 00000000H |
| 12CH | U_RDTER | USB EP0 Rx Data Phase Data Toggle Error Counter | R | 00000000H |
| 130H | U_RHT | USB EP0 Rx Handshake Phase Timeout Counter | R | 00000000H |
| 134H | U_RHN | USB EP0 Rx Handshake Phase NAK Counter | R | 00000000H |
| 138H | U_RHS | USB EP0 Rx Handshake Phase STALL Counter | R | 00000000H |
| 13CH | N/A | Reserved for future use | R | 00000000H |
| 140-15FH | N/A | Reserved for future use | - | - |
| 160H | U_DT2 | USB EP2 Data Phase Timeout Counter | R | 00000000H |
| 164H | U_DCER2 | USB EP2 Data Phase CRC Error Counter | R | 00000000H |
| 168H | U_DBER2 | USB EP2 Data Phase Bitstuff Error Counter | R | 00000000H |
| 16CH | N/A | Reserved for future use | - | - |
| 170-17FH | N/A | Reserved for future use | - | - |
| 180H | U_DN3 | USB EP3 Data Phase NAK Counter | R | 00000000H |
| 184H | U_DS3 | USB EP3 Data Phase STALL Counter | R | 00000000H |
| 188H | N/A | Reserved for future use | - | - |
| 18CH | N/A | Reserved for future use | - | - |
| 190H | U_HT3 | USB EP3 Handshake Phase Timeout Counter | R | 00000000H |
| 194H | N/A | Reserved for future use | - | - |
| 198H | N/A | Reserved for future use | - | - |
| 19CH | N/A | Reserved for future use | - | - |
| 1A0H | U_DT4 | USB EP4 Data Phase Timeout Counter | R | 00000000H |
| 1A4H | U_DCER4 | USB EP4 Data Phase CRC Error Counter | R | 00000000H |
| 1A8H | U_DBER4 | USB EP4 Data Phase Bitstuff Error Counter | R | 00000000H |
| 1ACH | U_DTER4 | USB EP4 Data Phase Data Toggle Error Counter | R | 00000000H |
| 1B0H | U_HT4 | USB EP4 Handshake Phase Timeout Counter | R | 00000000H |
| 1B4H | U_HN4 | USB EP4 Handshake Phase NAK Counter | R | 00000000H |
| 1B8H | U_HS4 | USB EP4 Handshake Phase STALL Counter | R | 00000000H |
| 1BCH | N/A | Reserved for future use | - | - |
| 1C0H | U_DN5 | USB EP5 Data Phase NAK Counter | R | 00000000H |
| 1C4H | U_DS5 | USB EP5 Data Phase STALL Counter | R | 00000000H |
| 1C8H | N/A | Reserved for future use | - | - |
| 1CCH | N/A | Reserved for future use | - | - |
| 1D0H | U_DT5 | USB EP5 Handshake Phase Timeout Counter | R | 00000000H |
| 1D4H | N/A | Reserved for future use | - | - |
| 1D8H | N/A | Reserved for future use | - | - |
| 1DCH | N/A | Reserved for future use | - | - |
| 1E0H | U_DT6 | USB EP6 Data Phase Timeout Counter | R | 00000000H |
| 1E4H | U_DCER6 | USB EP6 Data Phase CRC Error Counter | R | 00000000H |
| 1E8H | U_DBER6 | USB EP6 Data Phase Bitstuff Error Counter | R | 00000000H |
| 1ECH | U_DTER6 | USB EP6 Data Phase Data Toggle Error Counter | R | 00000000H |
| 1F0H | U_HT6 | USB EP6 Handshake Phase Timeout Counter | R | 00000000H |
| 1F4H | U_HN6 | USB EP6 Handshake Phase NAK Counter | R | 00000000H |
| 1F8H | U_HS6 | USB EP6 Handshake Phase STALL Counter | R | 00000000H |
| 1FCH | N/A | Reserved for future use | - | - |

**Remark** USB base address: 1000_1000H

### 6.2.2 Explanation of registers

This section explains each bit field of the address-mapped registers.

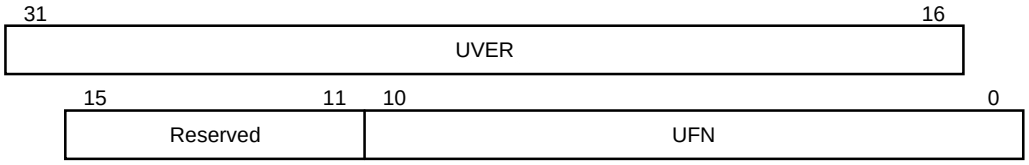
#### 6.2.2.1 USB General Mode Register (U_GMR): 00H




This register is used for setting the operation of USB Controller. The low-order eight bits can be written to once only when the device is being initialized. If the values of these bits are changed while sending or receiving is being performed, the operation of USB Controller may become unpredictable.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-24 | Reserved | Reserved for future use | R |
| 23 | VT<br>(Function Address Valid Timing) | If this bit is 1, FA become valid immediately.<br>If this bit is 0, FA will become valid after USB Controller receives subsequent ACK packet on EndPoint0. | R/W |
| 22-16 | FA<br>(Function Address) | Register that stores the USB Function Address. This is allocated by the Host PC as part of the USB configuration process. The VR4120A RISC Processor sets the allocated address in this register. | R/W |
| 15-8 | SOFINTVL<br>(SOF Interval) | This value is used to define the allowable skew for SOF packet. The default value should be 18H. | R/W |
| 7-3 | Reserved | Reserved for future use | R |
| 2 | AU<br>(Auto Update) | Frame Number auto update enable.<br>If this bit is set, Frame Number Register will be updated though SOF packet is corrupted. | R/W |
| 1 | LE<br>(Loopback Enable) | Bit for enabling internal loopback mode.<br>When this bit is set to 1, USB Controller operates in loopback mode.<br>Setting loopback mode enables the testing of the internal DMA controller that is built into USB Controller.<br>When loopback mode is set, USB packets are not output. Also, USB packets are not received.<br>For a detailed explanation of loopback mode, see Section 6.8. | R/W |
| 0 | RR<br>(Remote Resume) | When Remote Resume is to be performed, the VR4120A RISC Processor sets this bit.<br>Once this bit has been set, USB Controller issues Resume Signaling to the USB for a period of 5 ms. Upon the completion of Resume Signaling, this bit is automatically reset to 0. | R/W |

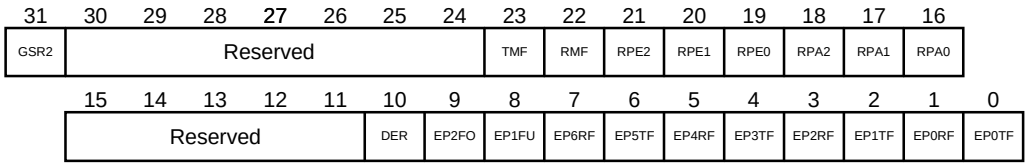
### 6.2.2.2 USB Frame Number/Version Register (U_VER): 04H




Register that stores the current Frame Number of the USB and version of the USB Controller block.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-16 | UVER<br>(USB Version) | The Revision Number of the USB Controller block is stored into this register. | R |
| 15-11 | Reserved | Reserved for future use | R |
| 10-0 | UFNR<br>(USB Frame Number) | Register that stores the Frame Number of the USB.<br>The current USB Frame Number can be viewed from the VR4120A RISC Processor. | R |

### 6.2.2.3 USB General Status Register 1 (U_GSR1): 10H




This register indicates the current status of USB Controller.

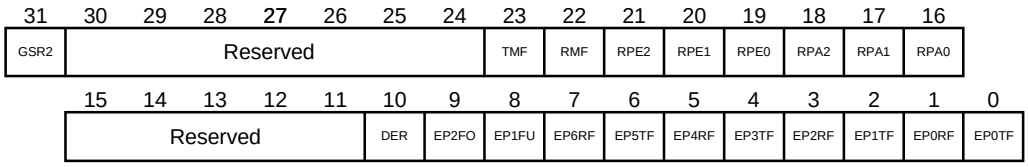
(1/2)

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31 | GSR2 | If some bit of General Status Register2 is set to 1 and the corresponding bit of Interrupt Mask Register2 is set to 1, this GSR2 bit will be active. | R |
| 30-24 | Reserved | Reserved for future use | R |
| 23 | TMF<br>(Tx MailBox Full) | Bit that indicates that the send mailbox area is full. This bit is set to 1 when the USB Tx MailBox Read Address and the USB Tx MailBox Write Address match. This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 22 | RMF<br>(Rx MailBox Full) | Bit that indicates that the receive mailbox area is full. This bit is set to 1 when the USB Rx MailBox Read Address and the USB Rx MailBox Write Address match. This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 21 | RPE2<br>(Rx Pool2 Empty) | Bit that indicates that receive Pool2 is empty.<br>This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 20 | RPE1<br>(Rx Pool1 Empty) | Bit that indicates that receive Pool1 is empty.<br>This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 19 | RPE0<br>(Rx Pool0 Empty) | Bit that indicates that receive Pool0 is empty.<br>This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 18 | RPA1<br>(Rx Pool2 Alert) | This bit is set to 1 when the number of Buffer Directories remaining in receive Pool2 matches the value set with the Rx Pool2 Information Register. | R/Clear |
| 17 | RPA1<br>(Rx Pool1 Alert) | This bit is set to 1 when the number of Buffer Directories remaining in receive Pool1 matches the value set with the Rx Pool1 Information Register. | R/Clear |
| 16 | RPA0<br>(Rx Pool0 Alert) | This bit is set to 1 when the number of Buffer Directories remaining in receive Pool0 matches the value set with the Rx Pool0 Information Register.<br>This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 15-11 | Reserved | Reserved for future use | R |
| 10 | DER<br>(DMA Error) | Bit that indicates that an error occurred while DMA transfer was being performed. This bit is set to 1 if an error occurs on the Internal BUS during DMA transfer.<br>This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 9 | EP2FO<br>(EP2 FIFO Error) | Bit that indicates that an overrun has occurred for the FIFO of EndPoint2 (Isochronous OUT). If the FIFO becomes full while EndPoint2 is performing a transaction, USB Controller can no longer receive data, such that all subsequent data is discarded. Should this occur, this bit is set to 1.<br>This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |

(2/2)

| Bit | Field | Description | R/W |
|---|---|---|---|
| 8 | EP1FU<br>(EP1 FIFO Error) | Bit that indicates that an underrun has occurred for the FIFO of EndPoint1 (Isochronous IN). If the FIFO empties while EndPoint1 is performing a transaction,<br>USB Controller sends an NAK handshaking packet, instead of a data packet, to the Host PC. Should this occur, this bit is set to 1.<br>This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 7 | EP6RF<br>(EP6 Rx Finished ) | Bit that indicates that EndPoint6 (Interrupt OUT) has completed the receiving of a data segment.<br>This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 6 | EP5TF<br>(EP5 Tx Finished) | Bit that indicates that EndPoint5 (Interrupt IN) has completed the sending of a data segment. This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 5 | EP4RF<br>(EP4 Rx Finished) | Bit that indicates that EndPoint4 (Bulk OUT) has completed the receiving of a data segment.<br>The timing when this bit will be set varies with Rx Mode.<br>This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 4 | EP3TF<br>(EP3 Tx Finished) | Bit that indicates that EndPoint3 (Bulk IN) has completed the sending of a data segment.<br>This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 3 | EP2RF<br>(EP2 Rx Finished) | Bit that indicates that EndPoint2 (Isochronous OUT) has completed the receiving of a data segment.<br>The timing when this bit will be set varies with Rx Mode.<br>This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 2 | EP1TF<br>(EP1 Tx Finished) | Bit that indicates that EndPoint1 (Isochronous IN) has completed the sending of a data segment. This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 1 | EP0RF<br>(EP0 Rx Finished) | Bit that indicates that EndPoint0 (Control) has completed the receiving of a data segment. This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 0 | EP0TF<br>(EP0 Tx Finished) | Bit that indicates that EndPoint0 (Control) has completed the sending of a data segment.<br>This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |

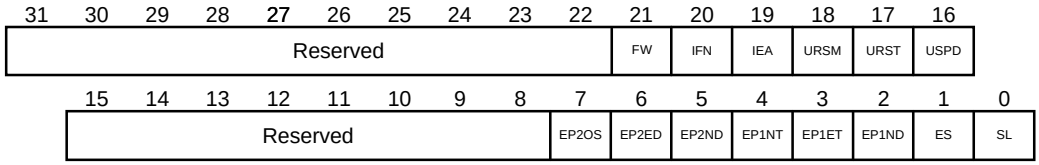
### 6.2.2.4 USB Interrupt Mask Register 1 (U_IMR1): 14H




This register is used to mask interrupts.

When a bit is set to 1, if the bit corresponding to the USB General Status Register1 (Address: 10H) is set to 1, an interrupt is issued.

### 6.2.2.5 USB General Status Register 2 (U_GSR2): 18H




This register indicates the current status of USB Controller.

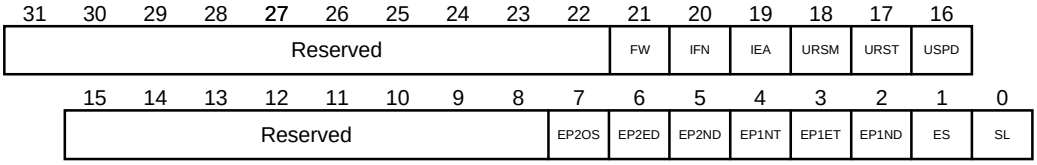
(1/2)

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-21 | Reserved | Reserved for future use | R |
| 21 | FW<br>(Frame Number Written) | This bit is set to 1 when Frame Number is written to “FrameNumber/Version Register”(04H). | |
| 20 | IFN<br>(Incorrect Frame Number) | This bit is set to 1 when received Frame Number is Incorrect. | R/Clear |
| 19 | IEA<br>(Incorrect EndPoint Access) | This bit is set to 1 when USB Controller received IN or OUT Token with incorrect EndPoint Number. This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 18 | URSM<br>(USB Resume) | Bit that indicates that USB Controller has received a Resume Signaling from the Host PC. When USB Controller detects Resume Signaling, it sets this bit to 1. This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 17 | URST<br>(USB Reset) | Bit that indicates that USB Controller has received a Reset Signaling from the Host PC. When USB Controller detects Resume Signaling, it sets this bit to 1.<br>This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 16 | USPD<br>(USB Suspend) | Bit that indicates that the USB has entered the Suspend status. When USB Controller detects the arrival of Suspend Signaling from the USB, this bit is set to 1. This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 15-8 | Reserved | Reserved for future use | R |

(2/2)

| Bit | Field | Description | R/W |
|---|---|---|---|
| 7 | EP2OS<br>(Over Size) | This bit is set to 1 when the received data size is over 1023Byte on EndPoint2. | R/Clear |
| 6 | EP2ED<br>(Extra Data on EndPoint2) | This bit is set to 1 when extra data packet is detected on Isochronous EndPoint (EndPoint2). | R/Clear |
| 5 | EP2ND<br>(Isochronous Data Corrupted) | Bit that indicates that Isochronous data is corrupted on EndPoint2.<br>This bit is reset to 0 when the VR4120A RISC Processor performs a read. | R/Clear |
| 4 | EP1NT<br>(No Token on EP1) | This bit is set to 1 when IN Token packet does not come on EndPoint1 between two SOFs. | R/Clear |
| 3 | EP1ET<br>(Extra Token on EP1) | This bit is set to 1 when IN Token packet comes twice on EndPoint1 between two SOFs. | R/Clear |
| 2 | EP1ND<br>(No Data on EP1) | This bit is set to 1 when IN Token packet comes but data is not ready on EndPoint1. | R/Clear |
| 1 | ES<br>(Extra SOF) | This bit is set to 1 when extra SOF packet is detected. | R/Clear |
| 0 | SL<br>(SOF Loss) | This bit is set to 1 when USB Controller doesn't receive SOF packet. | R/Clear |

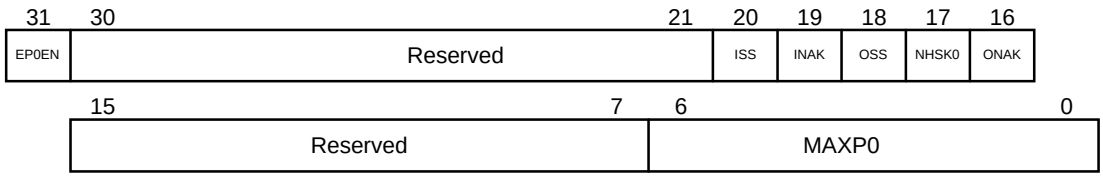
### 6.2.2.6 USB Interrupt Mask Register 2 (U_IMR2): 1CH




This register is used to mask interrupts.

When a bit is set to 1, if the bit corresponding to the USB General Status Register2 (Address: 18H) is set to 1, an interrupt is issued.

### 6.2.2.7 USB EP0 Control Register (U_EP0CR): 20H



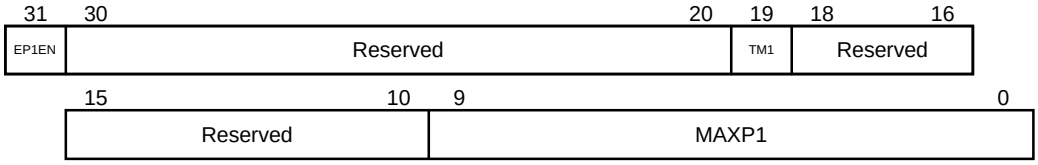

Register that is used for setting the operation of EndPoint0.

If the value in the MAXP field is rewritten during a send or receive operation, the operation of USB Controller may become unpredictable. Therefore the MAXP can be written to once only, when initial setting is being performed.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31 | EP0EN<br>(EndPoint Enable) | If the VR4120A RISC Processor sets this bit to 1, EndPoint0 is enabled for transmitting and receiving data from and to USB. | R/W |
| 30-19 | Reserved | Reserved for future use | R |
| 20 | ISS<br>(IN Send Stall) | If the VR4120A RISC Processor sets this bit to 1, STALL packet is sent at the Data Phase.<br>If SETUP packet is received when this bit is active, USB Controller clears this bit to 0. | R/W |
| 19 | INAK<br>(IN NAK) | If the VR4120A RISC Processor sets this bit to 1, NAK packet is sent at the Data Phase. | R/W |
| 18 | OSS<br>(OUT Send Stall) | If the VR4120A RISC Processor sets this bit to 1, STALL handshake is performed at the Handshake Phase. | R/W |
| 17 | NHSK0<br>(No Handshake) | If the VR4120A RISC Processor sets this bit to 1, No Handshake is performed at the Handshake Phase. | R/W |
| 16 | ONAK<br>(OUT NAK) | If the VR4120A RISC Processor sets this bit to 1, NAK Handshake is performed at the Handshake Phase.<br>If SETUP packet is received when this bit is active, USB Controller clears this bit to 0. | R/W |
| 15-7 | Reserved | Reserved for future use | R |
| 6-0 | MAXP0<br>(MAX Packet size) | Register that stores the Max Packet Size for EndPoint0. Prior to the start of a USB transaction, the VR4120A RISC Controller must write an appropriate value into this register.<br>When this field contains 0, no transaction is performed at EndPoint0. | R/W |

**Remark** When a STALL handshake is sent by a control endpoint in either the Data or Status stage of a control transfer, a STALL handshake must be returned on all succeeding access to that endpoint until a SETUP PID is received. The endpoint is not required to return a STALL handshake after it receives a subsequent SETUP PID. (referred from USB1.1 Specification)

### 6.2.2.8 USB EP1 Control Register (U_EP1CR): 24H

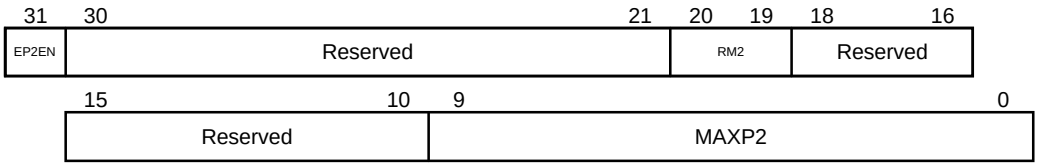



Register for setting the operation of EndPoint1.

If the value in the MAXP field is rewritten during a send or receive operation, the operation of USB Controller may become unpredictable. Therefore the MAXP can be written to once only, when initial setting is being performed.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31 | EP1EN<br>(EndPoint Enable) | If the VR4120A RISC Processor sets this bit to 1, EndPoint1 is enabled for transmitting and receiving data from and to USB. | R/W |
| 30-20 | Reserved | Reserved for future use | R |
| 19 | TM1<br>(Tx Mode) | Bit for setting the send mode.<br>When this bit is set to 0, sending is performed in SZLP Mode.<br>When this bit is set to 1, sending is performed in NZLP Mode.<br>For a detailed explanation of the send modes, see Section 6.5.3. | R/W |
| 18-10 | Reserved | Reserved for future use | R |
| 9-0 | MAXP1<br>(MAX Packet size) | Register that stores the Max Packet Size of EndPoint1. Prior to the start of a USB transaction, the VR4120A RISC Processor must write an appropriate value into this register.<br>When this field contains 0, no transaction is performed at EndPoint1. | R/W |

### 6.2.2.9 USB EP2 Control Register (U_EP2CR): 28H

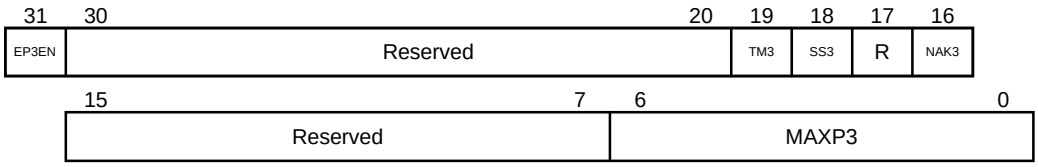



Register for setting the operation of EndPoint2.

If the value in the MAXP field is rewritten during a send or receive operation, the operation of USB Controller may become unpredictable. Therefore the MAXP can be written to once only, when initial setting is being performed.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31 | EP2EN<br>(EndPoint Enable) | If the VR4120A RISC Processor sets this bit to 1, EndPoint2 is enabled for transmitting and receiving data from and to USB. | R/W |
| 30-20 | Reserved | Reserved for future use | R |
| 20-19 | RM2<br>(Rx Mode) | Bit for setting the receive mode.<br>When this bit is set to 00 or 01, receiving is performed in Normal Mode.<br>When this bit is set to 10, receiving is performed in Assemble Mode.<br>When this bit is set to 11, receiving is performed in Separate Mode.<br>For a detailed explanation of the receive modes, see Section 6.6.4. | R/W |
| 18-10 | Reserved | Reserved for future use | R |
| 9-0 | MAXP2<br>(MAX Packet size) | Register that stores the Max Packet Size of EndPoint2. Prior to the start of a USB transaction, the VR4120A RISC Processor must write an appropriate value into this register.<br>When this field contains 0, no transaction is performed at EndPoint2. | R/W |

### 6.2.2.10 USB EP3 Control Register (U_EP3CR): 2CH

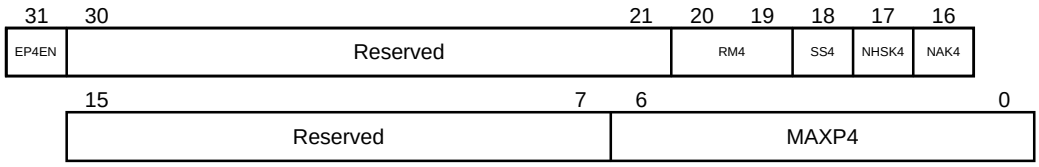



Register for setting the operation of EndPoint3.

If the value in the MAXP field is rewritten during a send or receive operation, the operation of USB Controller may become unpredictable. Therefore the MAXP can be written to once only, when initial setting is being performed.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31 | EP3EN<br>(EndPoint Enable) | If the VR4120A RISC Processor sets this bit to 1, EndPoint3 is enabled for transmitting and receiving data from and to USB. | R/W |
| 30-20 | Reserved | Reserved for future use | R |
| 19 | TM3<br>(Tx Mode) | Bit for setting the send mode.<br>When this bit is set to 0, sending is performed in SZLP Mode.<br>When this bit is set to 1, sending is performed in NZLP Mode.<br>For a detailed explanation of the send modes, see Section 6.5.3. | R/W |
| 18 | SS3<br>(Send Stall) | If the VR4120A RISC Processor sets this bit to 1, STALL packet is sent by EndPoint3. | R/W |
| 17 | Reserved | Reserved for future use | R |
| 16 | NAK3 | If the VR4120A RISC Processor sets this bit to 1, NAK packet is sent by EndPoint3. | R/W |
| 15-7 | Reserved | Reserved for future use | R |
| 6-0 | MAXP3<br>(MAX Packet size) | Register that stores the Max Packet Size for EndPoint3. Prior to the start of a USB transaction, the VR4120A RISC Processor must write an appropriate value into this register.<br>When this field contains 0, no transaction is performed at EndPoint3. | R/W |

### 6.2.2.11 USB EP4 Control Register (U_EP4CR): 30H

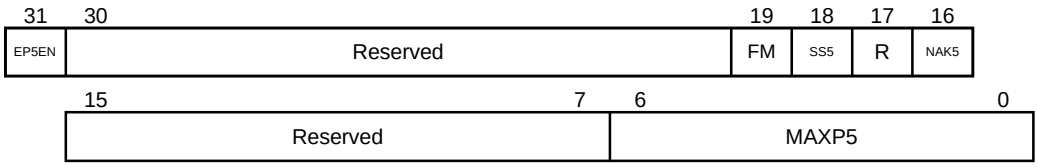



Register for setting the operation of and EndPoint4.

If the value in the MAXP field is rewritten during a send or receive operation, the operation of USB Controller may become unpredictable. Therefore the MAXP can be written to once only, when initial setting is being performed.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31 | EP4EN<br>(EndPoint Enable) | If the VR4120A RISC processor sets this bit to 1, EndPoint4 is enabled for transmitting and receiving data from and to USB. | R/W |
| 30-21 | Reserved | Reserved for future use | R |
| 20-19 | RM4<br>(Rx Mode) | Bit for setting the send mode.<br>When this bit is set to 0, sending is performed in SZLP Mode.<br>When this bit is set to 1, sending is performed in NZLP Mode.<br>For a detailed explanation of the send modes, see Section 6.5.3. | R/W |
| 18 | SS4<br>(Send Stall) | If the VR4120A RISC Processor sets this bit to 1, STALL handshake is performed at EndPoint4. | R/W |
| 17 | NHSK4<br>(No Handshake) | If the VR4120A RISC Processor sets this bit to 1, No Handshake is performed at EndPoint4. | R/W |
| 16 | NAK4 | If the VR4120A RISC Processor sets this bit to 1, NAK Handshake is performed at EndPoint4. | R/W |
| 15-7 | Reserved | Reserved for future use | R |
| 6-0 | MAXP4<br>(MAX Packet size) | Register that stores the Max Packet Size for EndPoint4. Prior to the start of a USB transaction, the VR4120A RISC Processor must write an appropriate value into this register.<br>When this field contains 0, no transaction is performed at EndPoint4. | R/W |

### 6.2.2.12 USB EP5 Control Register (U_EP5CR): 34H

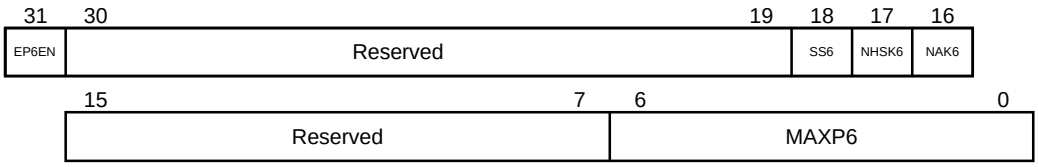



Register for setting the operation of EndPoint5.

If the value in the MAXP field is rewritten during a send or receive operation, the operation of USB Controller may become unpredictable. Therefore the MAXP can be written to once only, when initial setting is being performed.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31 | EP5EN<br>(EndPoint Enable) | If the VR4120A RISC processor sets this bit to 1, EndPoint5 is enabled for transmitting and receiving data from and to USB. | R/W |
| 30-20 | Reserved | Reserved for future use | R |
| 19 | FM<br>(Feedback Mode) | If the VR4120A RISC Processor sets this bit to 1, EndPoint5 performs in feedback mode.<br>(For further information about Feedback mode, please refer to USB Specification 1.1) | R/W |
| 18 | SS5<br>(Send Stall) | If the VR4120A RISC Processor sets this bit to 1, STALL handshake is performed at EndPoint5. | R/W |
| 17 | Reserved | Reserved for future use | R |
| 16 | NAK5 | If the VR4120A RISC processor sets this bit to 1, NAK Handshake is performed at EndPoint5. | R/W |
| 15-7 | Reserved | Reserved for future use | R |
| 6-0 | MAXP5<br>(MAX Packet size) | Register that stores the Max Packet Size for EndPoint5. Prior to the start of a USB transaction, the VR4120A RISC Processor must write an appropriate value into this register.<br>When this field contains 0, no transaction is performed at EndPoint5. | R/W |

### 6.2.2.13 USB EP6 Control Register (U_EP6CR): 38H

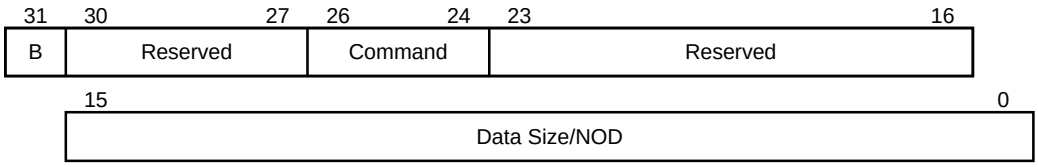



Register for setting the operation of EndPoint6.

If the value in the MAXP field is rewritten during a send or receive operation, the operation of USB Controller may become unpredictable. Therefore the MAXP can be written to once only, when initial setting is being performed.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31 | EP6EN<br>(EndPoint Enable) | If the VR4120A RISC Processor sets this bit to 1, EndPoint6 is enabled for transmitting and receiving data from and to USB. | R/W |
| 30-19 | Reserved | Reserved for future use | R |
| 18 | SS6<br>(Send Stall) | If the VR4120A RISC Processor sets this bit to 1, STALL handshake is performed at EndPoint6. | R/W |
| 17 | NHSK6<br>(No Handshake) | If the VR4120A RISC Processor sets this bit to 1, No Handshake is performed at EndPoint6. | R/W |
| 16 | NAK6 | If the VR4120A RISC Processor sets this bit to 1, NAK Handshake is performed at EndPoint6. | R/W |
| 15-7 | Reserved | Reserved for future use | R |
| 6-0 | MAXP6<br>(MAX Packet size) | Register that stores the Max Packet Size for EndPoint6. Prior to the start of a USB transaction, the VR4120A RISC Processor must write an appropriate value into this register.<br>When this field contains 0, no transaction is performed at EndPoint6. | R/W |

### 6.2.2.14 USB Command Register (U_CMR): 40H



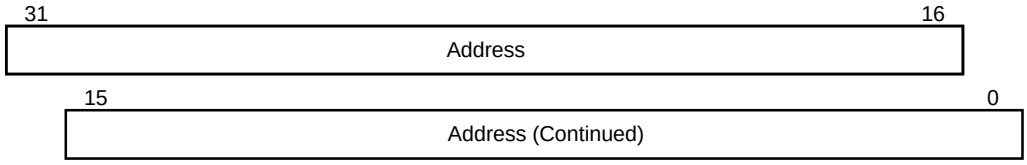

Register for adding data segment send or receive Buffer Directories.

The VR4120A RISC Processor writes commands into this register.

Whenever B bit (Bit 31) is set, even if VR4120A RISC Processor writes commands into this register, the value will not change.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31 | B<br>(Busy) | Bit that indicates whether the interpretation of an issued command has terminated. When the execution of the command has not yet terminated, this bit will be set to 1. When the execution of a command has terminated, this bit will be set to 0.<br>When the VR4120A RISC Processor issues one command immediately after another, it is necessary to confirm that this bit is set to 0. | R |
| 30-27 | Reserved | Reserved for future use | R/W |
| 26-24 | Command | Field for specifying the type of a command. USB Controller's internal processing varies depending on the value written into this field.<br>000: Data sending at EndPoint0<br>001: Data sending at EndPoint1<br>010: Data sending at EndPoint3<br>011: Data sending at EndPoint5<br>100: Addition of Buffer Directories to Pool0<br>101: Addition of Buffer Directories to Pool1<br>110: Addition of Buffer Directories to Pool2<br>111: Reserved (Don't Use) | R/W |
| 23-16 | Reserved | Reserved for future use | R |
| 15-0 | Data Size/NOD<br>(Data Size / Number Of Buffer Directory) | The meaning of this field depends on the value written into the EPN field.<br>EPN=0xx: This field has no meaning.<br>EPN=100,101,110: Indicates the number of Buffer Directories added to a pool.<br>EPN=111: This field has no meaning. | R/W |

### 6.2.2.15 USB Command Address Register (U_CA): 44H

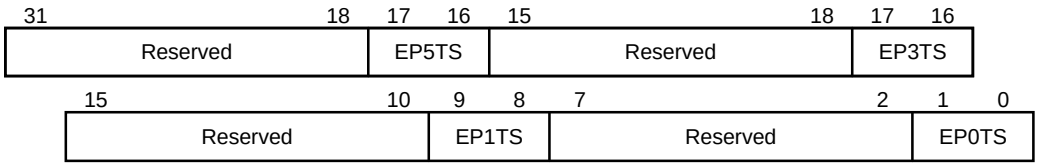



Register for adding data segment for send or receive Buffer Directories.

The VR4120A RISC Processor writes the head address of either the send data or the receive Buffer Directory into this register.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Address | The meaning of this field varies depending on the value written into the EPN field of the USB Tx Rx Command Register.<br>EPN=0xx: Head address of the send packet<br>EPN=100,101,110: Head address of the Buffer Directory to be added to the receive pool<br>EPN=111: This register has no meaning. | R/W |

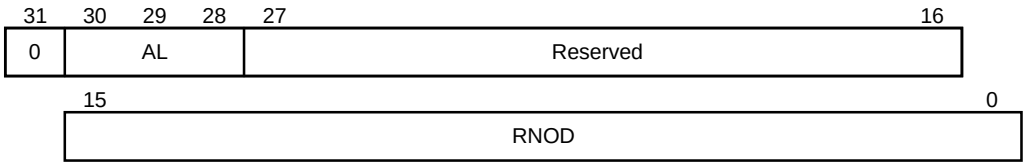
### 6.2.2.16 USB Tx EndPoint Status Register (U_TEPSR): 48H




Register that indicates the status of the EndPoint being used for data sending.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-26 | Reserved | Reserved for future use | R |
| 25-24 | EP5TS<br>(EP5 Tx Status) | Register that indicates the send status of EndPoint5<br>This register is not cleared, even if read.<br>00: There is no data scheduled to be sent (Idle)<br>01: There is one data item scheduled to be sent<br>10: There are two data items that are scheduled to be sent (Busy) | R |
| 23-18 | Reserved | Reserved for future use | R |
| 17-16 | EP3TS<br>(EP3 Tx Status) | Register that indicates the send status of EndPoint3<br>This register is not cleared, even if read.<br>00: There is no data scheduled to be sent (Idle)<br>01: There is one data item scheduled to be sent<br>10: There are two data items that are scheduled to be sent (Busy) | R |
| 15-10 | Reserved | Reserved for future use | R |
| 9-8 | EP1TS<br>(EP1 Tx Status) | Register that indicates the send status of EndPoint1<br>This register is not cleared, even if read.<br>00: There is no data scheduled to be sent (Idle)<br>01: There is one data item scheduled to be sent<br>10: There are two data items that are scheduled to be sent (Busy) | R |
| 7-2 | Reserved | Reserved for future use | R |
| 1-0 | EP0TS<br>(EP1 Tx Status) | Register that indicates the send status of EndPoint0<br>This register is not cleared, even if read.<br>00: There is no data scheduled to be sent (Idle)<br>01: There is one data item scheduled to be sent<br>10: There are two data items that are scheduled to be sent (Busy) | R |

### 6.2.2.17 USB Rx Pool0 Information Register (U_RP0IR): 50H

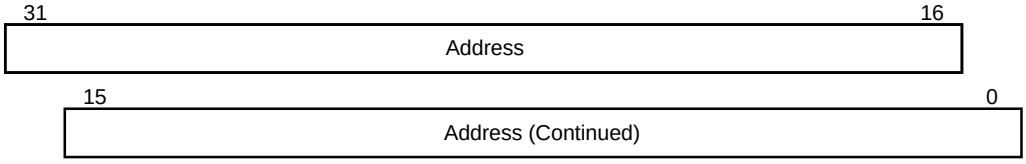



Register that indicates the information of Receive Pool0.

The VR4120A RISC Processor write to this register once only when the device is being initialized.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31 | Reserved | Reserved for future use<br>Write 0 to this bit. | R |
| 30-28 | AL<br>(Alert Level) | Sets the warning level for Pool0. If the number of Buffer Directories remaining in this pool equals the value set in this field, USB Controller sets the RPA0 bit of the USB General Status Register1 to 1.<br>Writing n into this field is equivalent to specifying n x 4 (remaining number of Buffer Directories = 4, 8, 12, ..., 28). When 000 is written into this field, this function is disabled and no notification is posted to the VR4120A RISC Processor. | R/W |
| 27-16 | Reserved | Reserved for future use | R |
| 15-0 | RNOD<br>(Remaining Number of Buffer Directory) | Indicates the number of Buffer Directories remaining in Pool0. The VR4120A RISC Processor can only read this field.<br>Unlike the μPD98401A and the μPD98405, there is no need to write a value into this field during initialization. Buffer Directory addition is performed entirely using the USB Tx Rx Command Register. | R |

### 6.2.2.18 USB Rx Pool0 Address Register (U_RP0AR): 54H

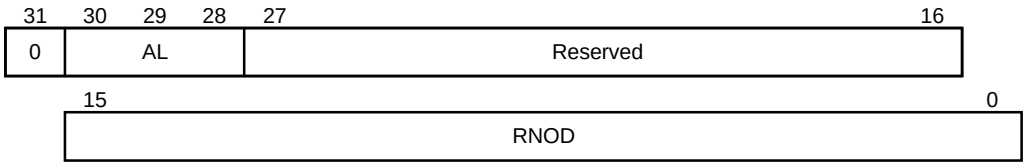



This register indicates the head address of Buffer Directory which is currently used.

The way to set up Rx Pool is described at Section 6.6.3.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Buffer Directory<br>Address | Register that indicates the head address of the first Buffer Directory in Pool0.<br>The VR4120A RISC Processor can only read this register.<br>Unlike the μPD98401A and the μPD98405, there is no need to write a value into this register during initialization. Buffer Directory addition is performed entirely using the USB Tx Rx Command Register. | R |

### 6.2.2.19 USB Rx Pool1 Information Register (U_RP1IR): 58H

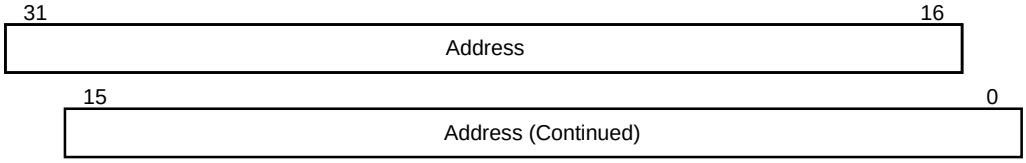



Register that indicates the data in Receive Pool1.

The VR4120A RISC Processor write to this register once only when the device is being initialized.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31 | Reserved | Reserved for future use<br>Write 0 to this bit. | R |
| 30-28 | AL<br>(Alert Level) | Sets the warning level for Pool1. If the number of Buffer Directories remaining in this pool equals the value set in this field, USB Controller sets the RPA1 bit of the USB General Status Register1 to 1.<br>Writing n into this field is equivalent to specifying n x 4 (remaining number of Buffer Directories = 4, 8, 12, ..., 28). When 000 is written into this field, this function is disabled and no notification is posted to the VR4120A RISC Processor. | R/W |
| 27-16 | Reserved | Reserved for future use | R |
| 15-0 | RNOD<br>(Remaining Number of Buffer Directory) | Indicates the number of Buffer Directories remaining in Pool1. The VR4120A RISC Processor can only read this field.<br>Unlike the μPD98401A and the μPD98405, there is no need to write a value into this field during initialization. Buffer Directory addition is performed entirely using the USB Tx Rx Command Register. | R |

### 6.2.2.20 USB Rx Pool1 Address Register (U_RP1AR): 5CH

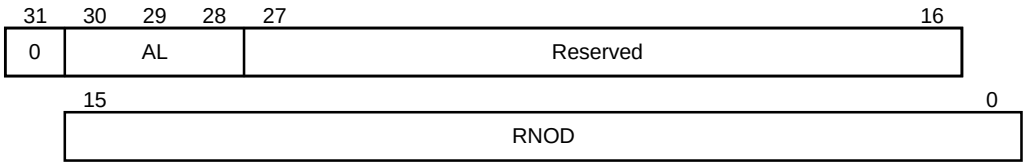



This register indicates the head address of Buffer Directory which is currently used.

The way to set up Rx Pool is described at Section 6.6.3.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Buffer Directory<br>Address | Register that indicates the head address of the first Buffer Directory in Pool1.<br>The VR4120A RISC Processor can only read this register.<br>Unlike the μPD98401A and the μPD98405, there is no need to write a value into this register during initialization. Buffer Directory addition is performed entirely using the USB Tx Rx Command Register. | R |

### 6.2.2.21 USB Rx Pool2 Information Register (U_RP2IR): 60H

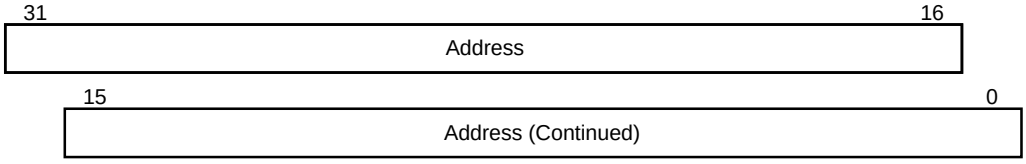



Register that indicates the data in Receive Pool2.

The VR4120A RISC Processor write to this register once only when the device is being initialized.

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31 | Reserved | Reserved for future use<br>Write 0 to this bit. | R |
| 30-28 | AL<br>(Alert Level) | Sets the warning level for Pool2. If the number of Buffer Directories remaining in this pool equals the value set in this field, USB Controller sets the RPA2 bit of the USB General Status Register1 to 1.<br>Writing n into this field is equivalent to specifying n x 4 (remaining number of Buffer Directories = 4, 8, 12, ..., 28). When 000 is written into this field, this function is disabled and no notification is posted to the VR4120A RISC Processor. | R/W |
| 27-16 | Reserved | Reserved for future use | R |
| 15-0 | RNOD<br>(Remaining Number of Buffer Directory) | Indicates the number of Buffer Directories remaining in Pool2. The VR4120A RISC Processor can only read this field.<br>Unlike the μPD98401A and the μPD98405, there is no need to write a value into this field during initialization. Buffer Directory addition is performed entirely using the USB Tx Rx Command Register. | R |

### 6.2.2.22 USB Rx Pool2 Address Register (U_RP2AR): 64H

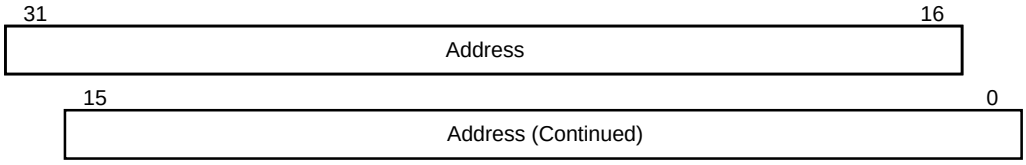



This register indicates the head address of Buffer Directory which is currently used.

The way to set up Rx Pool is described at Section 6.6.3.

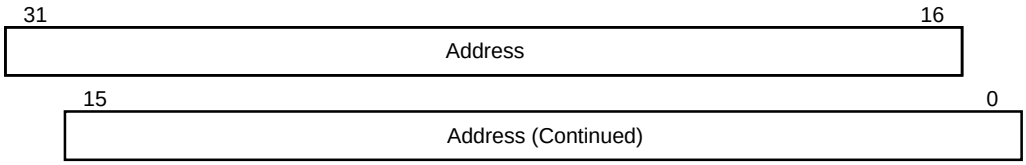
| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Buffer Directory<br>Address | Register that indicates the head address of the first Buffer Directory in Pool2.<br>The VR4120A RISC Processor can only read this register.<br>Unlike the μPD98401A and the μPD98405, there is no need to write a value into this register during initialization. Buffer Directory addition is performed entirely using the USB Tx Rx Command Register. | R |

### 6.2.2.23 USB Tx MailBox Start Address Register (U_TMSA): 70H



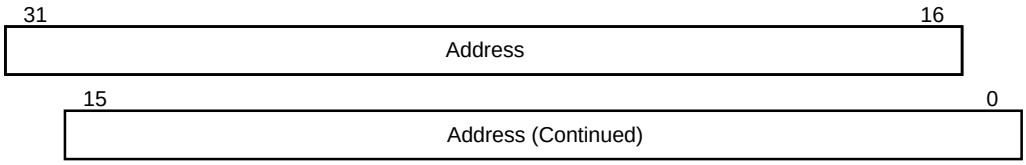

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Address | Register that indicates the head address of the send MailBox area. The VR4120A RISC Processor need set a value in this field once only, at initialization. | R/W |

### 6.2.2.24 USB Tx MailBox Bottom Address Register (U_TMBA): 74H



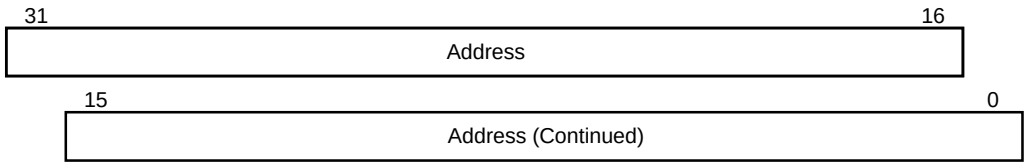

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Address | Register that indicates the end address of the send MailBox area. The VR4120A RISC Processor need set a value in this field once only, at initialization. | R/W |

### 6.2.2.25 USB Tx MailBox Read Address Register (U_TMRA): 78H



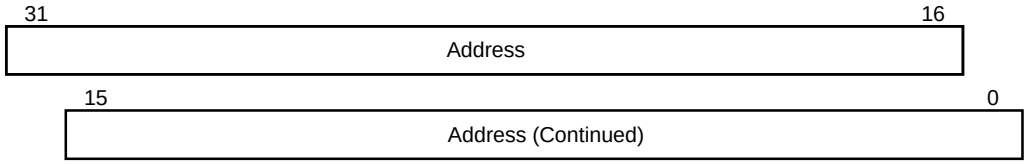

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Address | Register that indicates the address of the area that will be read next by the VR4120A RISC processor. After the VR4120A RISC Processor reads the contents of a MailBox, the value set in this register must be changed by VR4120A RISC Processor. | R/W |

### 6.2.2.26 USB Tx MailBox Write Address Register (U_TMWA): 7CH



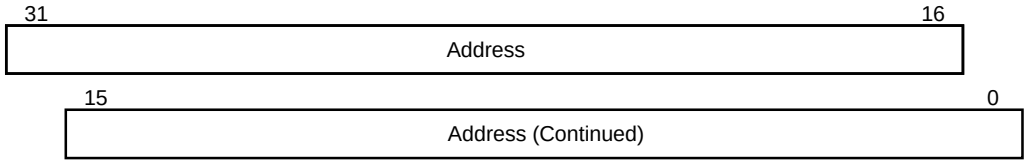

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Address | Register that indicates the address in the send mailbox area to which USB Controller will next write. | R |

### 6.2.2.27 USB Rx MailBox Start Address Register (U_RMSA): 80H



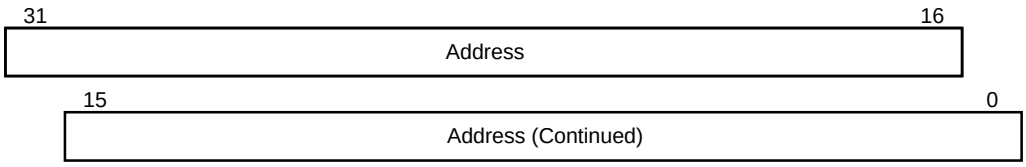

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Address | Register that indicates the head address of the receive mailbox area. The VR4120A RISC Processor need set a value in this field once only, at initialization. | R/W |

### 6.2.2.28 USB Rx MailBox Bottom Address Register (U_RMBA): 84H



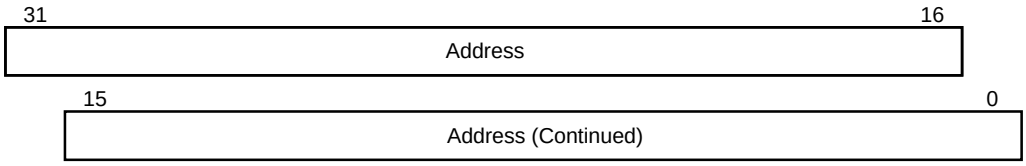

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Address | Register that indicates the end address of the receive mailbox area. The VR4120A RISC Processor need set a value in this field once only, at initialization. | R/W |

### 6.2.2.29 USB Rx MailBox Read Address Register (U_RMRA): 88H



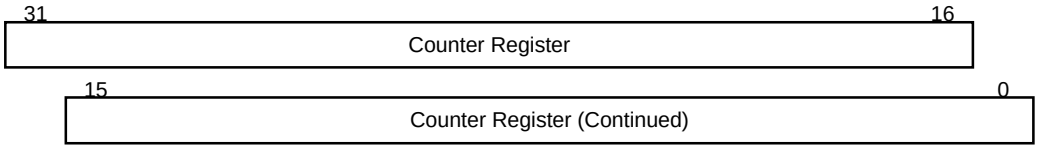

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Address | Register that indicates the address of the area that will be read next by the VR4120A RISC Processor. After the VR4120A RISC Processor reads the contents of a mailbox, the value set in this register must be changed by VR4120A RISC Processor. | R/W |

### 6.2.2.30 USB Rx MailBox Write Address Register (U_RMWA): 8CH



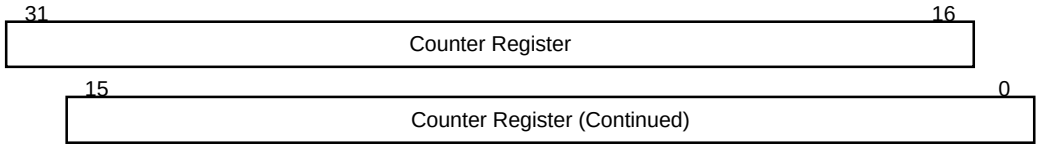

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Address | Register that indicates the address in the receive mailbox area to which USB Controller will next write. | R |

### 6.2.2.31 USB EP0 Tx Data Phase NAK Counter (U_TDNC) : 100H



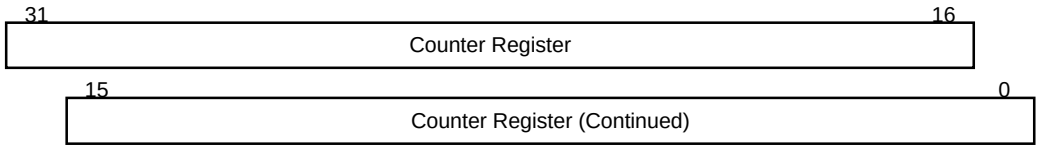

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register<br>(NAK Count) | Free-run counter that indicates the number of times that NAK Handshake is performed during Tx Data Phase on EndPoint0. | R |

### 6.2.2.32 USB EP0 Tx Data Phase STALL Counter (U_TDS): 104H



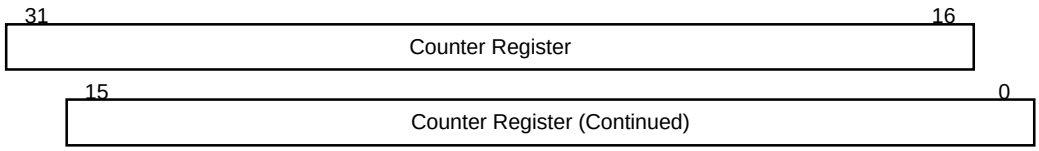

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (STALL Count) | Free-run counter that indicates the number of times STALL handshake has performed during Tx Data Phase on EndPoint0. | R |

### 6.2.2.33 USB EP0 Tx Handshake Phase Timeout Counter (U_THT) : 110H



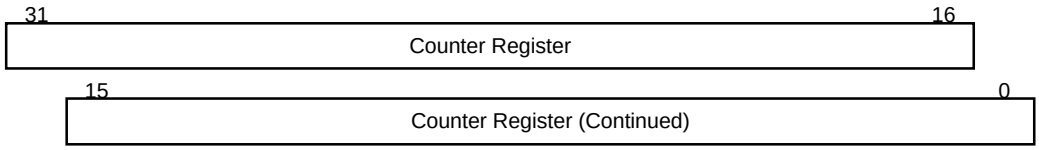

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (Timeout Count) | Free-run counter that indicates the number of times that Timeout has occurred during Tx Handshake Phase on EndPoint0. | R |

### 6.2.2.34 USB EP0 Rx Data Phase Timeout Counter (U_RDT): 120H



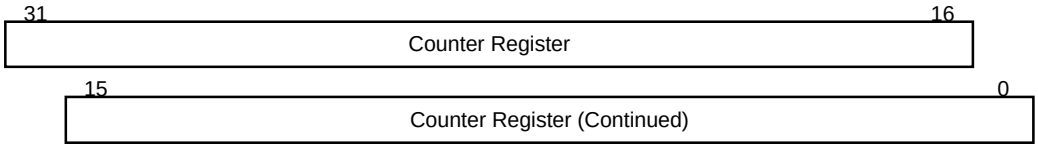

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (Timeout Count) | Free-run counter that indicates the number of times that Timeout has occurred during Rx Data Phase on EndPoint0. | R |

### 6.2.2.35 USB EP0 Rx Data Phase CRC Error Counter (U_RDCER) : 124H



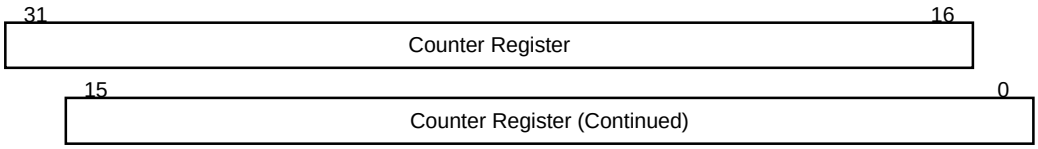

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (CRC Error Count) | Free-run counter that indicates the number of times that a CRC Error has occurred during Rx Data Phase on EndPoint0. | R |

### 6.2.2.36 USB EP0 Rx Data Phase Bitstuff Error Counter (U_RDBER) : 128H



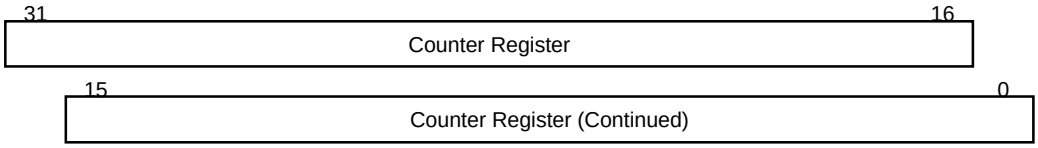

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register<br>(Bit Stuff Error Count) | Free-run counter that indicates the number of times that a Bit Stuff Error has occurred during Rx Data Phase on EndPoint0. | R |

### 6.2.2.37 USB EP0 Rx Data Phase Data Toggle Error Counter (U_RDTER) : 12CH



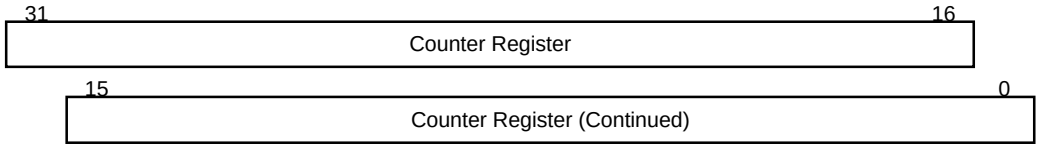

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register<br>(Data Toggle Error Count) | Free-run counter that indicates the number of times that a Data Toggle Error has occurred during Rx Data Phase on EndPoint0. | R |

### 6.2.2.38 USB EP0 Rx Handshake Phase Timeout Counter (U_RHT): 130H



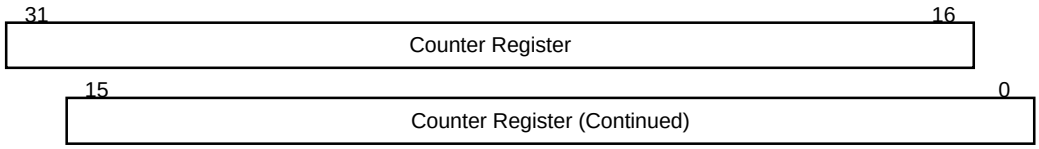

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register<br>(Timeout Count) | Free-run counter that indicates the number of times that Timeout is performed during Rx Handshake Phase on EndPoint0. | R |

### 6.2.2.39 USB EP0 Rx Handshake Phase NAK Counter (U_RHN): 134H



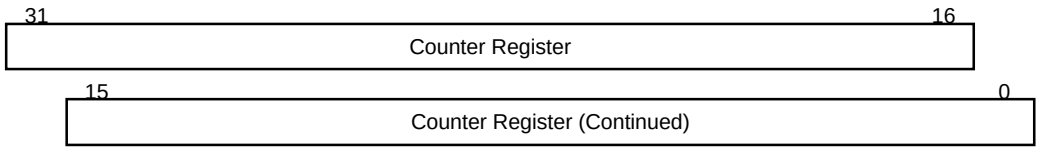

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (NAK Count) | Free-run counter that indicates the number of times that NAK Handshake is performed during Rx Handshake Phase on EndPoint0. | R |

### 6.2.2.40 USB EP0 Rx Handshake Phase STALL Counter (U_RHS): 138H



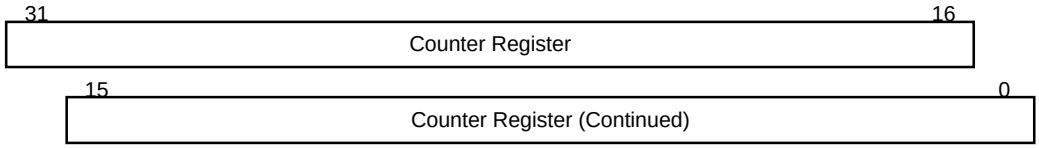

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (STALL Count) | Free-run counter that indicates the number of times that STALL Handshake is performed during Rx Handshake on EndPoint0. | R |

### 6.2.2.41 USB EP2 Data Phase Timeout Counter (U_DT2) : 160H



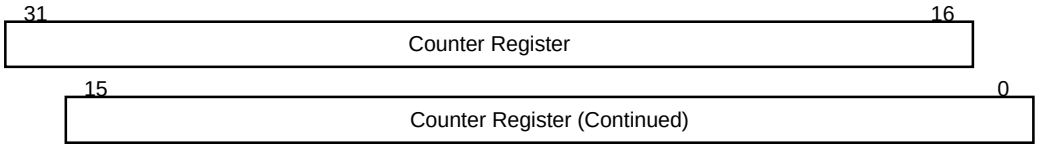

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (Timeout Count) | Free-run counter that indicates the number of times that a Timeout has occurred during Data Phase on EndPoint2. | R |

### 6.2.2.42 USB EP2 Data Phase CRC Error Counter (U_DCER2) : 164H



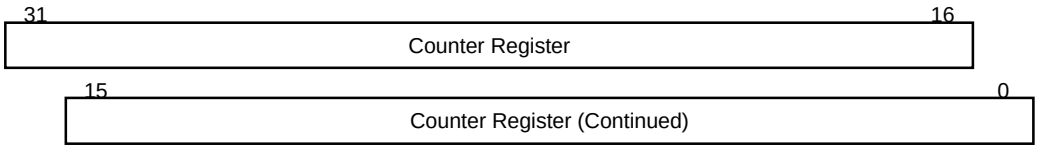

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (CRC Error Count) | Free-run counter that indicates the number of times that a CRC Error has occurred during Data Phase on EndPoint2. | R |

### 6.2.2.43 USB EP2 Data Phase Bit Stuffing Error Counter (U_DBER2) : 168H



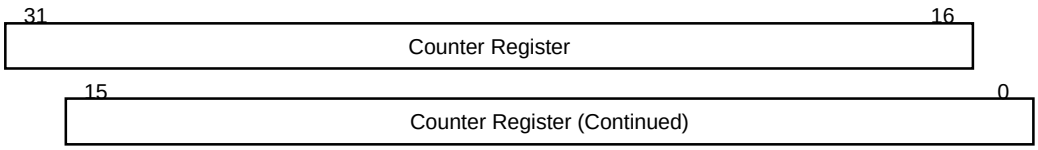

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (Bit Stuffing Error Count) | Free-run counter that indicates the number of times that a Bit Stuffing Error has occurred during Data Phase on EndPoint2. | R |

### 6.2.2.44 USB EP3 Data Phase NAK Counter (U_DN3) : 180H



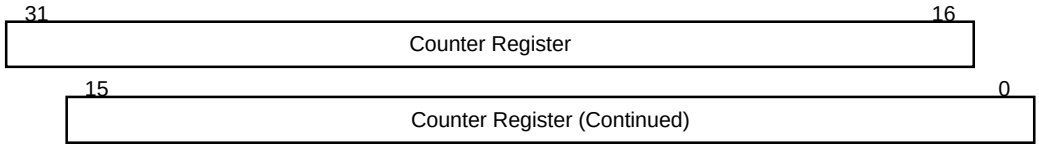

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (NAK Sent Count) | Free-run counter that indicates the number of times that NAK Handshake is performed during Data Phase on EndPoint3. | R |

### 6.2.2.45 USB EP3 Data Phase STALL Counter (U_DS3) : 184H



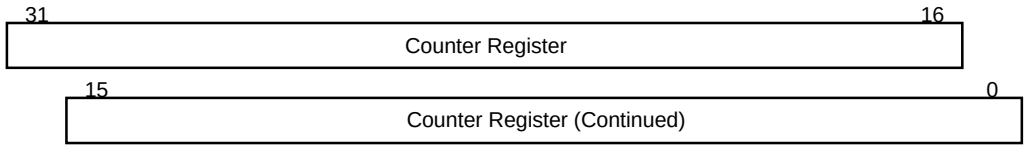

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (STALL Count) | Free-run counter that indicates the number of times that STALL Handshake is performed during Data Phase on EndPoint3. | R |

### 6.2.2.46 USB EP3 Handshake Phase Timeout Counter (U_HT3) : 190H



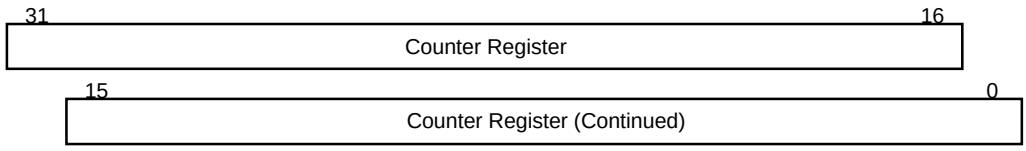

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (Timeout Count) | Free-run counter that indicates the number of times that Timeout has occurred during Handshake Phase on EndPoint3. | R |

### 6.2.2.47 USB EP4 Data Phase Timeout Counter (U_DT4) : 1A0H



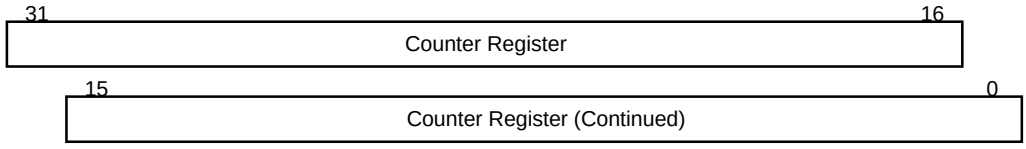

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (Timeout Count) | Free-run counter that indicates the number of times Timeout has occurred during Data Phase on EndPoint4. | R |

### 6.2.2.48 USB EP4 Data Phase CRC Error Counter (U_DCER4) : 1A4H



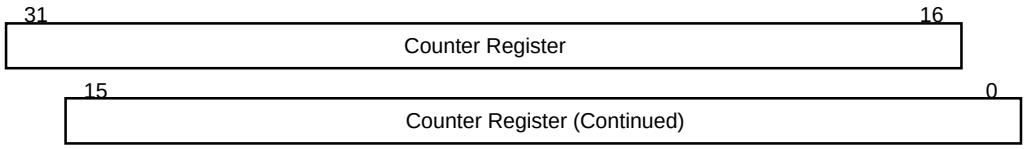

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (CRC Error Count) | Free-run counter that indicates the number of times that a CRC Error has occurred during Data Phase on EndPoint4. | R |

### 6.2.2.49 USB EP4 Data Phase Bit Stuff Error Counter (U_DBER4) : 1A8H



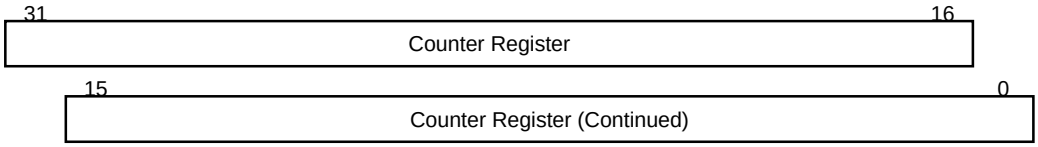

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (Bit Stuff Error Count) | Free-run counter that indicates the number of times that a Bit Stuff Error has occurred during Data Phase on EndPoint4. | R |

### 6.2.2.50 USB EP4 Data Phase Data Toggle Error Counter (U_DTER4) : 1ACH



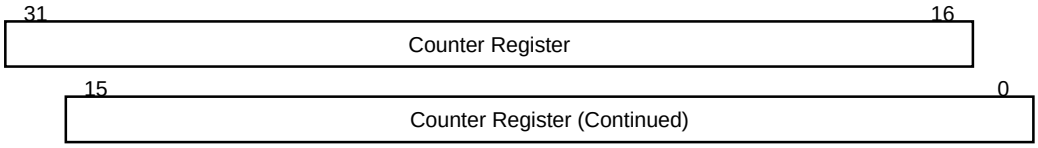

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register<br>(Data Toggle Error Count) | Free-run counter that indicates the number of times that a Data Toggle Error has occurred during Data Phase on EndPoint4. | R |

### 6.2.2.51 USB EP4 Handshake Phase Timeout Counter (U_HT4) : 1B0H



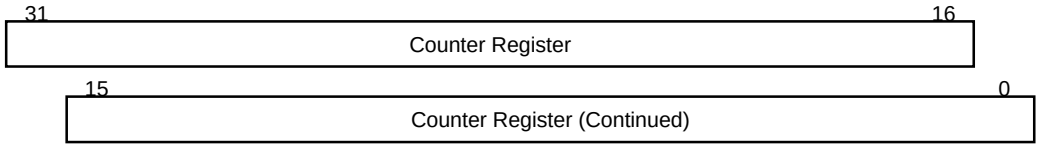

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register<br>(Timeout Count) | Free-run counter that indicates the number of times that Timeout is performed during Handshake Phase on EndPoint4. | R |

### 6.2.2.52 USB EP4 Handshake Phase NAK Counter (U_HN4) : 1B4H



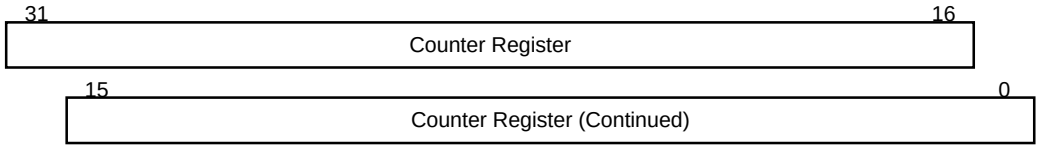

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register<br>(NAK Count) | Free-run counter that indicates the number of times that NAK handshake is performed during Handshake Phase on EndPoint4. | R |

### 6.2.2.53 USB EP4 Handshake Phase STALL Counter (U_HS4) : 1B8H



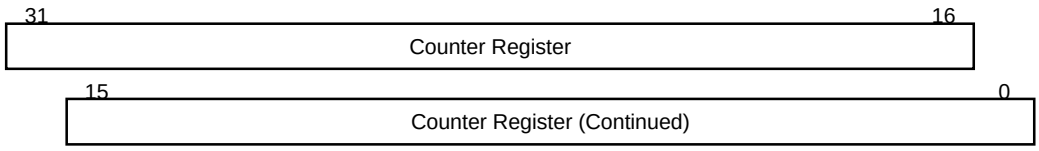

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (STALL Count) | Free-run counter that indicates the number of times that STALL Handshake is performed during Handshake Phase on EndPoint4. | R |

### 6.2.2.54 USB EP5 Data Phase NAK Counter (U_DN5) : 1C0H



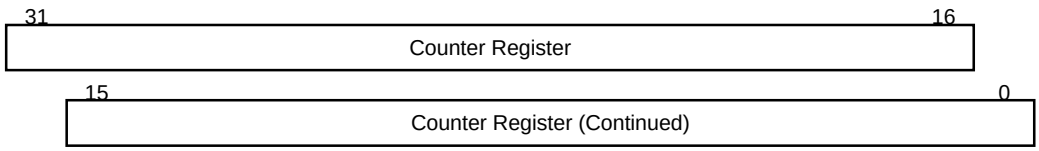

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (NAK Count) | Free-run counter that indicates the number of times that NAK handshake is performed during Data Phase on EndPoint5. | R |

### 6.2.2.55 USB EP5 Data Phase STALL Counter (U_DS5) : 1C4H



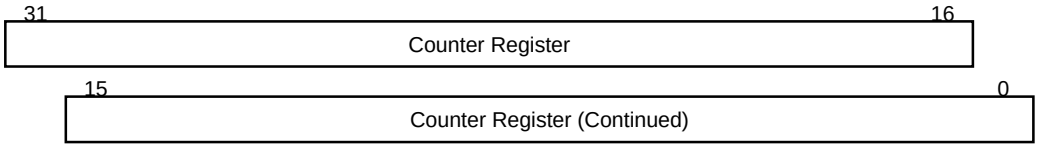

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (STALL Count) | Free-run counter that indicates the number of times that STALL Handshake is performed during Data Phase on EndPoint5. | R |

### 6.2.2.56 USB EP5 Handshake Phase Timeout Counter (U_HT5) : 1D0H



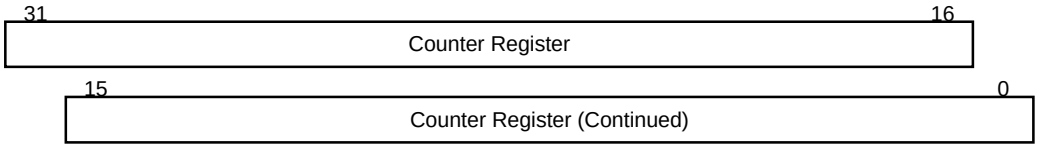

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (Timeout Count) | Free-run counter that indicates the number of times that Timeout has occurred during Handshake Phase on EndPoint5. | R |

### 6.2.2.57 USB EP6 Data Phase Timeout Counter (U_DT6) : 1E0H



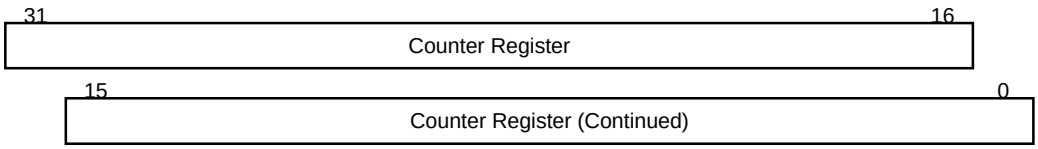

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (Timeout Count) | Free-run counter that indicates the number of times Timeout has occurred during Data Phase on EndPoint6. | R |

### 6.2.2.58 USB EP6 Data Phase CRC Error Counter (U_DCER6) : 1E4H



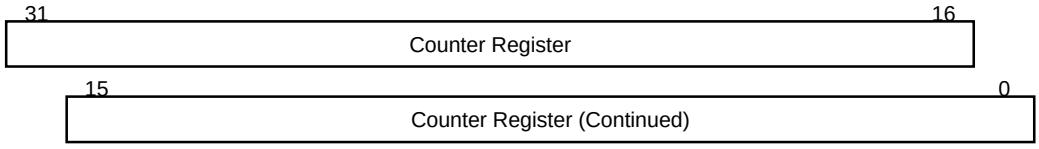

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (CRC Error Count) | Free-run counter that indicates the number of times that CRC Error has occurred during Data Phase on EndPoint6. | R |

### 6.2.2.59 USB EP6 Data Phase Bit Stuff Error Counter (U_EBER6) : 1E8H



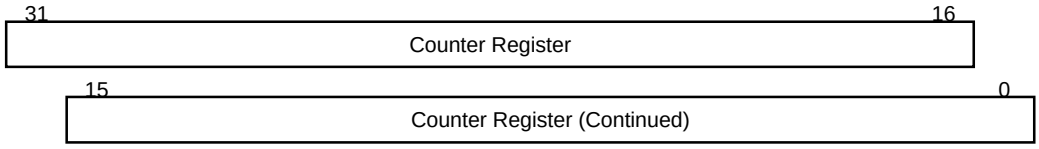

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (Bit Stuff Error Count) | Free-run counter that indicates the number of times that Bit Stuff Error has occurred during Data Phase on EndPoint6. | R |

### 6.2.2.60 USB EP6 Data Phase Data Toggle Error Counter (U_DTER6): 1ECH



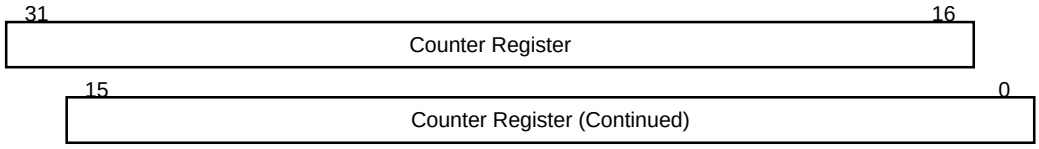

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (Data Toggle Error Count) | Free-run counter that indicates the number of times that Data Toggle Error has occurred during Data Phase on EndPoint6. | R |

### 6.2.2.61 USB EP6 Handshake Phase Timeout Counter (U_HT6) : 1F0H



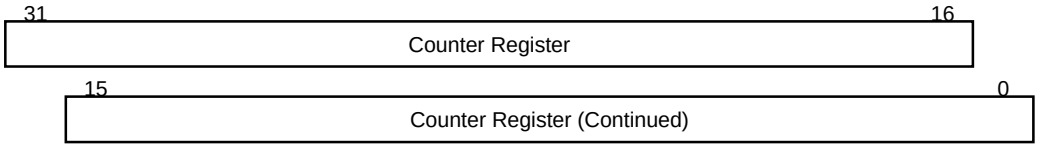

| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (Timeout Count) | Free-run counter that indicates the number of times that Timeout is performed during Handshake Phase on EndPoint6. | R |

### 6.2.2.62 USB EP6 Handshake Phase NAK Counter (U_HN6) : 1F4H




| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (NAK Count) | Free-run counter that indicates the number of times that NAK Handshake is performed during Handshake Phase on EndPoint6. | R |

### 6.2.2.63 USB EP6 Handshake Phase STALL Counter (U_HS6) : 1F8H

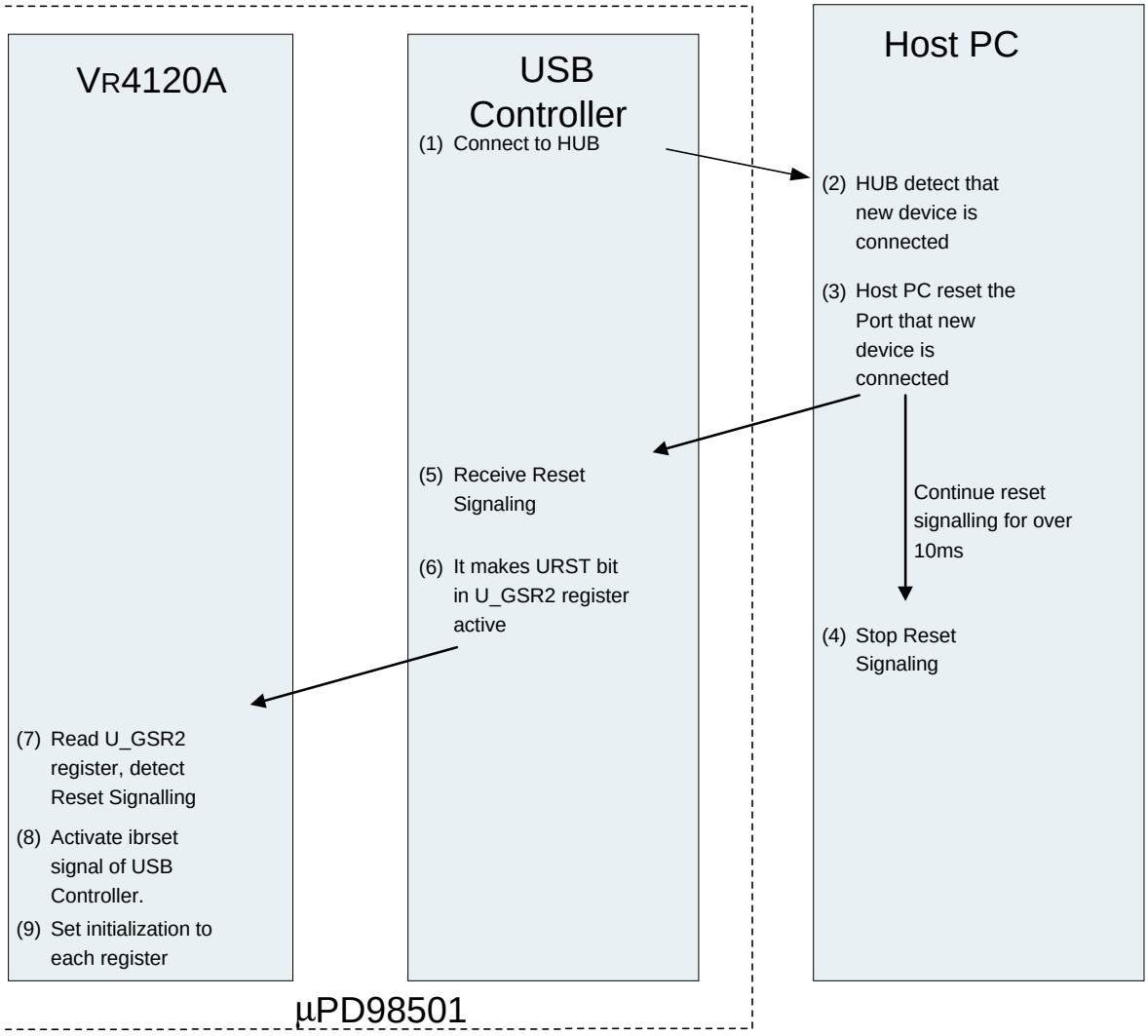



| Bit | Field | Description | R/W |
|---|---|---|---|
| 31-0 | Counter Register (STALL Count) | Free-run counter that indicates the number of times that STALL Handshake is performed during Handshake Phase on EndPoint6. | R |

## 6.3 USB Attachment Sequence

This section describes the sequence that is followed when USB Interface (μPD98501) is attached to a USB hub.

**Figure 6-2. USB Attachment Sequence**




(1) USB port of the μPD98501 is attached to the HUB.
(2) Notification that a new device (μPD98501) has been connected to the HUB is posted to the Host PC.
(3) The Host PC issues Reset Signaling to Reset the port to which the device has been connected.
(4) Once 10 ms have elapsed, the Host PC halts the issue of the Reset Signaling.
(5) Notification of the Reset Signaling performed by the Host PC is posted to USB Controller.
(6) To post notification of the Reset by the Host PC to the VR4120A RISC Processor, the URST bit of the U_GSR2 register is made active.
(7) The VR4120A RISC Processor reads the USB Controller U_GSR2 register and, upon finding that the URST bit is active, determines that Reset Signaling has been issued.
(8) To initialize USB Controller, the VR4120A RISC Processor activate reset signal of USB Controller.
(9) Upon the completion of the reset, USB Controller performs initialization by writing a value into each of USB Controller's internal registers.

## 6.4 Initialization

After USB Controller has been reset, the VR4120A RISC Processor must set several USB Controller registers. The setting that must be performed by the VR4120A RISC Processor is listed below.

(1) A desired mode is written into the USB General Mode Register.

(2) The receive pool area is secured in system memory, and the information it contains is set in the following registers:

| | |
|---|---|
| USB Rx Pool0 Information Register: | (Address: 50H) |
| USB Rx Pool0 Address Register: | (Address: 54H) |
| USB Rx Pool1 Information Register: | (Address: 58H) |
| USB Rx Pool1 Address Register: | (Address: 5CH) |
| USB Rx Pool2 Information Register: | (Address: 60H) |
| USB Rx Pool2 Address Register: | (Address: 64H) |

(3) The send/receive mailbox area is secured in system memory, and the information it contains is set in the following registers:

| | |
|---|---|
| USB Tx MailBox Start Address Register | (Address: 70H) |
| USB Tx MailBox Bottom Address Register | (Address: 74H) |
| USB Tx MailBox Read Address Register | (Address: 78H) |
| USB Tx MailBox Write Address Register | (Address: 7CH) |
| USB Rx MailBox Start Address Register | (Address: 80H) |
| USB Rx MailBox Bottom Address Register | (Address: 84H) |
| USB Rx MailBox Read Address Register | (Address: 88H) |
| USB Rx MailBox Write Address Register | (Address: 8CH) |

(4) A value is written into the MAXP field of the USB EP0 Control Register, the EP1-2 Control Register, the EP3-4 Control Register, and the EP5-6 Control Register. Then, each EndPoint is enabled.

The settings made up to this point enable the start of data sending/receiving, as well as the ability to respond to Device Configuration from the Host PC.

(5) Device Configuration is started via EndPoint0. Therefore a response must be returned to EndPoint0. At this stage, the USB Function Address is allocated. The VR4120A RISC Processor must write that value into the Function Address field in USB General Mode Register (Address: 00H).

The initialization procedure is completed.

In addition to the above, it may be necessary to write a value into the USB Interrupt Mask Register (Address: 14H or 1CH) and enable the interrupt.

### 6.4.1 Receive pool setting

For details of the receive pool setting, see Section 6.6.3.

### 6.4.2 Send/receive mailbox setting

After USB Controller sends a data segment, it indicates the status by writing a send indication into system memory. The area into which an indication is written is called a "mailbox," and exists within the system memory. As part of initialization, the VR4120A RISC Processor must set the mailboxes. USB Controller uses the mailboxes as a buffer having a ring configuration in system memory. This buffer is set using four registers for each of sending and receiving.

Registers for setting a send mailbox

- TMSA (USB Tx MailBox Start Address Register: 70H)
- TMBA (USB Tx MailBox Bottom Address Register: 74H)
- TMRA (USB Tx MailBox Read Address Register: 78H)
- TMWA (USB Tx MailBox Write Address Register: 7CH)

Registers for setting a receive mailbox

- RMSA (USB Rx MailBox Start Address Register: 80H)
- RMBA (USB Rx MailBox Bottom Address Register: 84H)
- RMRA (USB Rx MailBox Read Address Register: 88H)
- RMWA (USB Rx MailBox Write Address Register: 8CH)

**Remarks 1.** At initialization, TMRA must be set to the same value as TMSA. Similarly, RMRA must be set to the same value as RMSA.

**2.** When VR4120A RISC Processor writes the value to TMSA firstly after reset, USB Controller automatically copies the value to TMWA internally. Similarly, when VR4120A RISC Processor writes the value to RMSA firstly after reset, USB Controller automatically copies the value to RMWA internally.

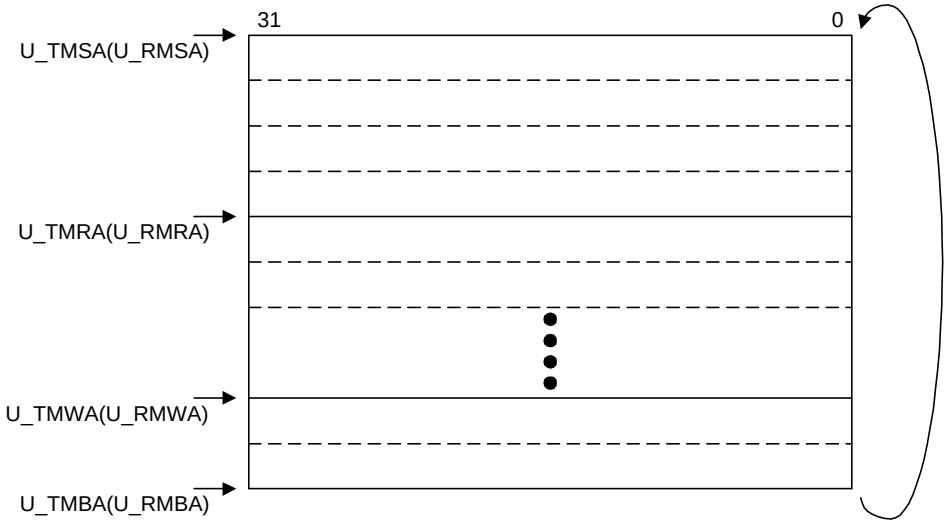
**3.** Do not set the same values for TMSA and TMBA.

**4.** Among those registers used to set the mailboxes, the VR4120A RISC Processor updates only TMRA and RMRA. The other registers are written to only as part of initialization. They are not to be written to at any other time.

**5.** Each receive indication has a two-word configuration. Therefore, the size of the receive mailbox must be an integer multiple of two words.

The configuration of the mailboxes in system memory is as shown in the following figure 6-3.

**Figure 6-3. Mailbox Configuration**




When USB Controller writes an indication, the write pointer (TMWA or RMWA) is incremented. Every time that USB Controller writes an indication, it also sets the send/receive end bit of the corresponding EndPoint and, provided it is not masked, issues an interrupt.

When USB Controller revises the value of the write pointer (TMWA or RMWA), the write pointer is forced to jump to the start address (TMSA or RMSA) when it reaches the bottom address (TMBA or RMBA). USB Controller uses the read pointer (TMRA or RMRA) to prevent the overwriting of those indications that the VR4120A RISC Processor has not yet read out. The read pointer (TMRA or RMRA) is managed and updated by the VR4120A RISC Processor. Each time the VR4120A RISC Processor reads an indication from a mailbox, it writes the address subsequent to the most recently read indication into the read pointer register (TMRA or RMRA). The read pointer register (TMRA or RMRA) is updated by the VR4120A RISC Processor, but only read by USB Controller.
When both the write pointer (TMWA or RMWA) and read pointer (TMRA and RMRA) point to the same address, USB Controller sets the TMF bit (send mailbox full) or RMF bit (receive mailbox full) of the USB General Status Register1 to indicate the mailbox full state and, provided it is not masked, issues an interrupt.

In the mailbox full status, USB Controller will not issue the next indication. The VR4120A RISC Processor must read an indication from the full mailbox and update the read pointer (TMRA or RMRA).

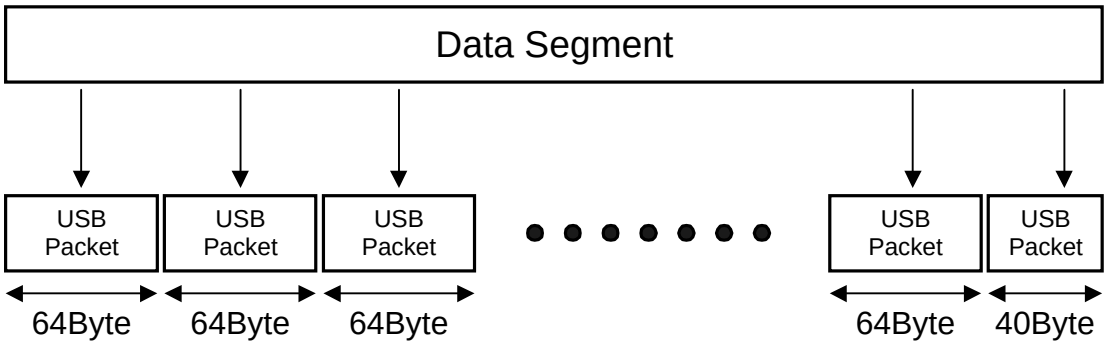
## 6.5 Data Send Function

This section explains USB Controller's data send function.

### 6.5.1 Overview of send processing

USB Controller takes the data segments in system memory, divides them into USB packets, then sends them to the Host PC. The VR4120A RISC Processor sets the size of USB packet in the MAXP field of the EP0 Control Register, the EP1-2 Control Register, the EP3-4 Control Register, and the EP5-6 Control Register (in the example shown below, a value of 64 bytes has been set).

**Figure 6-4. Division of Data into USB Packets**




When the data segments are divided by a value other than that set in the MAXP field, the last item of the divided data will be smaller than the value set in the MAXP field (40 bytes in the example shown above). As a result, the Host PC can identify the boundary between data segments. If a data segment is divided by the value set in the MAXP field, a zero-length USB packet will be sent after the last item of the divided data. (Only in send SZLP Mode. In send NZLP Mode, a zero-length USB packet is not sent. For an explanation of the send modes, see Section 6.5.3.)

After all the data segments have been transferred, USB Controller writes the "send indication" which have send status information into the mailbox in system memory.

For an explanation of the send indication, see Section 6.5.6.

With USB Controller, the sending of two data segments can be scheduled for a given EndPoint. Each scheduled send is performed upon the issue of a send command. The current send status can be determined by reading the contents of the USB Tx EndPoint Status Register (Address: 48H).

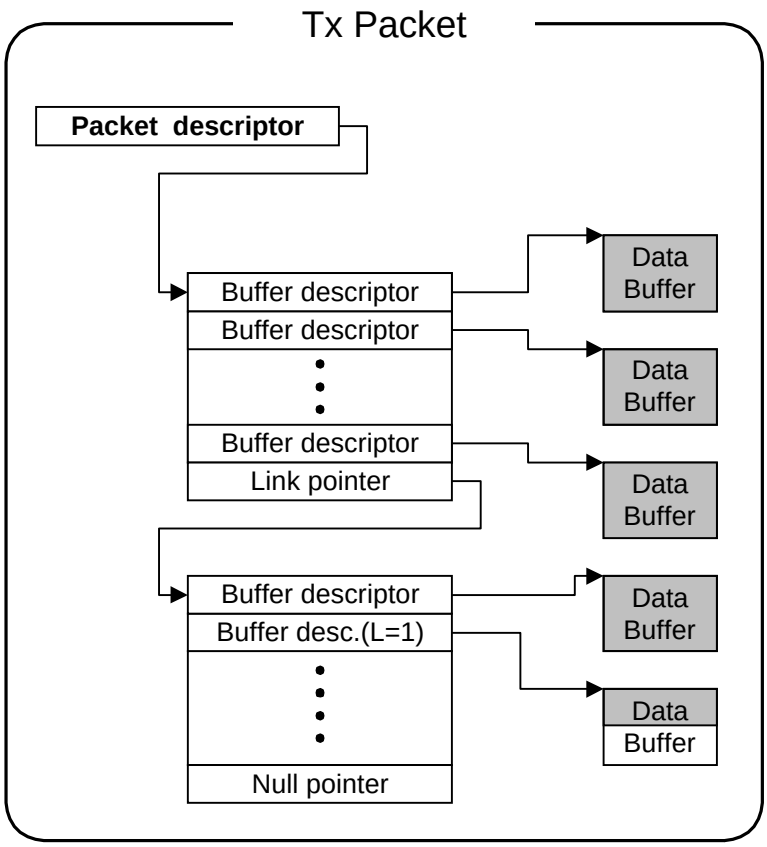
For an explanation of how to issue the send command, see Section 6.5.4.

### 6.5.2 Send buffer configuration

Upon sending data to the Host PC, the VR4120A RISC Processor creates a send buffer in system memory, then informs USB Controller.

The configuration of the send buffer is as shown below.

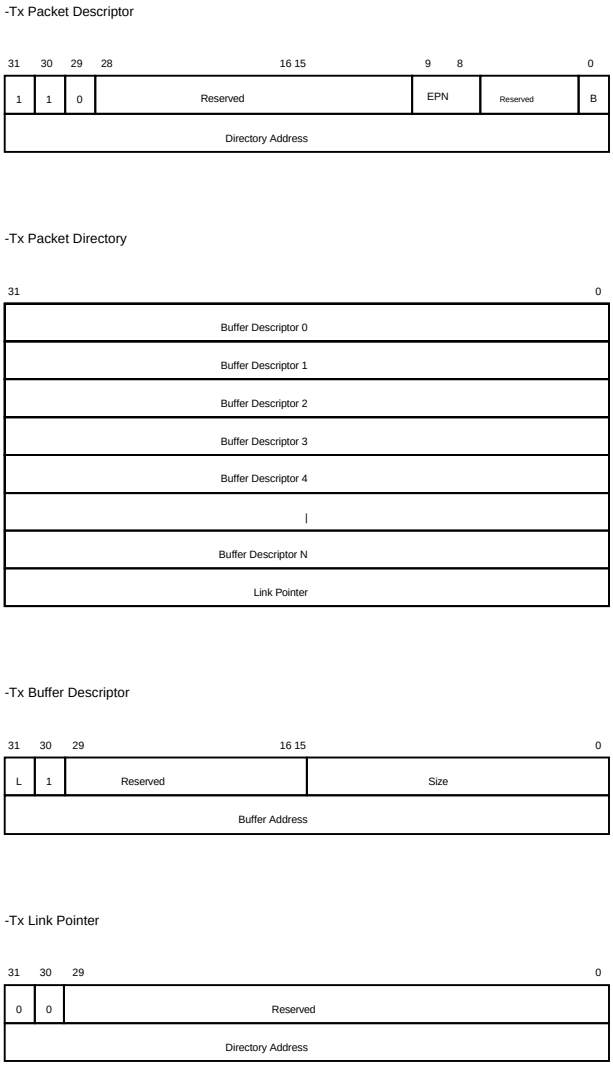
**Figure 6-5. Send Buffer Configuration**




A send packet is configured by breaking up multiple data buffers in system memory. These data buffers are bundled together in the "packet directory." The "ADDRESS" field of the Packet descriptor contains the start address of the packet directory.

The formats of the Packet descriptor, Packet directory, Buffer descriptor, and Link Pointer in the send buffer of the USB Controller block are as shown below.

**Figure 6-6. Configuration of Send Packet Descriptors**




**Tx Packet Descriptor:** Maintains the data in the Tx Packet.
Bit 31 to bit 29 are set to 110.
EPN indicates the number of the EndPoint that is to send a packet.
Bit0 indicates whether it is possible to write the next command..
The "Directory Address" field indicates the start address of the send packet directory.

**Tx Packet Directory:** This is the send packet directory. It is configured by bundling together the buffer descriptor and the link pointer. A single send Buffer Directory can accommodate up to 255 buffer descriptors.

**Tx Buffer Descriptor:** This is the send buffer descriptor. It maintains the data in the receive buffer.
When Bit31 (Last bit) is active, the Buffer Descriptor indicates the last buffer in a packet.
Bit30 is used to discriminate between the Buffer Descriptor and Link Pointer. When set to 1, this bit indicates the Buffer Descriptor.
The "Size" field indicates the buffer size. As the buffer size, a value between 1 and 64K bytes can be set. The "Buffer Address" field indicates the head address of the buffer.

**Tx Link Pointer:** This is the link pointer. It points to the next packet directory.
Bit31 is usually set to 0.

Bit30 is used to discriminate between the Buffer Descriptor and Link Pointer. When set to 0, this bit indicates the Link Pointer.
The "Directory Address" field indicates the head address of the next Buffer Directory.

### 6.5.3 Data send modes

USB Controller supports two send modes. These modes differ only in whether a zero-length USB packet is sent after the data segment. In all other aspects they are identical. The send mode is switched using the TM bit (Bit 19) of the USB EP1 EndPoint Control Register (Address: 24H) and USB EP3 EndPoint Control Register (Address: 2CH).

**Cautions 1. The setting made with the TM bit applies only to EndPoint1 and EndPoint3.**
**2. When EndPoint0 and EndPoint5 are used to send data, NZLP mode is usually used.**

**(1) SZLP (Send Zero Length Packet) mode**

In this mode, when the data segment is divided by the value set in the MAXP field, a zero-length packet is sent after the completion of data segment sending.

When the data segment has not been divided by the value set in the MAXP field, a USB Short Packet is sent.

**(2) NZLP (Non Zero Length Packet) mode**

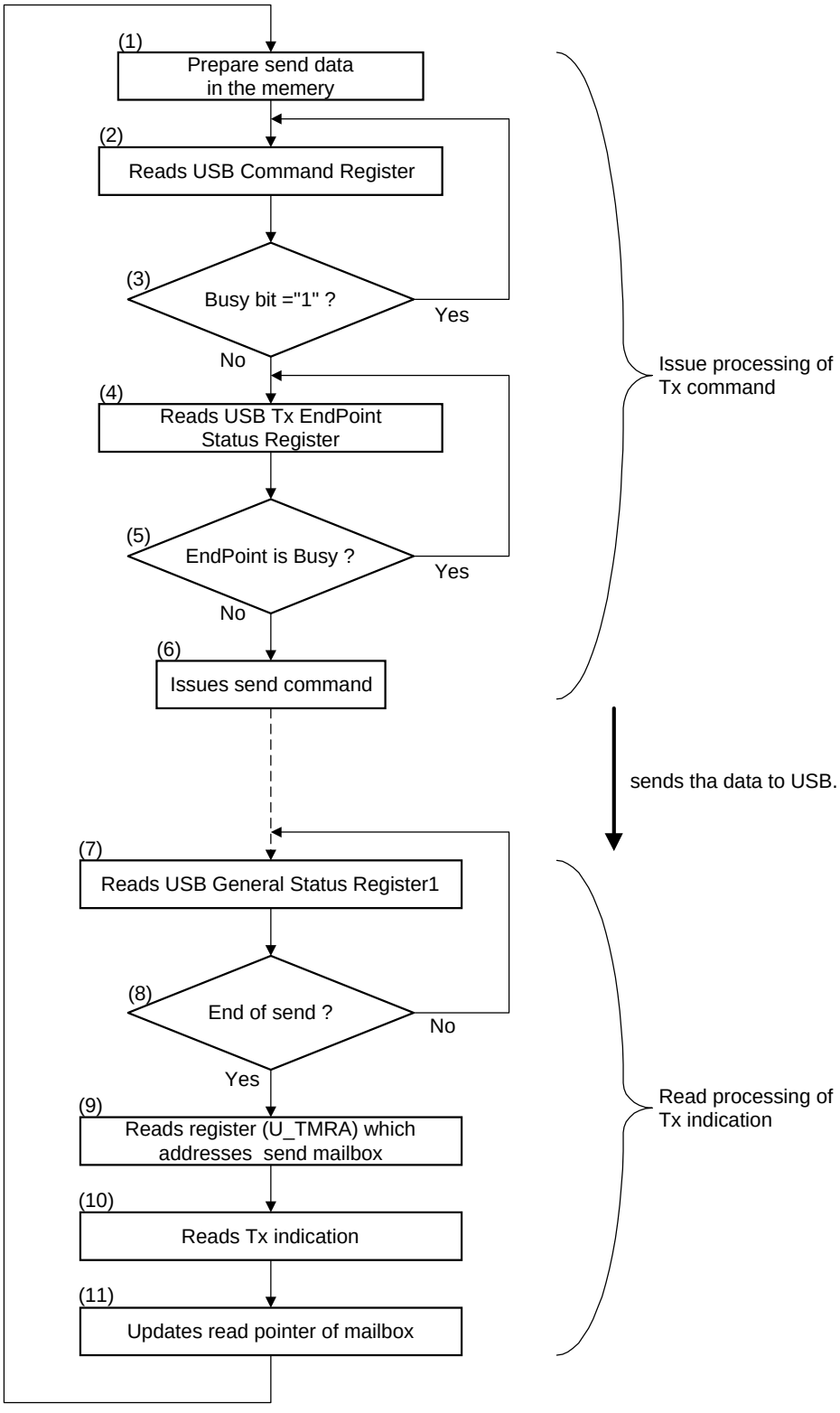
In this mode, even when the data segment is divided by the value set in the MAXP field, a zero-length packet is not sent after the completion of data segment sending.

When the data segment has not been divided by the value set in the MAXP field, a USB Short Packet is sent in the same way as in SZLP Mode.

### 6.5.4 VR4120A RISC processor processing at data sending

This section explains the processing performed by the VR4120A RISC Processor when sending data.

**Figure 6-7. VR4120A RISC Processor Processing at Data Sending**



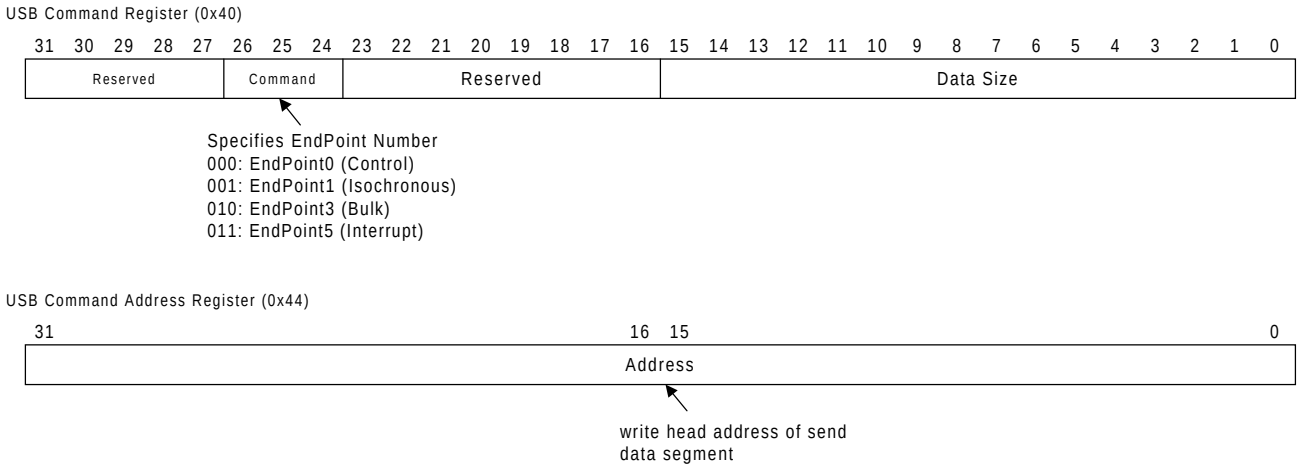
(1) First, the VR4120A RISC Processor prepares the send data in system memory. The data must be of a format that corresponds to the configuration of the send buffer.
(2) The VR4120A RISC Processor reads the USB Command Register.
(3) The VR4120A RISC Processor checks whether the Busy bit of the USB Command Register has been set. If the Busy bit is set, it indicates that USB Controller is still executing the previous command. Thus the VR4120A RISC Processor can not issue a new command.
(4) The VR4120A RISC Processor reads the USB Tx EndPoint Status Register.
(5) The VR4120A RISC Processor checks whether the EndPoint that is to perform send next is in the Busy status. If the EndPoint is Busy, the VR4120A RISC Processor repeats the processing from the reading of the USB Tx EndPoint Status Register.
(6) The VR4120A RISC Processor issues the Tx command.
(7) The VR4120A RISC Processor reads the USB General Status Registers 1.
(8) The VR4120A RISC Processor checks whether sending has terminated.
(9) The VR4120A RISC Processor reads the contents of the USB Tx Mailbox Read Address Register (Address: 78H).
(10) The VR4120A RISC Processor reads the send indication.
(11) The VR4120A RISC Processor updates the contents of the USB Tx MailBox Read Address Register.

By issuing a send command, the sending of a data segment can be scheduled.

A send command is issued by writing a value into the registers listed below. When writing, it is necessary to write first to the USB Command Address Register, then the USB Command Register.

If data size field of USB Command Register is “0”, USB Controller sends Zero-Length Packet.

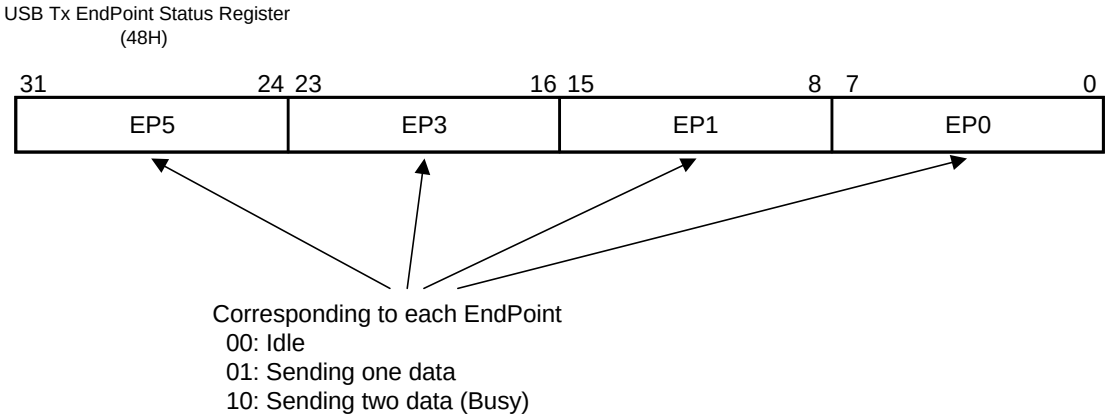
**Figure 6-8. Send Command Issue**




For a given EndPoint, it is possible to schedule the sending of up to two data items. Once two data items have been sent, even if the VR4120A RISC Processor writes a send command into the USB Command Register to send a third data item, that command is ignored.

The number of data items that are scheduled to be sent can be determined by reading the USB Tx EndPoint Status Register.

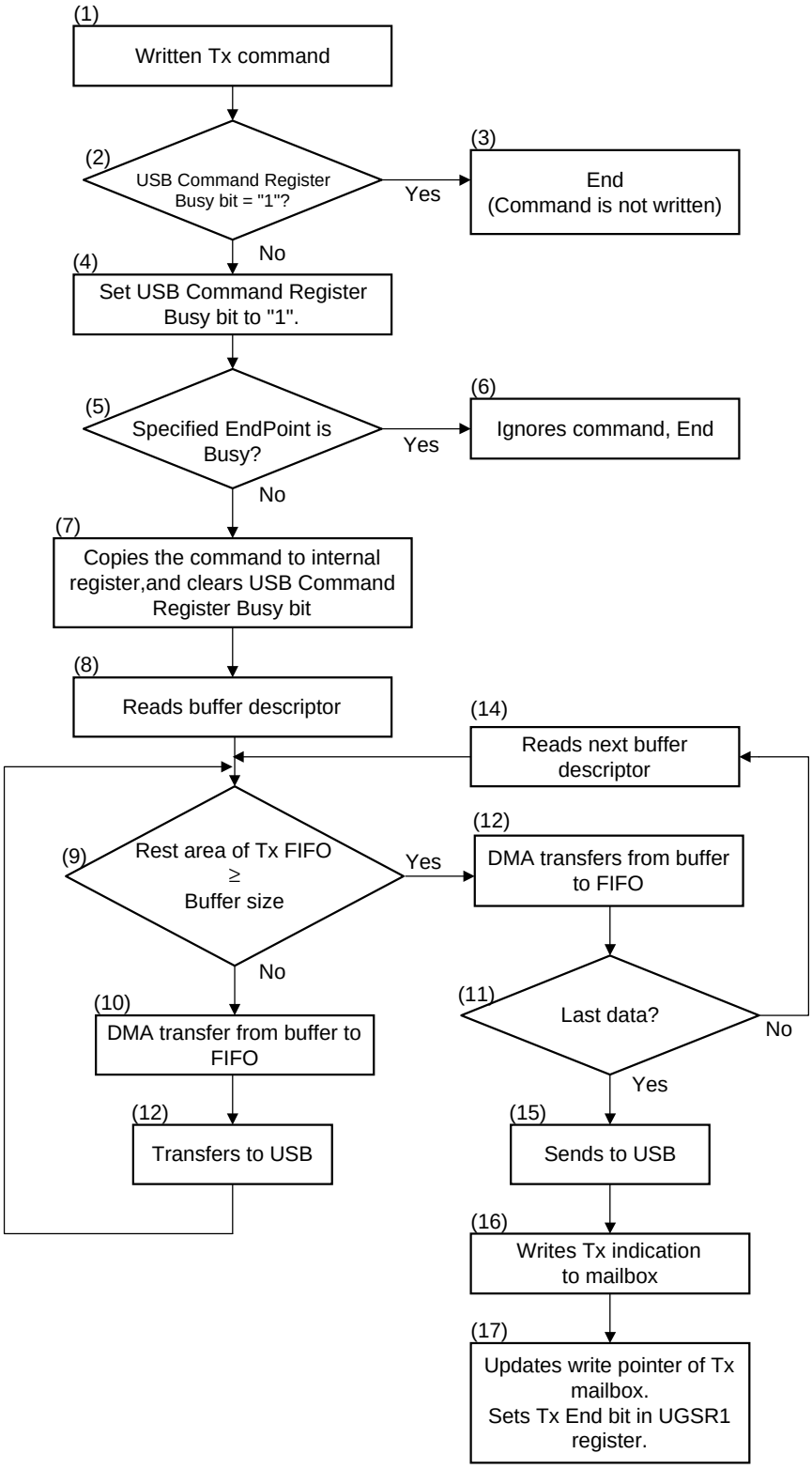
**Figure 6-9. Send Status Register**



### 6.5.5 USB controller processing at data sending

This section presents all of the processing performed by USB Controller at data sending. For an explanation of the operation of each block at data sending, see Section 6.9.

**Figure 6-10. USB Controller Send Operation Flow Chart**



Numbers (1) to (15) do not indicate the order in which USB Controller must perform processing. Instead, these numbers correspond to those in the following explanation.

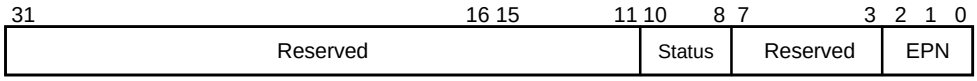
(1) USB Controller starts send processing upon receiving a send command from the VR4120A RISC Processor.
(2) At the instant that the command is written, USB Controller checks whether the Busy bit (Bit 31) of the USB Command Register is set.
(3) Set Busy Bit of USB Command Register to “1”.
(4) If the Busy bit (Bit 31) of the USB Command Register is set, the command written in (1) is ignored.
(5) USB Controller checks whether the EndPoint specified with the send command is currently in the Busy status (two items of data are scheduled to be sent).
(6) If the EndPoint specified with the send command is found to be in the Busy status, the command written in (1) is ignored.
(7) The command written into the USB Command Register and USB Command Address Register in (1) is copied to an internal register and the Busy bit of the USB Command Register is returned to 0.
(8) USB Controller reads the buffer descriptor that contains the value that is written into the USB Command Address Register at the same time as the send command.
(9) USB Controller compares the size of the area remaining in the send FIFO with the buffer size of the buffer descriptor read in the previous step.
(10) If step (8) reveals that the area remaining in the send FIFO is smaller, USB Controller DMA- transfers the data from the buffer until the send FIFO is full.
(11) Once the send FIFO is full, USB Controller transfers the data to the USB.
(12) If step (8) reveals that the area remaining in the send FIFO is larger, USB Controller DMA-transfers all the data in the buffer to the send FIFO.
(13) USB Controller checks whether the DMA-transferred data is the last data of the data segments to be sent to a Host.
(14) If the DMA-transferred data is not the last to be sent, it indicates that the buffer is empty. Therefore, USB Controller reads the next buffer descriptor.
(15) If the DMA-transferred data is the last to be sent, USB Controller writes a Tx indication in the mailbox.
(16) USB Controller updates the mailbox write pointer (USB Tx MailBox Write Address Register). It also sets the send completion bit of the USB General Status Register1 and, provided it is not masked, issues an interrupt to the VR4120A RISC Processor.

### 6.5.6 Tx indication

For every data segment that it sends, USB Controller writes a send indication into the send mailbox. After writing a send indication, USB Controller sets the send completion bit of USB General Status Register1 to 1 and, provided it is not masked, issues an interrupt to the VR4120A RISC Processor.

The format of the send indication is as shown below.

**Figure 6-11. Send Indication Format**




**Status:** Field that indicates the status upon data sending.

Bit10: When set to 0, indicates that an IBUS error has not occurred.
When set to 1, indicates that processing terminated abnormally due to the occurrence of an IBUS error.

Bit9: When set to 0, indicates that a buffer underrun did not occur during data sending.
When set to 1, indicates that a buffer underrun occurred.
This bit is set only when sending data to EndPoint1.

Bit8: When set to 0, data sending is performed in SZLP mode.
When set to 1, data sending is performed in NZLP mode.
For EndPoint0 and EndPoint5, this bit is usually set to 1.

**EPN:** Field that indicates the EndPoint number.

000: EndPoint0
010: EndPoint1
100: EndPoint3
110: EndPoint5

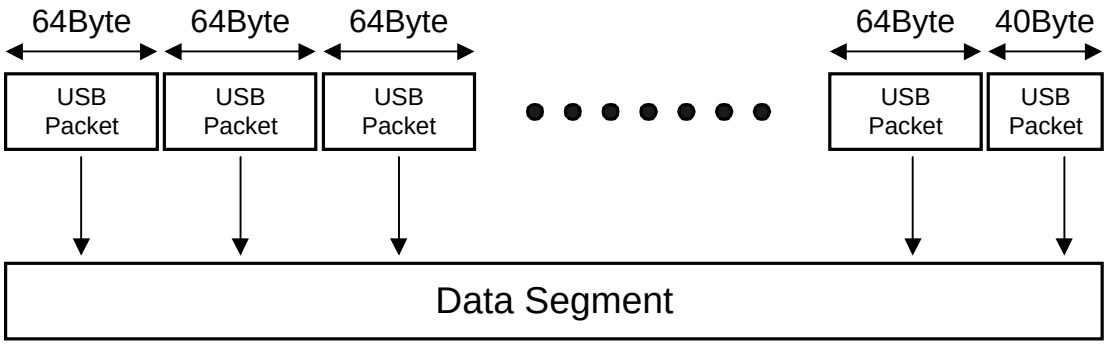
## 6.6 Data Receive Function

This section explains USB Controller's data receive function.

### 6.6.1 Overview of receive processing

USB Controller takes the USB packets that it receives from the USB, stores them into system memory in order, then creates a single data segment. The VR4120A RISC Processor sets the size of a single USB packet in the MAXP field of the EP0 Control Register, EP1-2 Control Register, EP3-4 Control Register, and EP5-6 Control Register. (The figure shown below is an example when the packet size is set to 64 bytes. )

**Figure 6-12. Division of Data into USB Packets**




When the data segments are divided by a value other than that set in the MAXP field, the last item of the divided data will be smaller than the value set in the MAXP field (40 bytes in the example shown above). As a result, USB Controller can identify the boundary between data segments. If a data segment is divided by the value set in the MAXP field, a zero-length USB packet will be sent from the Host PC to USB Controller after the last item of the divided data.

When sending data received from the USB to system memory, an area for the send data is required in system memory. This area is referred to as the receive buffer. It must be secured by the VR4120A RISC Processor. For an explanation of the receive buffer, see Section 6.6.2.

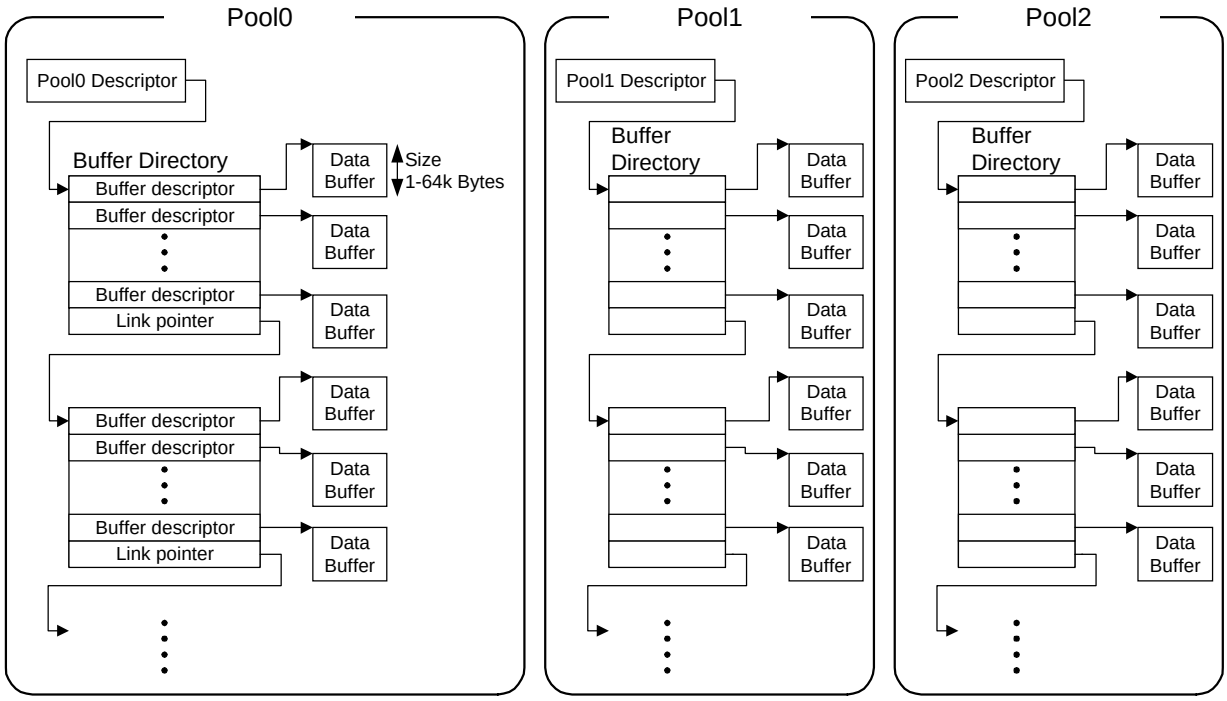
Upon the completion of data segment transfer, USB Controller writes the "Rx indication" into a mailbox in system memory. For an explanation of the Rx modes, see Section 6.6.7.

### 6.6.2 Receive buffer configuration

Data received from the USB is stored into a receive pool in system memory.

USB Controller uses three receive pools. The configuration of the receive pools is shown below.

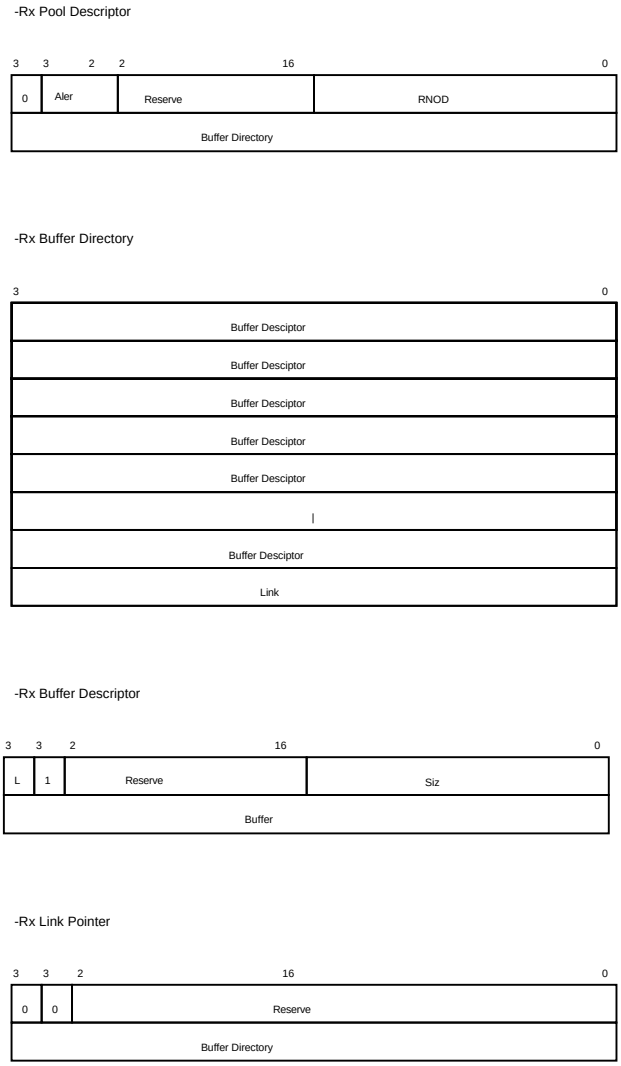
**Figure 6-13. Receive Pool Configuration**




The receive pool is configured of the buffer directory, that bundles together the buffers, as well as the buffers themselves. The receive pool is prepared in system memory by the VR4120A RISC Processor.

The formats of the Pool Descriptor, Buffer Directory, Buffer descriptor, and Link Pointer are each described below.

**Figure 6-14. Receive Descriptor Configuration**




**Rx Pool Descriptor:** Contains the receive pool data.

The "Buffer Directory Number" field indicates the remaining Buffer Directory number. When USB Controller uses a new Buffer Directory upon receiving data, USB Controller decrements this field by one.

After decrementing the value in RNOD field, if decremented value becomes equal to the value 4 times of Alert filed USB Controller sets Bit16 (Pool0) or Bit17 (Pool1) or Bit18 (Pool2) in USB General Status Register1 to one. If not masked, USB Controller generates interrupt to VR4120A RISC Processor.

Once decrementing causes this field to reach 0, USB Controller makes Bit19 (Pool0) or Bit20 (Pool1) or Bit21 (Pool2) of the General Status Register1 active provided it is not masked, issues an interrupt to the VR4120A RISC Processor. The "Buffer Directory Address" field indicates the start address of the receive Buffer Directory.

**Rx Buffer Directory:** A receive Buffer Directory. This is configured of a buffer descriptor and a link pointer. A single receive Buffer Directory can contain up to 255 buffer descriptors.

**Rx Buffer Descriptor:** Contains the receive buffer data.

When Bit31 (Last bit) is active, that Buffer Descriptor indicates the last buffer in the pool.

Bit30 is used to discriminate between the Buffer Descriptor and Link Pointer. When set to 1, this bit indicates the Buffer Descriptor.
The "Size" field indicates the buffer size. As the buffer size, a value between 1 and 64K bytes can be set. The "Buffer Address" field indicates the head address of the buffer.

**Rx Link Pointer:** This is the link pointer. It indicates the last Buffer Directory.
Bit31 is usually set to 0.
Bit30 is used to discriminate between the Buffer Descriptor and Link Pointer. When set to 0, this bit indicates the Link Pointer.
The "Buffer Directory Address" field indicates the head address of the next Buffer Directory.

### 6.6.3 Receive pool settings

USB Controller uses two receive pools.

**Pool0 For EndPoint0 (Control) and EndPoint6 (Interrupt)**
**Pool1 For EndPoint2 (Isochronous)**
**Pool2 For EndPoint4 (Bulk)**

The data in each of these two pools is written into the corresponding registers.

| | | |
|---|---|---|
| **Pool0** | **USB Rx Pool0 Information Register** | **(Address: 50H)** |
| | **USB Rx Pool0 Address Register** | **(Address: 54H)** |
| **Pool1** | **USB Rx Pool1 Information Register** | **(Address: 58H)** |
| | **USB Rx Pool1 Address Register** | **(Address: 5CH)** |
| **Pool2** | **USB Rx Pool2 Information Register** | **(Address: 60H)** |
| | **USB Rx Pool2 Address Register** | **(Address: 64H)** |

The VR4120A RISC Processor can know the current status of each pool by reading these registers.

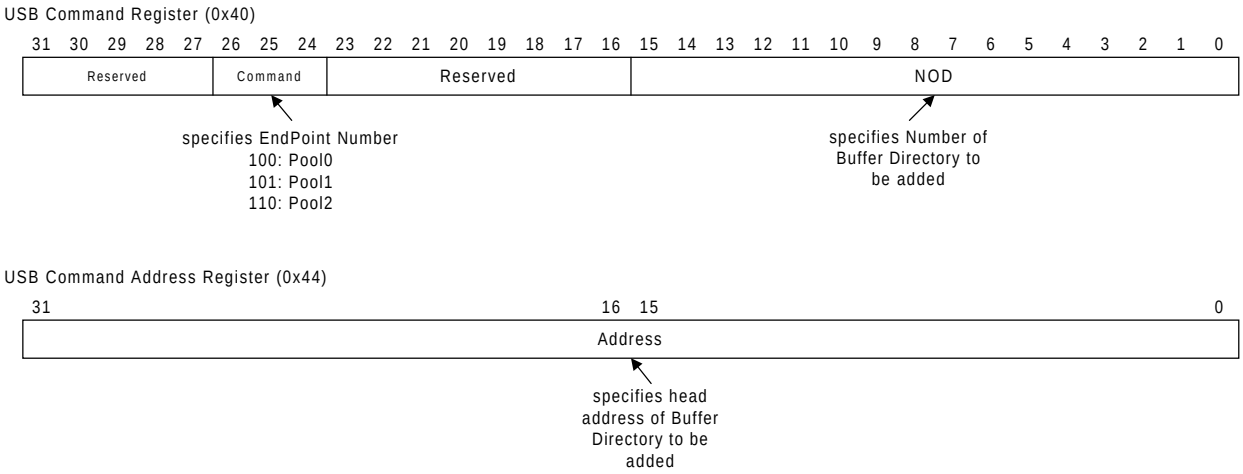
Upon initialization, after the VR4120A RISC Processor secures a receive pool area in the system memory area, it makes the settings for the pool by setting the head address of the pool, plus the number of Buffer Directories that the pool contains. The VR4120A RISC Processor can write values only into the Alert filed of four registers above. Other filed must be set using USB Command Register and USB Command Address Register.

The VR4120A RISC Processor adds Buffer Directories to each pool by using the USB Command Register (Address: 40H) and the USB Command Address Register (Address: 44H).

To add Buffer Directories to a receive pool, the VR4120A RISC Processor performs the following processing.

(1) The VR4120A RISC Processor secures the Buffer Directory to be added to the pool, and the buffer, in system memory. When multiple Buffer Directories are to be added, they are linked in advance.
(2) The VR4120A RISC Processor writes the head address of the Buffer Directory to be added into the link pointer to the last Buffer Directory in the list of dependent Buffer Directories in the pool.
(3) The VR4120A RISC Processor writes the head address of the Buffer Directory to be added into the USB Command Address Register (Address: 44H).
(4) The VR4120A RISC Processor writes the pool number and size of the Buffer Directory to be added into the USB Command Register (Address: 40H).

**Figure 6-15. Buffer Directory Addition Command**




The operation of USB Controller varies with whether any unused Buffer Directories remain in the corresponding pool when the Buffer Directory addition command is written into the USB Command Register.

(a) If any unused Buffer Directories remain in the pool (when the RNOD field in the Pool Information Register is set to other than 0), USB Controller adds the number in the NOD field of the command to the RNOD field of the Pool Information Register.
(b) When the pool is empty (when the RNOD field in the Pool Information Register is 0), USB Controller loads the value set in the NOD field of the command into the RNOD field of the Pool Information Register. Furthermore, it loads the value written in the USB Command Address Register into the Pool Address Register.

### 6.6.4 Data receive mode

USB Controller has different receive processing every EndPoint and receive mode.

The receive mode was determined by RM field (Bit20-19) in USB EP2 Control Register (Address 28H) and USB EP4 Control Register (Address 30H). There are four kinds of receive processing.

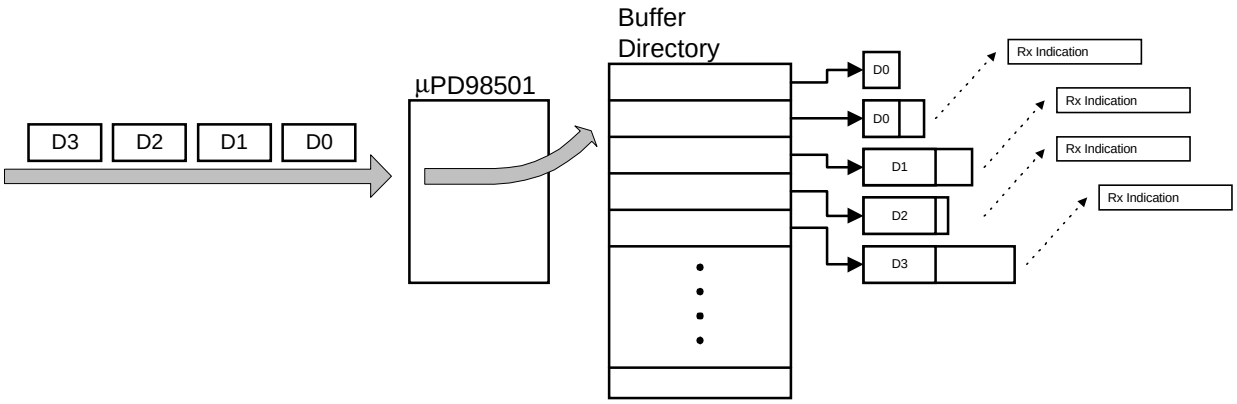
(1) EndPoint0, EndPoint6
(2) EndPoint2, EndPoint4 Normal Mode
(3) EndPoint2, EndPoint4 Assemble Mode
(4) EndPoint2, EndPoint4 Separate Mode

Each processing explains below.

**(1) Reception in EndPoint0, EndPoint6**

Same processing is executed without relations in receive mode in EndPoint0, EndPoint6 every time.

**Figure 6-16. Data Receiving in EndPoint0, EndPoint6**

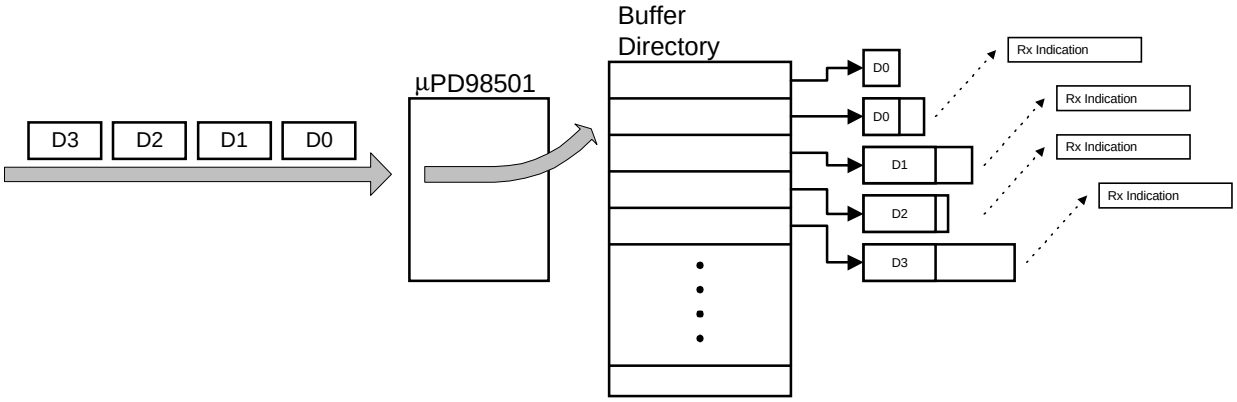



When USB Controller receives one USB packet, stores it in Data Buffer and write Rx indication to the Mailbox. USB Controller renews the size field and Last field in Buffer Descriptor every USB packet before writing Rx Indication.

**(2) EndPoint2, EndPoint4, normal mode**

The processing in EndPoint2, EndPoint4 receive Normal mode is explained below.

**Figure 6-17. EndPoint2, EndPoint4 Receive Normal Mode**

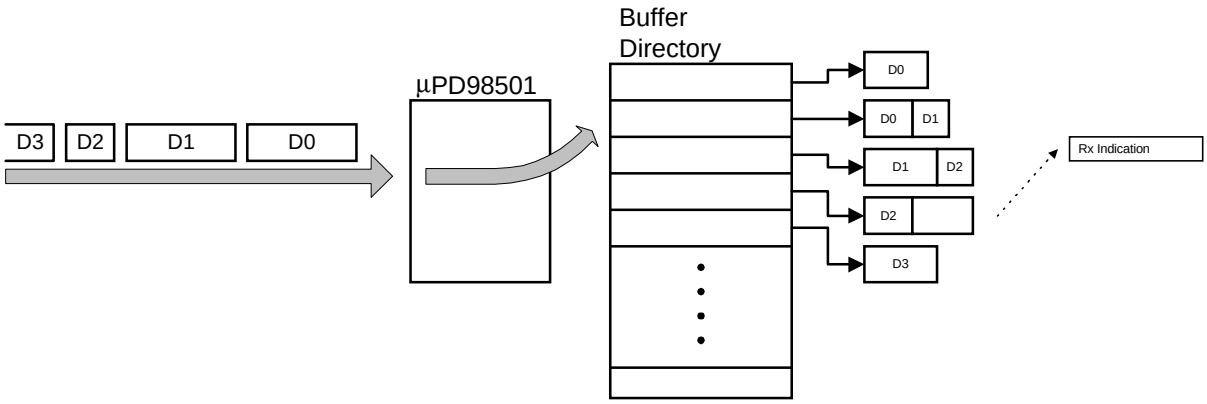



When USB Controller receives one USB packet, stores it in Data Buffer and write Rx indication to the Mailbox. USB Controller renews the size field and Last field in Buffer Descriptor every USB packet before writing Rx Indication.

**(3) EndPoint2, EndPoint4, assemble mode**

The processing in EndPoint2, EndPoint4 receive Assemble mode is explained below.

**Figure 6-18. EndPoint2, EndPoint4 Receive Assemble Mode**



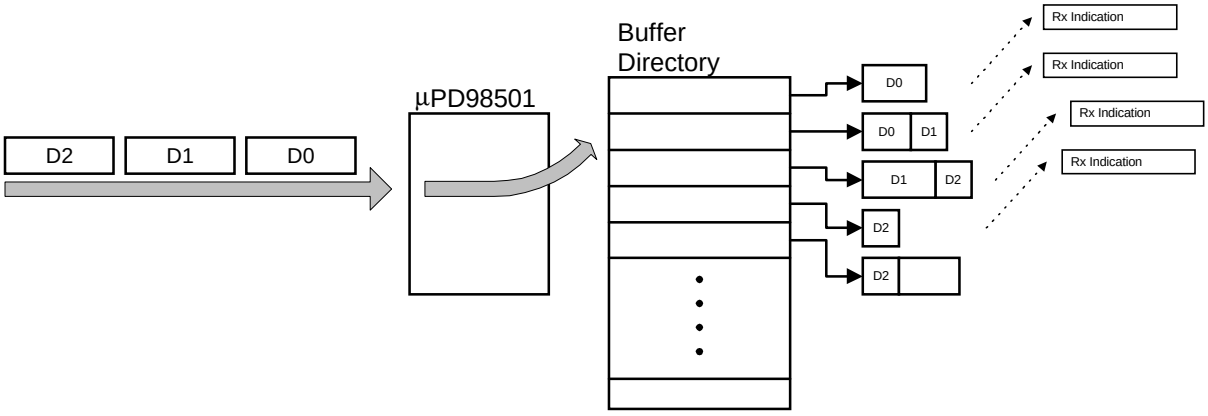

In this mode USB Controller issues Rx indication after receiving one data segment.

In other word, after USB Controller writes the Short packet or Zero-Length Packet received from USB to buffer in system memory, USB Controller renewing Size field, Last bit in last Buffer Descriptor and issues Rx indication.

**(4) EndPoint2, EndPoint4, separate mode**

The processing in EndPoint2, EndPoint4 receive separate mode is explained below.

**Figure 6-19. EndPoint2, EndPoint4 Receive Separate Mode**




In this mode, after USB Controller receives USB packet and stores the data in receive buffer, it issues Rx indication when buffer was full. If buffer was full in the middle of USB packets, in that time it issues indication. After that, processing of storing the USB packet to next buffer continues.

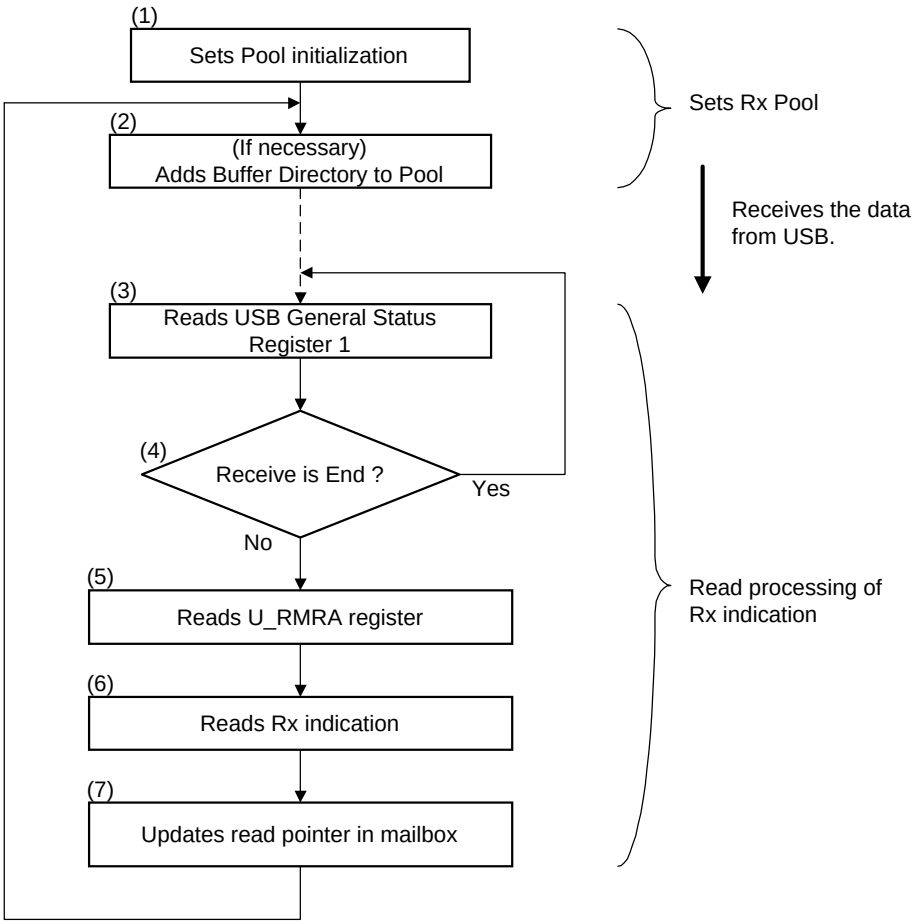
It does not execute to renew Size field, Last bit in Buffer Descriptor.

**Caution If you want to guarantee the behavior of Isochronous EndPoint as well as possible, Rx Normal Mode and software processing is recommended. Rx Assemble mode and Separate Mode is not fit to Isochronous EndPoint.**

### 6.6.5 VR4120A RISC processor receive processing

This section explains the processing that the VR4120A RISC Processor must perform when data is being sent.

**Figure 6-20. VR4120A RISC Processor Receive Processing**




Numbers (1) to (7) do not indicate the order in which the VR4120A RISC Processor must perform processing. Instead, these numbers correspond to those in the following explanation.

(1) First, as part of initialization, the VR4120A RISC Processor must make the necessary Pool settings.
(2) For receiving, the VR4120A RISC Processor must add Buffer Directories to the Pool, if necessary.
(3) The VR4120A RISC Processor reads the USB General Status Register 1.
(4) The VR4120A RISC Processor checks whether receiving has ended.
(5) If receiving has ended, the VR4120A RISC Processor reads the receive mailbox setting register (USB Rx MailBox Read Address Register Address: 88H) to determine the address of Mailbox VR4120A RISC Processor must read in the next time.
(6) Then, the VR4120A RISC Processor reads the Rx indication from the indicated mailbox.
(7) The VR4120A RISC Processor updates the receive mailbox read address (USB Rx MailBox Read Address Register).

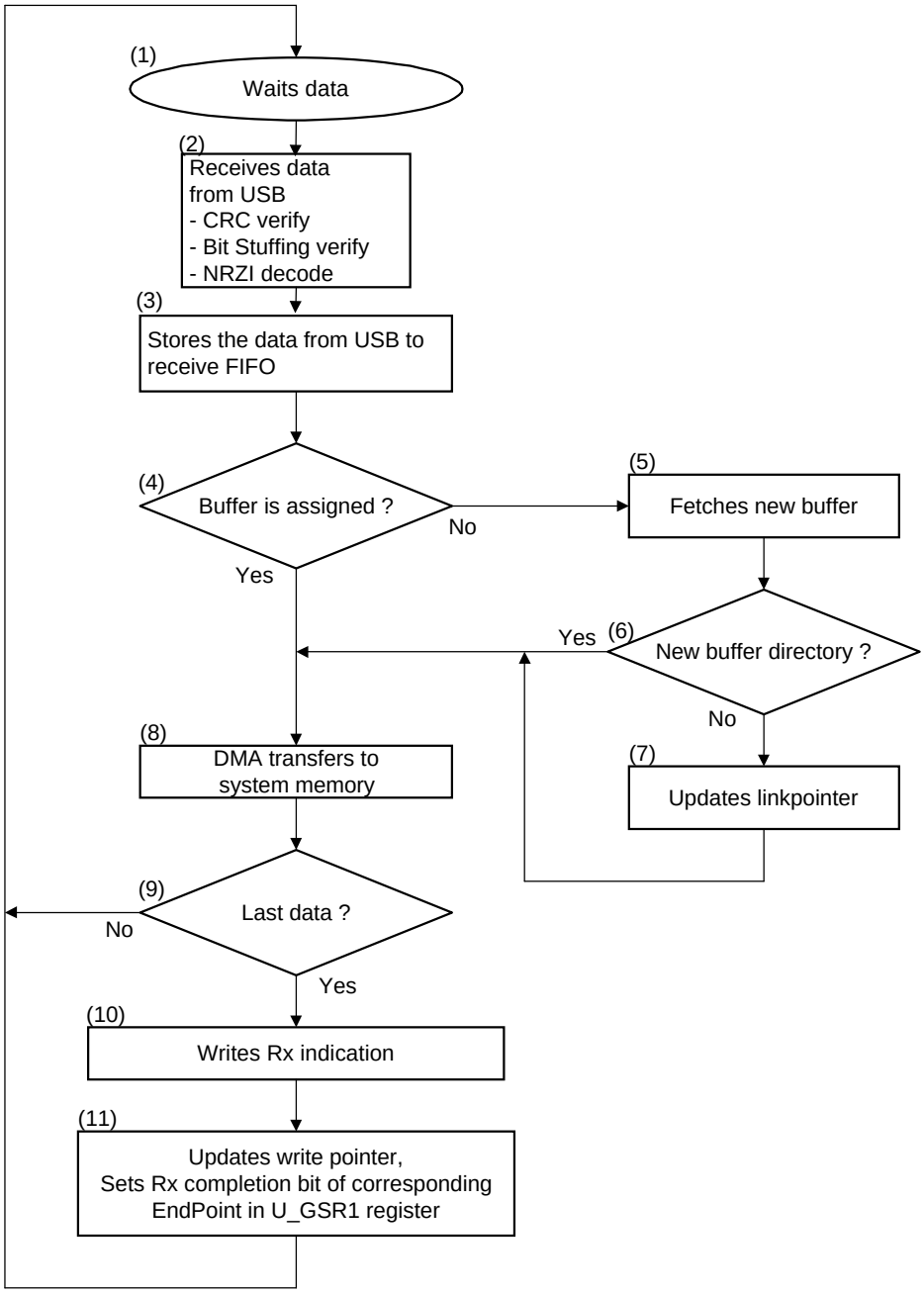
### 6.6.6 USB controller receive processing

This section presents all of the processing performed by USB RISC Processor at data receiving. For an explanation of the operation of each block at data receiving, see Section 6.9.

#### 6.6.6.1 Assemble mode

The following figure illustrates the receive operations performed by USB Controller in Assemble Mode.

**Figure 6-21. USB Controller Receive Operations (Assemble Mode)**



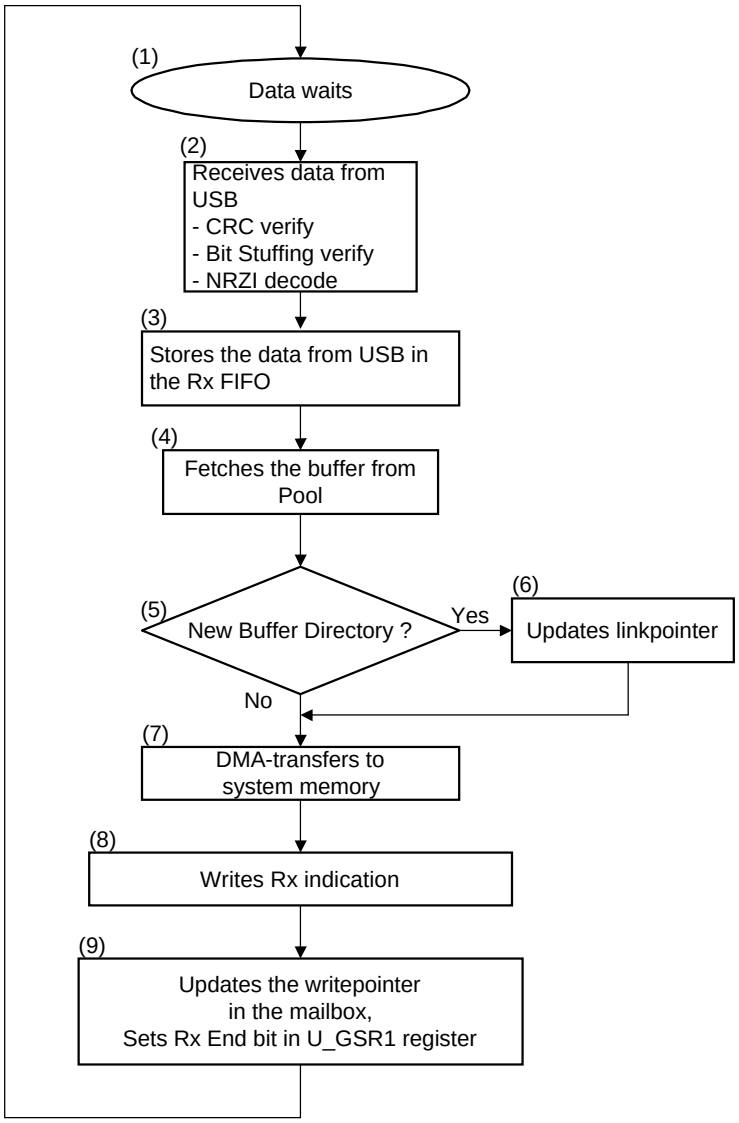
Numbers (1) to (11) do not indicate the order in which USB Controller must perform processing. Instead, these numbers correspond to those in the following explanation.

(1) USB Controller is in the status where it waits to receive data (USB Packets) from the USB.
(2) USB Controller receives data (USB Packets) from the USB. As it is receiving the data, USB Controller performs NRZI decoding, CRC check, and Bit Stuffing Error check.
(3) USB Controller stores the received data into the FIFO.
(4) USB Controller checks whether a buffer for storing receive data has been allocated in system memory.
(5) If USB Controller finds that a buffer has not been allocated, it prepares a new buffer.
(6) USB Controller checks whether the buffer prepared in (5) is in a new Buffer Directory.
(7) If USB Controller finds that the buffer is in a new Buffer Directory, USB Controller updates the link pointer for the Buffer Directory that has been used up to that point. It also updates the Buffer Directory address of the pool descriptor.
(8) USB Controller then DMA-transfers data from the FIFO to system memory.
(9) USB Controller checks whether the DMA-transferred data is the last data.
(10) If USB Controller finds that the transferred data is in fact the last data, renews the Size field and Last bit of Buffer Descriptor and writes the Rx indication into the prepared Mailbox.
(11) USB Controller updates the write pointer of the mailbox (Rx MailBox Write Address Register Address: 8CH). Also, it sets the receive completion bit of the USB General Status Register1 and, provided it is not masked, issues an interrupt to the VR4120A RISC Processor.

#### 6.6.6.2 Separate mode

The following figure illustrates the receive operations performed by USB Controller in Separate Mode.

**Figure 6-22. USB Controller Receive Operation Sequence (Separate Mode)**



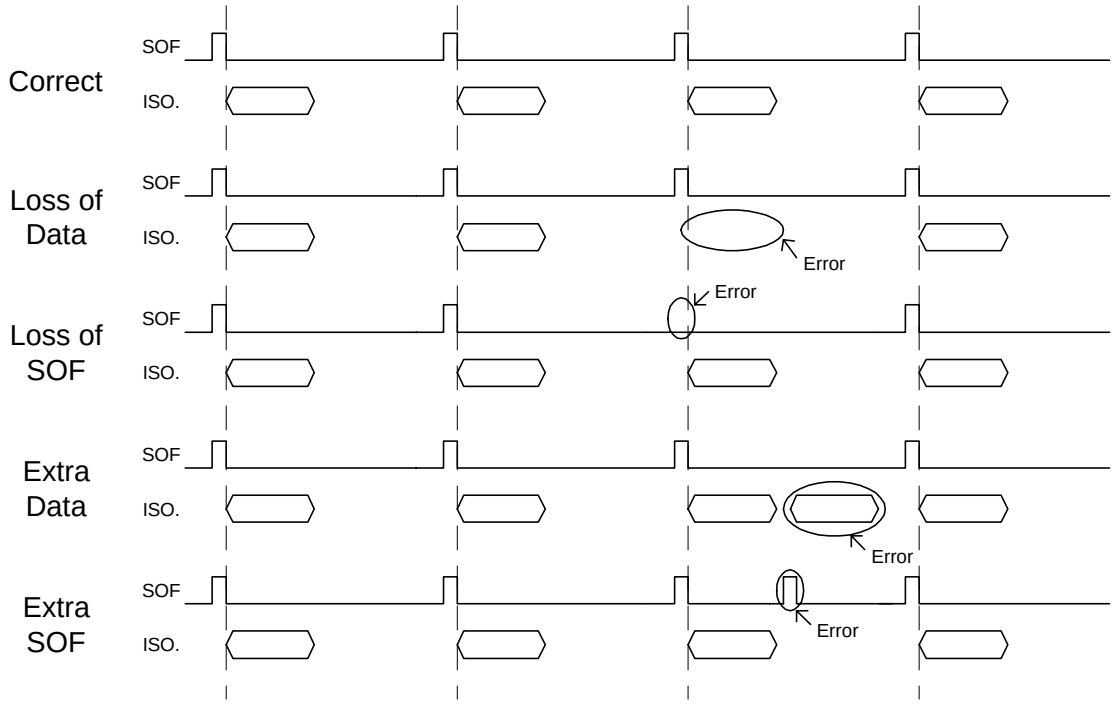

The difference from Assemble mode is the timing at which USB Controller writes the Rx Indication.

### 6.6.7 Detection of errors on USB

USB Controller has some functions which detect some errors on the USB.

Errors shown in figure below are related to Isochronous EndPoint and SOF packet.

**Figure 6-23. USB Timing Errors**




(1) If “Loss of Data” error has occurred, EP2ND bit (Bit 5) in USB General Status Register2 will be set. The other action of USB Controller for this error is explained in next section (Section 6.6.8).

(2) If “Loss of SOF” error has occurred, SL bit (Bit 0) in USB General Status Register2 will be set.
In this case, USB Controller only reflect the error to USB General Status Register.

(3) If “Extra Data” error has occurred, EP2ED bit (Bit 6) in USB General Status Register2 will be set.
In this case, USB Controller only reflect the error to USB General Status Register.

(4) If “Extra SOF” error has occurred, ES bit (Bit 1) in USB General Status Register2 will be set.
In this case, USB Controller only reflect the error to USB General Status Register.

USB Controller can detect the other Error listed below.

- Isochronous data oversize error: If received data packet size is over MaxPacketSize of EndPoint2, USB Controller will set EP2OS bit (Bit 7) in USB General Status Register2.

- Incorrect EndPoint Number: If received IN/OUT TOKEN packet includes the EndPoint Number which is not enabled by VR4120A RISC Processor or which is over 7, USB Controller will set IEA bit (Bit 19) in USB General Status Register2.

- No data in EndPoint1 Tx FIFO: If IN TOKEN packet for EndPoint2 comes when Tx FIFO for EndPoint2 is not ready, USB Controller will not send any data

to USB and will set EP1ND bit (2 Bit) in USB General Status Register2.

- Extra Token on EndPoint1: If IN TOKEN packet for EndPoint2 comes which between two SOFs, USB Controller will set EP1ET bit (Bit 3) in USB General Status Register2. In this case, USB Controller will send data only once.

- No Token on EndPoint1: If IN TOKEN packet for EndPoint2 does not come between two SOFs, USB Controller will set EP1NT bit (Bit4) in USB General Status Register2.

### 6.6.8 Rx data corruption on isochronous EndPoint

On Isochronous Rx EndPoint (EP2), one data packet comes per one frame.

If any Isochronous data packet doesn't come between two SOF packet, it is assumed that Isochronous data is corrupted.

In the case of corruption, action of USB Controller varies according to Rx Mode.

**(a) Rx normal mode**

USB Controller sets EP2ND (EndPoint2 No Data) bit (Bit 6) in USB General Status Register2.

USB Controller doesn't write any Rx Indications.

USB Controller doesn't write any data to Data Buffer on System Memory.

**(b) Rx assemble mode**

USB Controller sets EP2ND (EndPoint2 No Data) bit (Bit 6) in USB General Status Register2.

USB Controller writes dummy data to Data Buffer. (In fact, USB Controller only increment pointer which addresses Data Buffer by MaxPacketSize. No DMA transfer occurs.)

USB Controller writes Rx Indications which indicates that received data was corrupted on Isochronous EndPoint.

**(c) Rx separate mode**

USB Controller sets EP2ND (EndPoint2 No Data) bit (Bit 6) in USB General Status Register2.

USB Controller writes dummy data to Data Buffer (In fact, USB Controller only increment pointer which addresses Data Buffer by MaxPacketSieze. No DMA transfer occurs.)

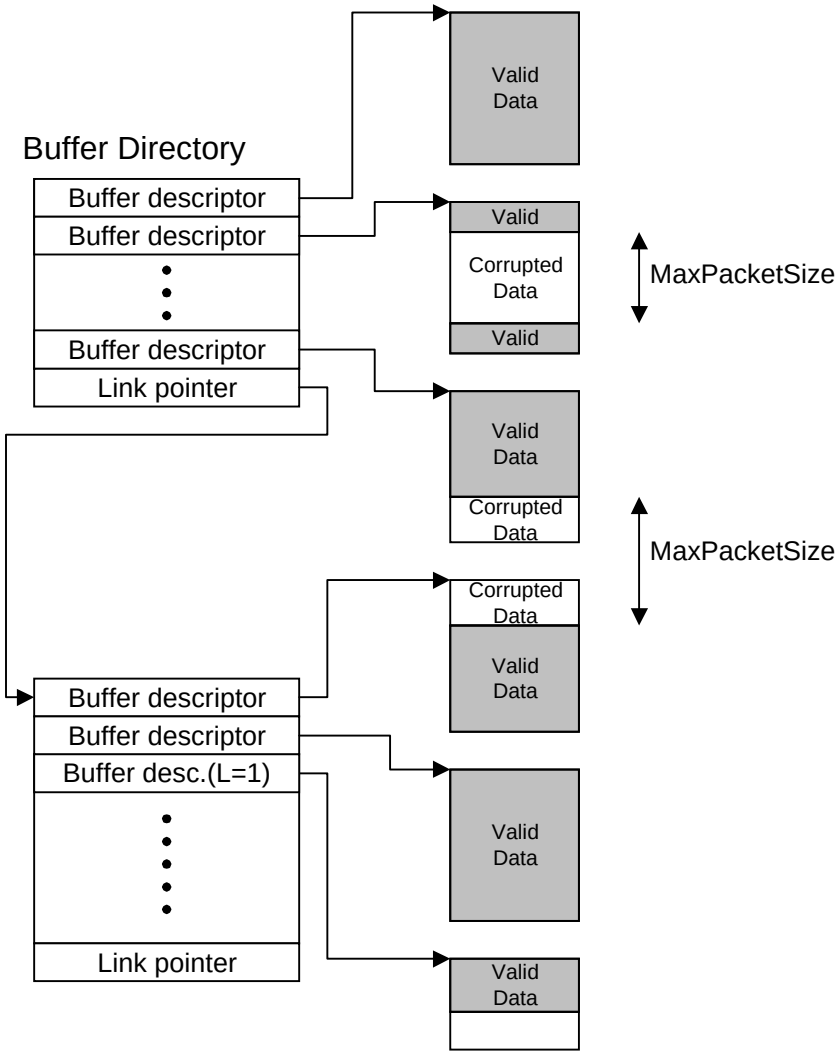
USB Controller writes Rx Indications which indicates that received data was corrupted on Isochronous EndPoint.

**Example**

When USB Controller performs receiving in Rx Assemble Mode or Rx Separate Mode, if data corruption occurs on Isochronous Rx EndPoint (EndPoint4), Data Buffers becomes as figure 6-24.

Shaded area in following figure is filled by valid data. If USB Controller detects that data is corrupted, next data must be stored in Data Buffer which keeps area for corrupted data. Corrupted data area is equal to MaxPacketSize of Isochronous Rx EndPoint (EndPoint2).

**Figure 6-24. Example of Buffers Including Corrupted Data**




### 6.6.9 Rx FIFO overrun

On Isochronous Rx EndPoint (EP2), if data supplement to FIFO is delayed by some problem, Rx FIFO Overrun will occur.

In the case of corruption, action of USB Controller varies according to Rx Mode.

**(a) Rx normal mode**

USB Controller sets EP2FO (EndPoint2 No Data) bit (Bit 9) in USB General Status Register2.

USB Controller writes the Rx Indications which indicates that EP2 FIFO Overrun has occurred.

USB Controller doesn't write any dummy data to Data Buffer on System Memory.

**(b) Rx assemble mode**

USB Controller sets EP2FO (EndPoint2 No Data) bit (Bit 9) in USB General Status Register2.

USB Controller writes dummy data to Data Buffer (In fact, USB Controller only increment pointer which addresses Data Buffer by MaxPacketSize. No DMA transfer occurs.) so that the sum of received data and dummy data becomes equal to MaxPacketSize.

After USB Controller receives “USB short packet”, USB Controller writes Rx Indications which indicates that EP2 FIFO Overrun has occurred.

**(c) Rx separate mode**

USB Controller sets EP2FO (EndPoint2 No Data) bit (Bit 9) in USB General Status Register2.

USB Controller writes dummy data to Data Buffer (In fact, USB Controller only increment pointer which addresses Data Buffer by MaxPacketSieze. No DMA transfer occurs.) so that the sum of received data and dummy data becomes equal to MaxPacketSize.

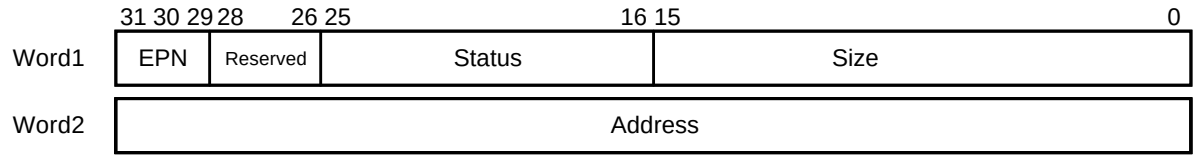
USB Controller writes Rx Indications which indicates that EP2 FIFO Overrun has occurred.

### 6.6.10 Rx indication

For every data segment that it receives, USB Controller writes a receive indication into the receive mailbox. After writing a receive indication, USB Controller sets the receive completion bit of the USB General Status Register to 1 and, provided it is not masked, issues an interrupt.

The format of the receive indication is as shown below.

**Figure 6-25. Receive Indication Format**




**Remark** Bit29 to Bit25 are reserved.

**EPN:** Field that indicates the EndPoint number.

001: EndPoint0
011: EndPoint2
101: EndPoint4
111: EndPoint6

**Status:** Field that indicates the status upon data receiving.

Bit25: When set to 0, indicates that an data corruption didn’t occur on EndPoint2.
When set to 1, indicates that an data corruption occurred on EndPoint2.

Bit24: When set to 0, indicates that an IBUS error has not occurred.
When set to 1, indicates that processing terminated abnormally due to the occurrence of an IBUS error.
When this bit is set to 1, the Address field has no meaning.

Bit23: When set to 0, indicates that the received data is other than a USB Setup packet.
When set to 1, indicates that the received data is a USB Setup packet.

This bit is set only when receiving the data from the Control EndPoint (EndPoint0). If data is received from any other EndPoint, this bit is not set.

Bit22: When set to 0, indicates that a buffer overrun did not occur.
When set to 1, indicates that a buffer overrun occurred.
This bit is set only when receiving the data from the EndPoint1.

Bit21: When set to 0, indicates that a Data toggle Error has not occurred.
When set to 1, indicates that a Data toggle Error has occurred.
When this bit is set to 1 when receiving data from the Isochronous EndPoint (EndPoint2), it indicates that part of the data stored in system memory may have been lost.
For any other EndPoint, even if this bit is set to 1, data is not lost.

Bit20: When set to 0, indicates that a CRC error has not occurred.
When set to 1, indicates that a CRC error has occurred.
When this bit is set to 1 when receiving data from the Isochronous EndPoint (EndPoint2), it indicates that the data stored in system memory includes a CRC error.
For any other EndPoint, even if a CRC error occurs, the packet will be resent such that the data stored into main memory will not include an error.

Bit19: When set to 0, indicates that a Bit Stuffing Error has not occurred.
When set to 1, indicates that a Bit Stuffing Error has occurred.
When this bit is set to 1 when receiving data from the Isochronous EndPoint (EndPoint2), it indicates that the data stored in system memory contains a Bit Stuffing Error.
For any other EndPoint, even if a Bit Stuffing Error occurs, the packet will be resent such that the data stored into main memory does not contain an error.

Bit18: When set to 0, indicates that the size of the received data is up to 65535 bytes.
When set to 1, indicates that the size of the received data is greater than 65535 bytes.

Bit17-16: When set to 00 or 01, indicates that data is received in Normal Mode.
When set to 10, indicates that data is received in Assemble Mode.
When set to 11, indicates that data is received in Separate Mode.
When using EndPoint0 and EndPoint6, this field should be set to 00.

**Size:** Indicates the size of the received data.
When the size of the received data exceeds 65535 (FFFFH) bytes, Bit18 is set to 1, and this field will contain 65535 (FFFFH).

**Address:** Indicates the head address of the buffer into which the received data is stored.

## 6.7 Power Management

USB Controller has a built in feature that allows it to use interrupts to inform the VR4120A RISC Processor of its having received Suspend or Resume signaling from the Host PC. When the VR4120A Processor receives a Suspend or Receive interrupt, it must perform the appropriate processing.

Also, for those instances when the port to which the μPD98501 is connected is in the Suspend status (the μPD98501 is in the Suspend status), USB Controller has a function for issuing Remote Wake Up signaling to switch the Suspend status to Resume.

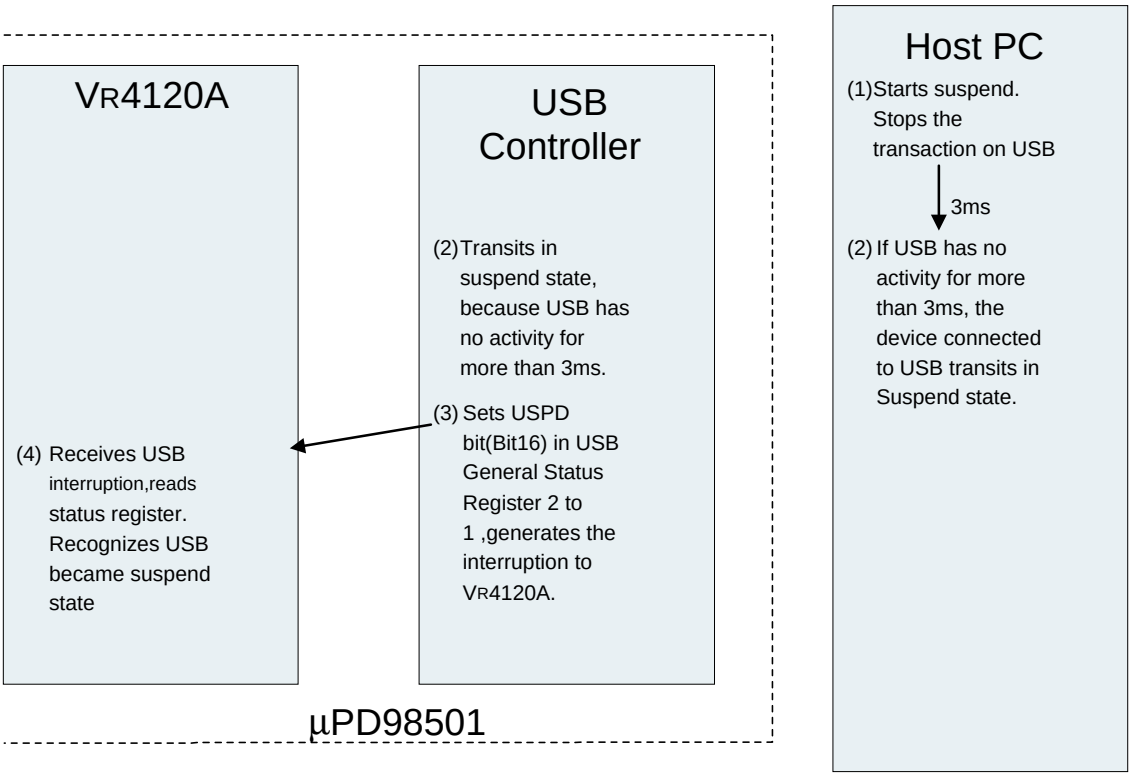
As a result, even if the μPD98501 is in the Suspend status, data that arrives from the line (ADSL) is not discarded but is instead can be passed to the Host PC.

The following sections explain each of the sequences.

### 6.7.1 Suspend

The Suspend sequence is as shown below.

**Figure 6-26. Suspend Sequence**




(1) The host places the USB in the Suspend status. Traffic stops flowing through the USB.

(2) After 3 ms there is no traffic through the USB. Therefore, all of the devices connected to the USB shift to the Suspend status. In the same way, USB Controller also enters the Suspend status. If DMA transfer is being performed, however, USB Controller does not enter the Suspend status until after the completion of DMA transfer.

(3) USB Controller makes the USPD bit (Bit16) of the USB General Status Register2 (Address: 18H) active, then issues an interrupt to the VR4120A RISC Processor if interrupt is not masked.

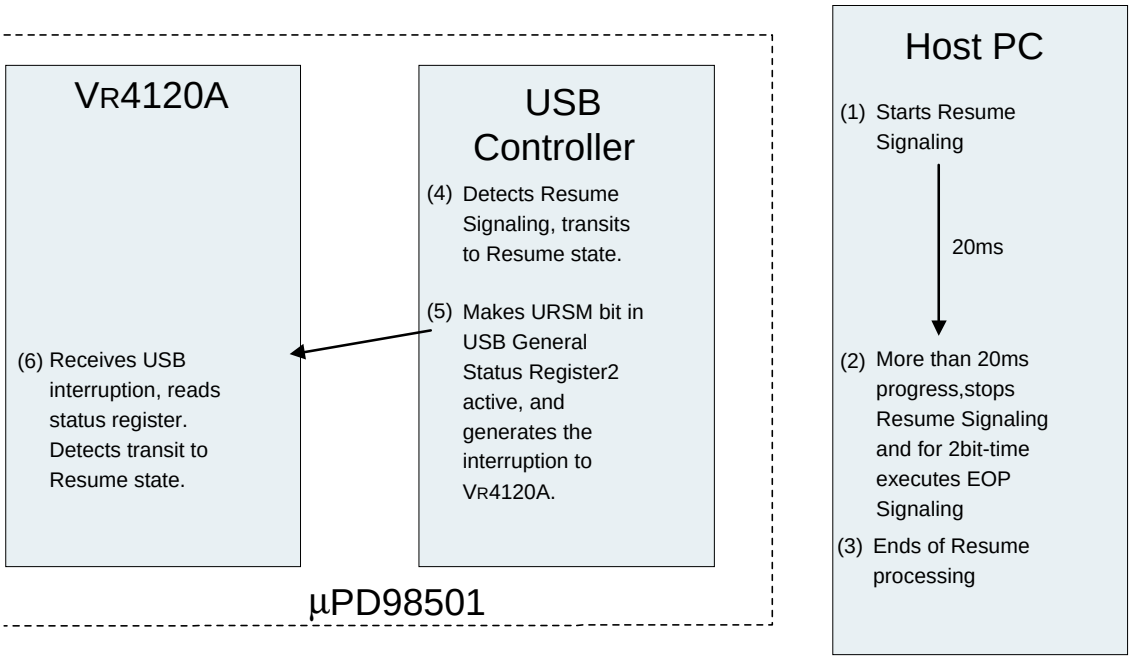
(4) The VR4120A RISC Processor receives the interrupt from USB Controller, reads the USB General Status Register2 and, as a result, determines that the USB is in the Suspend status.

The VR4120A RISC Processor is not permitted to write to other than USB Controller's USB General Mode Register and USB Interrupt Mask Register2 while USB Controller is in the Suspend status. Otherwise, after USB Controller enters the Resume status, its operation will be unpredictable.

### 6.7.2 Resume

The Resume sequence is shown below.

**Figure 6-27. Resume Sequence**



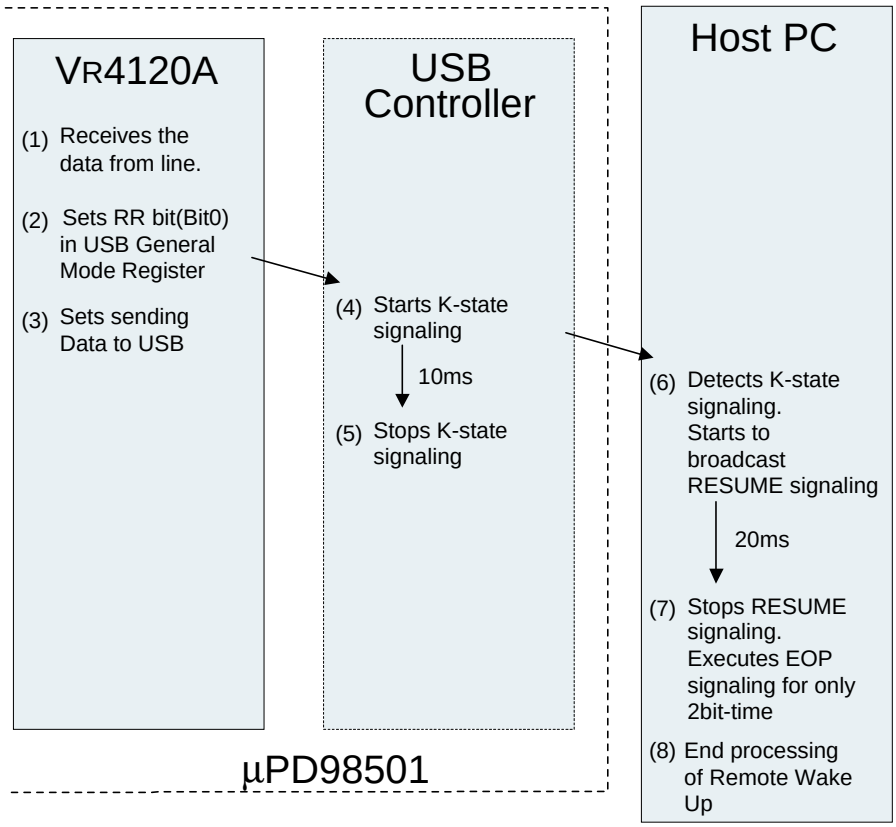

(1) The Host PC starts Resume Signaling. The Resume Signaling is passed to every device connected to the USB.
(2) After at least 20 ms have elapsed, the Host PC stops the Resume Signaling then performs EOP Signaling for a 2bit-time duration.
(3) This causes the Host PC to terminate its Resume processing.
(4) USB Controller receives the Resume Signaling.
(5) USB Controller makes the URSM bit (Bit18) of the USB General Status Register2 (Address: 18H) active, then issues an interrupt to the VR4120A RISC Processor. If clock supplement is stopped, VR4120A RISC Processor has to check “usbwakeup_p” signal instead of “URSM” bit.
(6) The VR4120A RISC Processor receives the interrupt from USB Controller and, as a result, determines that the USB has entered the Resume status.

After USB Controller enters the Resume status, it continues with the send/receive processing it was performing immediately before it entered the Suspend status.

### 6.7.3 Remote wake up

The Remote Wake Up sequence is shown below.

**Figure 6-28. Remote Wake Up Sequence**




(1) Here, it is assumed that the USB is in the Suspend status. Data received from another communication line.

(2) The VR4120A RISC Processor sets the RR bit (Bit 0) of the USB General Mode Register in order to switch the USB in the Suspend status to the Resume status.

(3) Once the RR bit of the USB General Mode Register has been set, USB Controller starts K-state Signaling for the USB.

(4) The VR4120A RISC Processor can continue to set send data for the USB. Specifically, the VR4120A RISC Processor prepares send data in system memory, then writes data into the USB Command Register (Address: 40H) and the USB Command Address Register (Address: 44H).

(5) USB Controller continues K-state Signaling for 5 ms, then terminates the signaling.

(6) The Host PC, upon receiving the K-state Signaling, broadcasts RESUME signaling. This RESUME signaling continues for a minimum of 20 ms.

(7) Once at least 20 ms have elapsed, the Host PC terminates RESUME Signaling, then issues EOP Signaling for a 2bit-time duration.

(8) As a result of this sequence, Remote Wake Up is terminated, and the transaction being performed by the USB is restarted.

## 6.8 Loopback Mode

USB Controller features a built-in loopback function for test purposes.

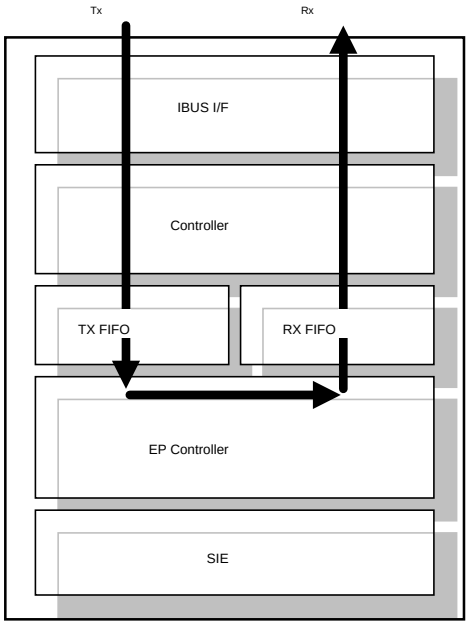
To enable the loopback function, set the LE bit (Bit 1) of the USB General Mode Register to 1.

Once the loopback function has been activated, USB Controller writes the data read from system memory into the send FIFO. The data is returned by the EndPoint Controller. The returned data is written into the receive FIFO, after which it is returned to system memory.

Sending and receiving should be performed using the normal settings. The send and receive indications are issued as normal.

The internal data flow is as shown below.

**Figure 6-29. Data Flow in Loopback Mode**




As shown in the figure, in loopback mode data reception from the USB and data sending to the USB are not performed. All data is returned by the EndPoint Controller.

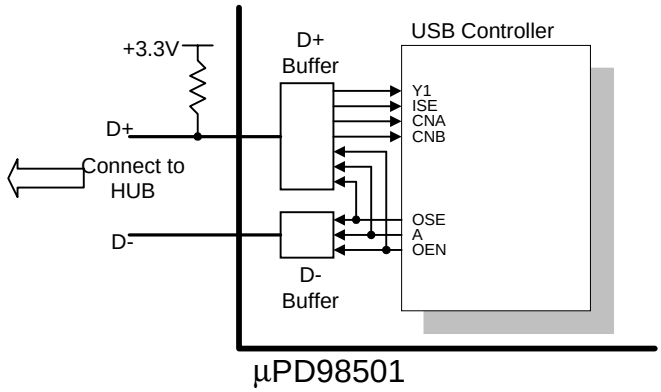
Following table indicates the Endpoint which is used to send data and the Endpoint at which the data will be received.

| Tx EndPoint | Rx EndPoint |
|---|---|
| EndPoint0 (Control) | EndPoint0 (Control) |
| EndPoint1 (Isochronous) | EndPoint2 (Isochronous) |
| EndPoint3 (Bulk) | EndPoint4 (Bulk) |
| EndPoint5 (Interrupt) | EndPoint6 (Interrupt) |

## 6.9 Example of Connection

USB Controller is connected to the μPD98501 internal USB I/O buffer and the μPD98501 internal bus (IBUS) as shown in the following figure 6-30.

**Figure 6-30. Example of Connection**




When designing a PCB, it is necessary to connect a 1.5 k ohm pull-up resistor between the D+ pin and the 3.3 V power supply to indicate the presence of a full-speed device. In addition, the impedance between the USB connector and each of the D+ and D- pins must be adjusted to 27 ohm.

The circuit must be designed such that the μPD98501 power supply is turned on and off together with the external 3.3 V power supply. If the μPD98501 power supply is off, but the external 3.3 V power supply is on, the USB HUB connected to the μPD98501 will assume that a new device has been connected but, because the μPD98501 power is off, no response can be returned.

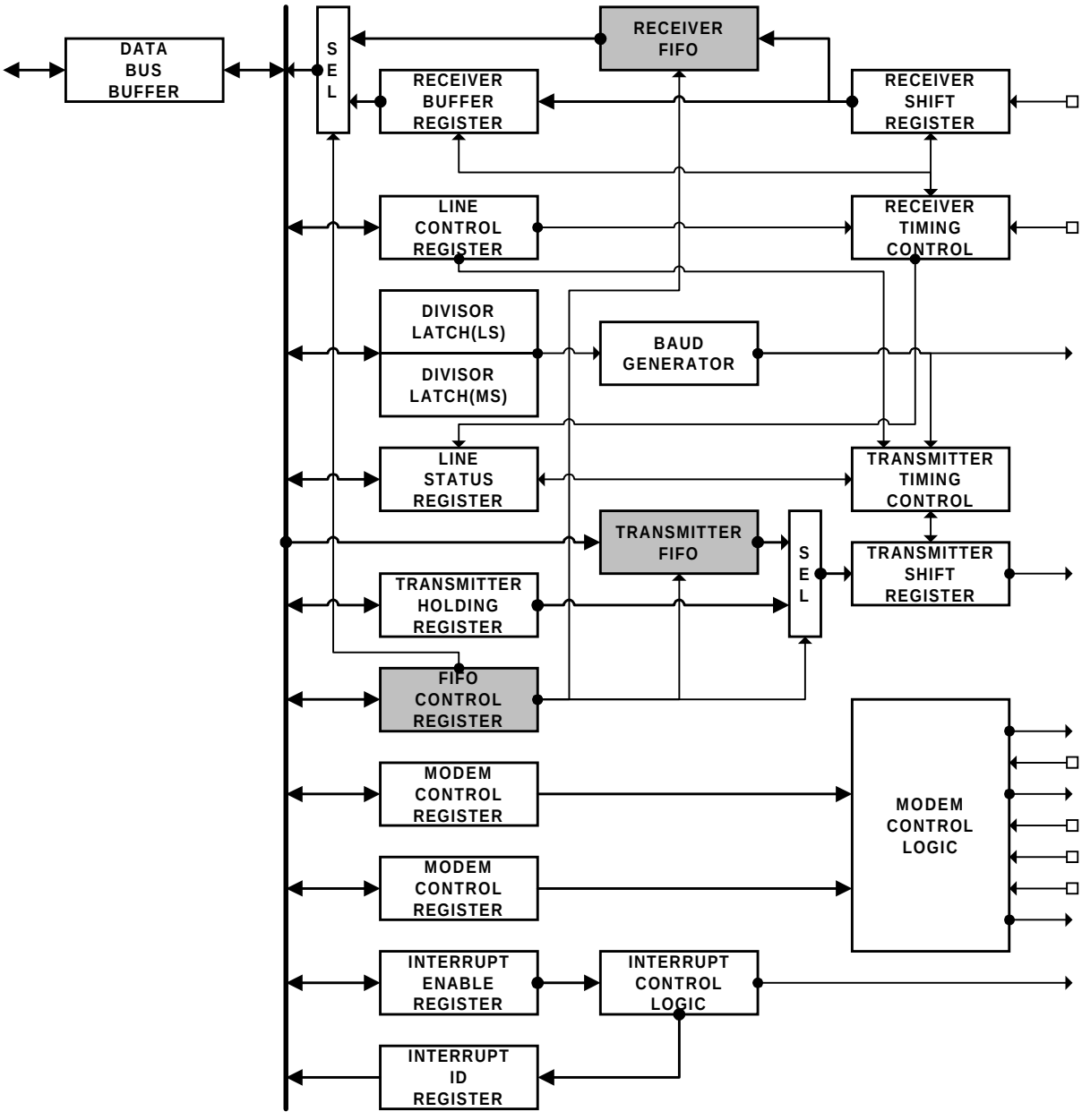
For details of the electrical specifications of the USB, refer to USB Specification 1.1.

# CHAPTER 7 UART

## 7.1 Overview

UART is a serial interface that conforms to the RS-232-C communication standard and is equipped with two one-channel interfaces, one for transmission and one for reception. This unit is functionally compatible with the NS16550D.

## 7.2 UART Block Diagram




**Caution The FIFO control blocks are implemented in the UART, but not support this feature**

## 7.3 UART Registers

This controller uses the NEC NA16550L Mega-Function as its internal UART. This UART is functionally identical to the National Semiconductor NS16550D. Refer to the NEC “User’s Manual. Mega FunctionNA16550L” for more information and programming details.

### 7.3.1 UART Receiver data Buffer Register (UARTRBR) (80H, DLAB = 0, R)

This register holds receive data. It is only accessed when the Divisor Latch Access bit(DLAB) is cleared in the UARTLCR.

| Bits | Field | Description |
|---|---|---|
| 7:0 | UDATA | UART data (read only) when DLAB = 0. |
| 31:8 | Reserved | Hardwired to 0. |

### 7.3.2 UART Interrupt Enable Register (UARTIER) (84H, DLAB = 1, R/W)

This register is used to enable UART interrupts. It is only accessed when the Divisor Latch Access bit (DLAB) is set in the UARTLCR. The UARTIM bit2 in Interrupt Mask Register “IMR” is a global enable for interrupt sources enabled by this register.

| Bits | Field | Description |
|---|---|---|
| 0 | ERBFI | UART Receive data Buffer Full Interrupt<br>1 = Enable receive-data-available interrupt<br>0 = Disable such interrupts<br>Receive data-Buffer-Full state reported to UARTLSR |
| 1 | ERBEI | UART Transmitter Buffer empty Interrupt<br>1 = Enable Transmitter Buffer empty interrupt<br>0 = Disable such interrupts<br>Transmitter Buffer empty state reported to UARTLSR |
| 2 | ERBLI | UART Line status Interrupts<br>1 = Enable Line status error interrupt<br>0 = Disable such interrupts<br>Line status error interrupt state reported to UARTLSR |
| 3 | ERBMI | UART Modem status Interrupts<br>1 = Enable Modem status change interrupt<br>0 = Disable such interrupts<br>Modem status changes are reported to UARTMSR |
| 31:4 | Reserved | Hardwired to 0. |

### 7.3.3 UART Divisor Latch LSB Register (UARTDLL) (80H, DLAB = 1, R/W)

This register is used to set the divisor (division rate) for the baud rate generator. The data in this register and the upper 8-bit data in UARTDLM register are together handled as 16-bit data.

| Bits | Field | Description |
|---|---|---|
| 7:0 | DIVLSB | UART divisor latch (least significant byte)<br>Only accessed when DLAB = 1 in UARTLCR |
| 31:8 | Reserved | Hardwired to 0. |

### 7.3.4 UART Divisor Latch MSB Register (UARTDLM) (84H, DLAB = 1, R/W)

This register is used to set the divisor (division rate) for the baud rate generator. The data in this register and the lower 8-bit data in UARTDLL register are together handled as 16-bit data.

| Bits | Field | Description |
|---|---|---|
| 7:0 | DIVLSB | UART divisor latch (least significant byte)<br>Only accessed when DLAB = 1 in UARTLCR |
| 31:8 | Reserved | Hardwired to 0. |

**Table 7-1. Correspondence between Baud Rates and Divisors**

| Baud Rate | UART Source Clock Frequency | | | | | |
|---|---|---|---|---|---|---|
| | 18.432MHz (Use External Clock) | | 33.000MHz (CPU Clock = 66MHz) | | 50.000MHz (CPU Clock = 100MHz) | |
| | Divisor | Error | Divisor | Error | Divisor | Error |
| 50 | 23040 | 0 | 41250 | >1% | 62500 | 0 |
| 75 | 15360 | 0 | 27500 | >1% | 41667 | >1% |
| 110 | 10473 | 0 | 18750 | >1% | 28409 | >1% |
| 134.5 | 8565 | 0 | 15335 | >1% | 23234 | >1% |
| 150 | 7680 | 0 | 13750 | >1% | 20833 | >1% |
| 300 | 3840 | 0 | 6875 | >1% | 10417 | >1% |
| 600 | 1920 | 0 | 3438 | >1% | 5208 | >1% |
| 1200 | 920 | 0 | 1719 | >1% | 2604 | >1% |
| 1800 | 640 | 0 | 1146 | >1% | 1736 | >1% |
| 2000 | 573 | 0 | 1031 | >1% | 1562 | >1% |
| 2400 | 480 | 0 | 859 | >1% | 1302 | >1% |
| 3600 | 320 | 0 | 573 | >1% | 868 | >1% |
| 4800 | 240 | 0 | 430 | >1% | 651 | >1% |
| 7200 | 160 | 0 | 286 | >1% | 434 | >1% |
| 9600 | 120 | 0 | 215 | >1% | 326 | >1% |
| 19200 | 60 | 0 | 107 | >1% | 163 | >1% |
| 38400 | 30 | 0 | 54 | >1% | 81 | >1% |
| 56000 | 21 | 2.04% | 37 | >1% | 56 | >1% |
| 128000 | 9 | 0 | 16 | >1% | 24 | 1.69% |
| 144000 | 8 | 0 | 14 | 2.25% | 22 | 1.38% |
| 192000 | 6 | 0 | 11 | 2.41% | 16 | 1.69% |
| 230400 | 5 | 0 | 9 | >1% | 14 | 3.22% |
| 288000 | 4 | 0 | 7 | 2.25% | 11 | 1.38% |
| 384000 | 3 | 0 | 5 | 6.90% | 8 | 1.69% |
| 576000 | 2 | 0 | 4 | 11.7% | 5 | 7.83% |
| 1152000 | 1 | 0 | 2 | 11.7% | 3 | 10.6% |

**Remark** If UCSEL bit in the S_GMR Register is set, The external UART clock “URCLK” is used as UART source clock.

If UCSEL bit is reset, 1/2 of CPU clock is used as UART source clock.

### 7.3.5 UART Interrupt ID Register (UARTIIR) (88H, R)

This register indicates priority levels for interrupts and existence of pending interrupt. From highest to lowest priority, these interrupts are receive line status, receive data ready, character timeout, transmit holding register empty, and modem status. The contents of UARTIIR[3] bit is valid only in FIFO mode, and it is always 0 in 16550 mode. UARTIIR[2] bit becomes 1 when UARTIIR[3] bit is set to 1.

| Bits | Field | Description | | |
|---|---|---|---|---|
| 0 | INTPENDL | Pending interrupts<br>0 = UART Interrupt pending(read only)<br>1 = No UART interrupt pending | | |
| 3:1 | UIID | Indicates the priority level of pending interrupt. | | |
| | | UIID[2:0] | Priority | Source of interrupt |
| | | 3 | Highest | Receiver Line status:<br>Overrun Error, Parity, Framing Error, or Break interrupt |
| | | 2 | 2nd Highest | Received data available:<br>Receiver Data Available or Trigger Level Reached |
| | | 6 | 3rd Highest | Character timeout indication:<br>No change in receiver FIFO during the last four character times and FIFO is not empty. |
| | | 1 | 4th Highest | Transmitter holding Register Empty |
| | | 0 | 5th Highest | Modem Status: CTS_L, DSR_L or DCD_L. |
| 5:4 | Reserved | Hardwired to 0. | | |
| 7:6 | UFIFOEN | UART FIFO is enable (read only)<br>Both bits set to 1 when the transmit/receive FIFO is enabled in the UFIFOEN0 bit is set in the UARTFCR. | | |
| 31:8 | Reserved | Hardwired to 0. | | |

**Remark** The μPD98501 does NOT support the FIFIO mode, thus the UFIFOEN field will show zero.

### 7.3.6 UART FIFO Control Register (UARTFCR) (88H, W)

This register is used to control the FIFOs: enable FIFO, clear FIFO, and set the receive FIFO trigger level. (This UARTFCR register is not available, this is because that it is not possible to use FIFO mode. )

| Bits | Field | Description |
|---|---|---|
| 0 | UFIFOEN0 | UART Receiver FIFO Enable. (write-only)<br>1 = enable receive and transmit FIFOs.<br>0 = disable and clear receive and transmit FIFOs. |
| 1 | URFRST | UART Receiver FIFO Reset. (write-only)<br>1 = clear receive FIFO and reset counter.<br>0 = no clear. |
| 2 | UTFRST | UART Transmitter FIFO Reset. (write-only)<br>1 = clear transmit FIFO and reset counter.<br>0 = no clear. |
| 3 | FIFOMD | Switch between 16550 mode and FIFO mode<br>1= From 16550 mode to FIFO mode (INVALID)<br>0= From FIFO mode to 16550 mode |
| 5:4 | Reserved | Hardwired to 0. |
| 7:6 | URFTR | UART Receive FIFO Trigger level.<br>When the trigger level is reached, a Receive-buffer-Full interrupt is generated, if enable by the ERBFI bit in the UARTIER. Number of bytes in Receiver FIFO is following.<br>0 = 1byte<br>1 = 4byte<br>2 = 8byte<br>3 = 14bytes |
| 31:8 | Reserved | Hardwired to 0. |

### 7.3.7 UART Line Control Register (UARTLCR) (8CH, R/W)

This register is used to specify the format for asynchronous communication and exchange and to set the divisor latch access bit. Bit 6 is used to send the break status to the receive side's UART. When bit 6 = 1, the serial output (URSDO) is forcibly set to the spacing (0) state. The setting of bit 5 becomes valid according to settings in bits 4 and 3.

| Bits | Field | Description |
|---|---|---|
| 1:0 | WLS | Word length select.<br>11 = 8 bits<br>10 = 7 bits<br>01 = 6 bits<br>00 = 5 bits |
| 2 | STB | Number of stop bits.<br>1 = 2 bits, except 1.5 stop bits for 5 words<br>0 = 1 bit |
| 3 | PEN | Parity enable.<br>1 = generate parity on writes, check it on reads.<br>0 = no parity generation or checking.<br>For the UART, even or odd parity can be generated or checked, as specified in Bit 4 (EPS). |
| 4 | EPS | Parity select.<br>1 = even parity<br>0 = odd parity |
| 5 | USP | Stick parity.<br>Please set 0. |
| 6 | REV | Reserved for future use.<br>Please set 0 when VR4120A access. |
| 7 | DLAB | Divisor Latch access bit.<br>1 = access baud-rate divisor at offset 84H<br>0 = access URSDO/URSDI and IE at offset 84H<br>When this bit is set, UART accesses the UART Divisor Latch LSB Register (UARTDLM) at offset 84H. When cleared, the UART accesses the Receiver Data Buffer Register (UARTRBR) on reads at offset 80H, the UARTTHR on writes at offset 80H, and UARTIER on any accesses at offset 84H. |
| 31:8 | Reserved | Hardwired to 0. |

### 7.3.8 UART Modem Control Register (UARTMCR) (90H, R/W)

This register controls the state of external URDTR_B and URRTS_B modem-control signals and of the loop-back test.

| Bits | Field | Description |
|---|---|---|
| 0 | DTR | Data Terminal Ready.<br>1 = negate URDTR_B signal.<br>0 = assert URDTR_B signal. |
| 1 | RTS | Request To Send.<br>1 = negate URRTS_B signal.<br>0 = assert URRTS_B signal. |
| 2 | OUT1 | Out 1 (internal).<br>1 = OUT1_B (internal) state active.<br>0 = OUT1_B (internal) state inactive (reset value).<br>This is a user-defined bit that has no associated external signal. Software can write to the bit, but this has no effect. |
| 3 | OUT2 | Out 2 (internal).<br>1 = OUT2_B (internal) state active.<br>0 = OUT2_B (internal) state inactive (reset value).<br>This is a user-defined bit that has no associated external signal. Software can write to the bit, but this has no effect. |
| 4 | LOOP | Loop-Back Test.<br>1 = loop-back.<br>0 = normal operation.<br>This is an NEC internal test function. |
| 31:5 | Reserved | Hardwired to 0. |

### 7.3.9 UART Line Status Register (UARTLSR) (94H, R/W)

This register reports the current state of the transmitter and receiver logic.

| Bits | Field | Description |
|---|---|---|
| 0 | DR | Receive-Data Ready.<br>1 = receive data buffer full.<br>0 = receive data buffer not full.<br>Receive data is stored in the UART Receiver Data Buffer Register (UARTRBR). |
| 1 | OE | Receive-Data Overrun Error.<br>1 = overrun error on receive data.<br>0 = no such error. |
| 2 | PE | Receive-Data Parity Error.<br>1 = parity error on receive data.<br>0 = no such error. |
| 3 | FE | Receive-Data Framing Error.<br>1 = framing error on receive data.<br>0 = no such error. |
| 4 | BI | Break Interrupt.<br>1 = break received on URSDI signal.<br>0 = no break. |
| 5 | THRE | Transmitter Holding Register Empty.<br>1 = transmitter holding register empty.<br>0 = transmitter holding register not empty.<br>Transmit data is stored in the UART Transmitter Data Holding Register (UARTTHR) |
| 6 | TEMT | Transmitter Empty.<br>1 = transmitter holding and shift registers empty.<br>0 = transmitter holding or shift register not empty. |
| 7 | RFERR | Receiver FIFO Error.<br>1 = parity, framing, or break error in receiver buffer.<br>0 = no such error. |
| 31:8 | Reserved | Hardwired to 0. |

**Remark** RFERR will not become to 1, because the μPD98501 does NOT support FIFO mode.

### 7.3.10 UART Modem Status Register (UARTMSR) (98H, R/W)

This register reports the current state of and changes in various control signals.

| Bits | Field | Description |
|---|---|---|
| 0 | DCTS | Delta Clear To Send.<br>1 = URCTS_B state changed since this register was last read.<br>0 = no such change. |
| 1 | DDSR | Delta Data Set Ready.<br>1 = URDSR_B input signal changed since this register was last read.<br>0 = no such change. |
| 2 | TERI | Trailing Edge Ring Indicator.<br>1 = RI_B state changed since this register last read.<br>0 = no such change.<br>RI_B is not implemented as an external signal, so this bit is never set by the controller. |
| 3 | DDCD | Delta Data Carrier Detect.<br>1 = URDCD_B state changed since this register was last read.<br>0 = no such change. |
| 4 | CTS | Clear To Send.<br>1 =URCTS_B state active.<br>0 = URCTS_B state inactive.<br>This bit is the complement of the URCTS_B input signal. If the LOOP bit in the UART Modem Control Register (UARTMCR), is set to 1, the CTS bit is equivalent to the RTS bit in the UARTMCR. |
| 5 | DSR | Data Set Ready.<br>1 = URDSR_B state active.<br>0 = URDSR_B state inactive.<br>This bit is the complement of the URDSR_B input signal. If the LOOP bit in the UART Modem Control Register (UARTMCR), is set to 1, the DSR bit is equivalent to the DTR bit in the UARTMCR. |
| 6 | RI | Ring Indicator.<br>1 = not valid.<br>0 = always reads 0.<br>This bit has no associated external signal. |
| 7 | DCD | Data Carrier Detect.<br>1 =URDCD_B state active.<br>0 = URDCD_B state inactive.<br>This bit is the complement of the URDCD_B input signal. If the LOOP bit in the UART Modem Control Register (UARTMCR), is set to 1, the DCD bit is equivalent to the OUT2 bit in the UARTMCR. |
| 31:8 | Reserved | Hardwired to 0. |

### 7.3.11 UART Scratch Register (UARTSCR) (9CH, R/W)

This register contains a UART reset bit plus 8 bits of space for any software use.

| Bits | Field | Description |
|---|---|---|
| 7:0 | USCR | UART Scratch Register.<br>Available to software for any purpose. |
| 31:8 | Reserved | Hardwired to 0. |

**[MEMO]**

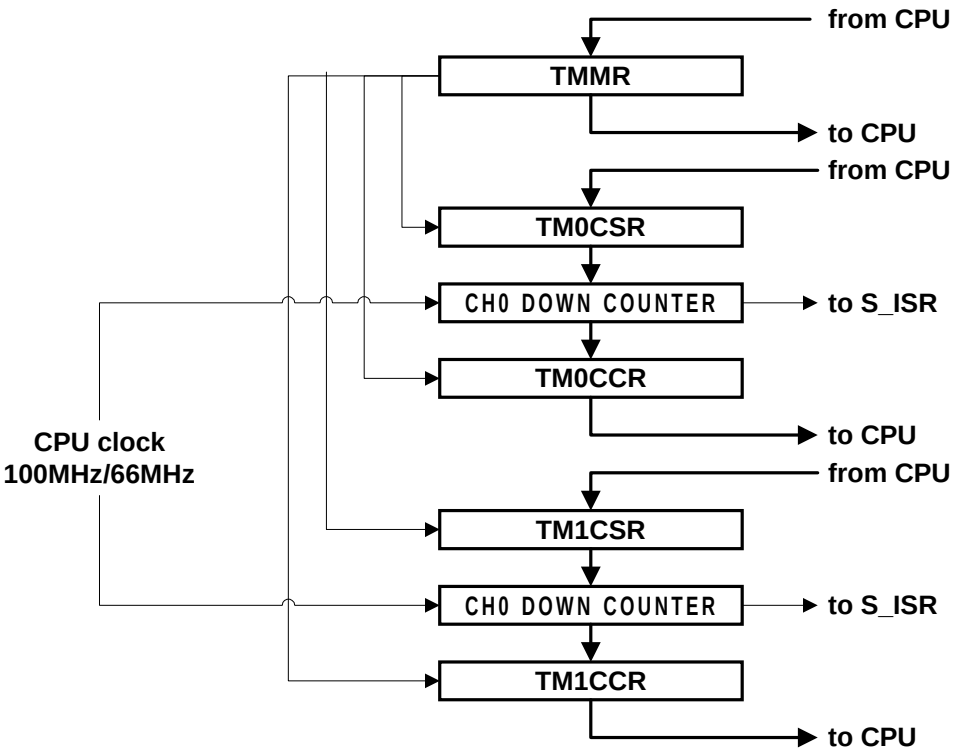
# CHAPTER 8 TIMER

## 8.1 Overview

There are two Timers. The timers are clocked at the system clock rate. All two timers are read/writable by the CPU. Timers can be read by the CPU while they are counting. They can be automatically reloaded with the previously loaded value and restarted or can be stopped while in progress. Two timers issues interrupt to the CPU upon reaching their maximum value, the interrupts can be enabled/disabled.

The TM0IS and TM1IS fields in the Interrupt Status Register “S_ISR” indicate the end of timer count, when set indicate there is a timer event that completed. All timers count up. The read-write registers “TM0CSR” or “TM1CSR” have different offset from the read register so write registers are not affected while a value is read from the read registers “TM0CCR” / ”TM1CCR” which indicate a running count of the timer/counter at a given time. Once a value is loaded in the TM0CSR/TM1CSR, it stays there until Timer’s interrupts are cleared in the Interrupt Status Register “S_ISR”. The original value can be reloaded in the counter to restart it from that count if Timer CH0 / CH1 reload enable bit is set in the Timer Mode Register “TMMR”. Interrupts are automatically cleared upon CPU reading the Interrupt Status Register of System Controller “S_ISR”.

## 8.2 Block Diagram



## 8.3 Timer Registers

### 8.3.1 Timer Mode Register (TMMR) (B0H, R/W)

The Timer Mode Register “TMMR” is read-write and word aligned 32bit register. TMMR is used to control the timer. TMMR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 0 | TM0EN | Timer CH0 enable:<br>1 = Enable, starts the timer<br>(Automatically reloads the original timer value and starts if timer had expired)<br>0 = Disable, stops the timer |
| 7:1 | Reserved | Hardwired to 0. |
| 8 | TM1EN | Timer CH1 enable:<br>1 = Enable, starts the timer<br>(Automatically reloads the original timer value and starts if timer had expired)<br>0 = Disable, stops the timer |
| 31:9 | Reserved | Hardwired to 0. |

### 8.3.2 Timer CH0 Count Set Register (TM0CSR) (B4H, R/W)

The Timer CH0 Count Set Register “TM0CSR” is read-write and word aligned 32bit register. CPU (VR4120A) loads a value in it and the counter starts counting down from the (TM0CSR –1) value, when it reaches 0000_0000H, it generates an interrupt to the CPU via Interrupt Status Register “ISR” if the TM0IS in ISR is not masked by TM0IM in IMR. TM0CSR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 31:0 | TM0SET | Initial and Reload Value for Timer CH0 |

### 8.3.3 Timer CH1 Count Set Register (TM1CSR) (B8H, R/W)

The Timer CH1 Count Set Register “TM1CSR” is read-write and word aligned 32bit register. CPU (VR4120A) loads a value in it and the counter starts counting down from the (TM1CSR –1) value, when it reaches 0000_0000H, it generates an interrupt to the CPU via Interrupt Status Register “ISR” if the TM1IS in ISR is not masked by TM1IM in IMR. TM1CSR is initialized to 0 at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 31:0 | TM1SET | Initial and Reload Value for Timer CH1 |

### 8.3.4 Timer CH0 Current Count Register (TM0CCR) (BCH, R)

The Timer CH0 Current Count Register “TM0CCR” is read-only and word aligned 32bit register. CPU (VR4120A) can read its value to get timer CH0 current count. TM0CSR is initialized to FFFF_FFFFH at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 31:0 | TM0CNT | Timer CH0 Current Count Value |

### 8.3.5 Timer CH1 Current Count Register (TM1CCR) (C0H, R)

The Timer CH1 Current Count Register “TM1CCR” is read-only and word aligned 32bit register. CPU (VR4120A) can read its value to get timer CH1 current count. TM1CCR is initialized to FFFF_FFFFH at reset and contains the following fields:

| Bits | Field | Description |
|---|---|---|
| 31:0 | TM0CNT | Timer CH0 Current Count Value |

**[MEMO]**

# CHAPTER 9 MICRO WIRE

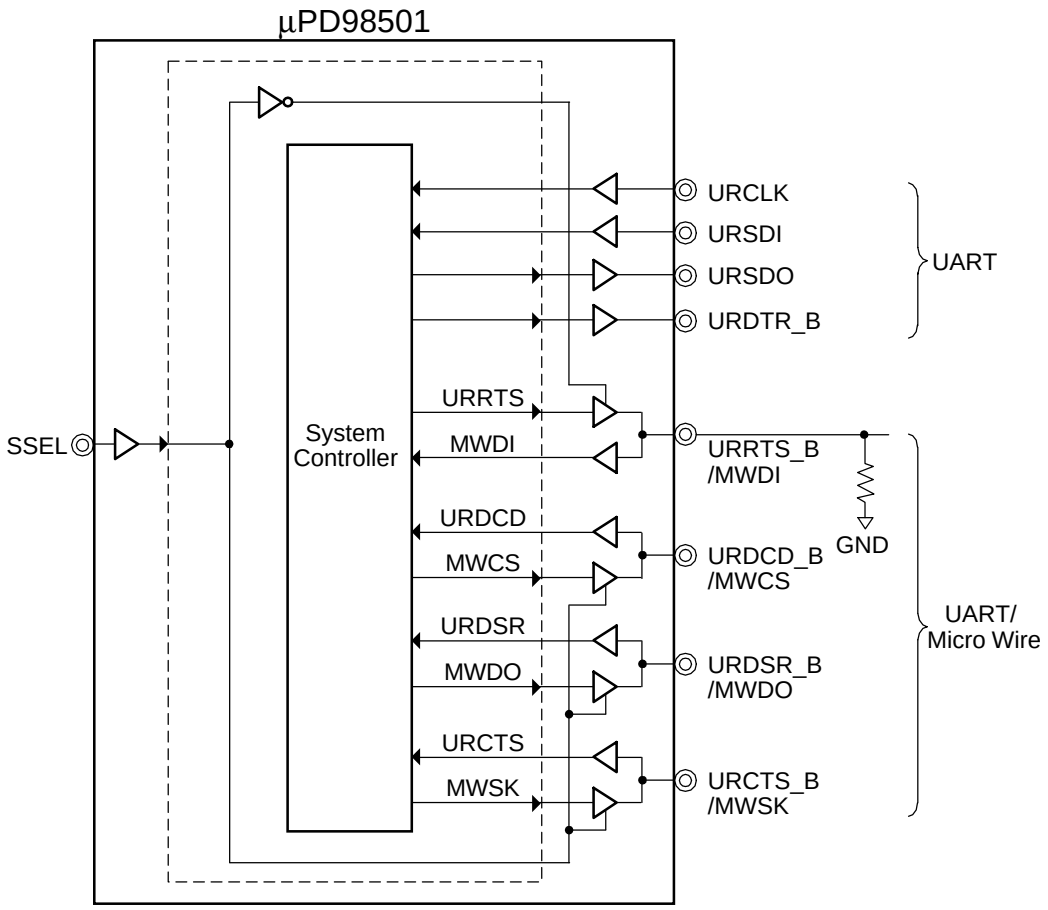
## 9.1 Overview

EEPROM interface has compatibility with MICROWIRE serial interface (This port is shared with UART I/F).

MAC Address is stored in EEPROM, and it automatically loaded to MAC address registers during the power up sequence.

## 9.2 Micro Wire Connection Diagram

**Figure 9-1. Micro Wire Connection Diagram**




Function select SSEL=1:MICROWIRE Mode

- MWSK : Clock output line
- MWCS : Chip select line
- MWDO : Serial data output line
- MWDI : Serial data input line

## 9.3 Operations

### 9.3.1 Data read at the power up load

After reset release, power up load processes starts .

In case of the value from EEPROM address 00h is :

1. 5A5Ah

   System Controller sets the EEPROM data (address:01h-06h) in the internal registers (MACAR1, MACAR2, MACAR3).

2. NOT 5A5Ah

   System Controller sets the fix data “0000_0000h” in the internal registers (MACAR1, MACAR2, MACAR3).

**Table 9-1. EEPROM Initial Data**

| EEPROM Address | Data | Stored Register |
|---|---|---|
| 00h | 5A5A h | - |
| 01h | MAC1 Address data[15: 0] | MACAR1[15: 0] |
| 02h | MAC1 Address data[31:16] | MACAR1[31:16] |
| 03h | MAC1 Address data[45:32] | MACAR2[15: 0] |
| 04h | MAC2 Address data[15: 0] | MACAR2[31:16] |
| 05h | MAC2 Address data[31:16] | MACAR3[15: 0] |
| 06h | MAC2 Address data[45:32] | MACAR3[31:16] |

### 9.3.2 Accessing to EEPROM

Access to EEPROM starts by writing to the ECCR (EEPROM Command Control Register).

Write command (3bit) and address (6bit) of EEPROM into lower 9bit of ECCR.

There is a difference between write command and read command.

1. Write command

   Write data into upper 16bit of ECCR.

2. Read command

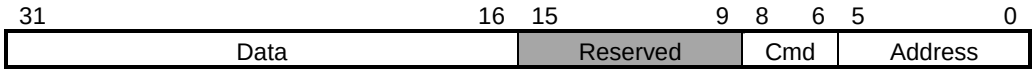
   Data is loaded into lower 16bit of ERDR (EEPROM Read Data Register).

**Table 9-2. EEPROM Command List**

| Command | bit8-6 | bit5-0 | Operation |
|---|---|---|---|
| READ | 110 | A5-A0 | Read data from EEPROM assigned by address A5-A0. |
| EWEN | 100 | 11xxxx | Enable Erase/Write command.<br>This command must be executed before other operating. |
| ERASE | 111 | A5-A0 | Erase data of EEPROM assigned by address A5-A0. |
| WRITE | 101 | A5-A0 | Write data to EEPROM assigned by address A5-A0. |
| ERAL | 100 | 10xxxx | Erase all data of EEPROM. |
| WRAL | 100 | 01xxxx | Write all data to EEPROM. |
| EWDS | 100 | 00xxxx | Disable Erase/Write command. |

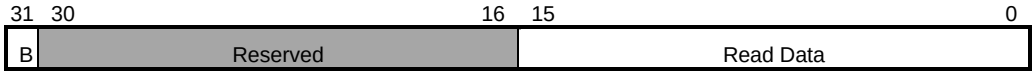
## 9.4 Registers

### 9.4.1 ECCR (EEPROM Command Control Register) Address 1000_00D0h [Write Only]



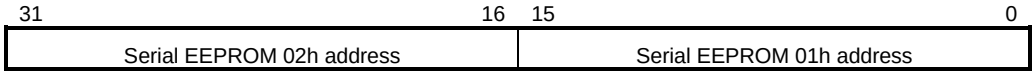

bit 31-16 : Write data to Serial EEPROM. No meaning in case of data read.
bit 15- 9 : Reserved
bit 8- 6 : Serial EEPROM command.
bit 5- 0 : Serial EEPROM address.

### 9.4.2 ERDR (EEPROM Read Data Register) Address 1000_00D4h [Read Only]



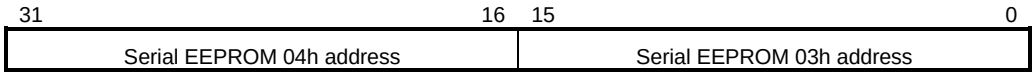

bit 31 : Operation status of MICROWIRE block.
1 : busy
0 : idle
bit 30-16 : Reserved
bit 15- 0 : Read data from Serial EEPROM.

### 9.4.3 MACAR1 (MAC Address Register 1) Address 1000_00D8h [Read Only]



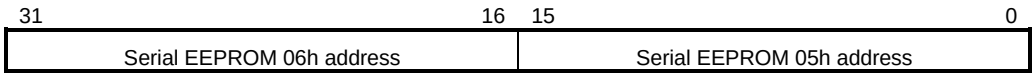

bit 31-0 : Stored Serial EEPROM data of address 01h,02h.

### 9.4.4 MACAR2 (MAC Address Register 2) Address 1000_00DCh [Read Only]




bit 31-0 : Stored Serial EEPROM data of address 03h,04h.

### 9.4.5 MACAR3 (MAC Address Register 3) Address 1000_00E0h [Read Only]




bit 31-0 : Stored Serial EEPROM data of address 05h,06h.

**[MEMO]**

# APPENDIX A MIPS III INSTRUCTION SET DETAILS

This chapter provides a detailed description of the operation of each instruction in both 32- and 64-bit modes. The instructions are listed in alphabetical order.

## A.1 Instruction Notation Conventions

In this chapter, all variable subfields in an instruction format (such as *rs*, *rt*, *immediate*, etc.) are shown in lowercase names.

For the sake of clarity, we sometimes use an alias for a variable subfield in the formats of specific instructions. For example, we use *base* instead of *rs* in the format for load and store instructions. Such an alias is always lower case, since it refers to a variable subfield.

Figures with the actual bit encoding for all the mnemonics are located at the end of this chapter (**A.6 CPU Instruction Opcode Bit Encoding**), and the bit encoding also accompanies each instruction.

In the instruction descriptions that follow, the operation section describes the operation performed by each instruction using a high-level language notation. The VR4120A CPU can operate as either a 32- or 64-bit microprocessor and the operation for both modes is included with the instruction description.

Special symbols used in the notation are described in Table A-1.

**Table A-1. CPU Instruction Operation Notations**

| Symbol | Meaning |
|---|---|
| ← | Assignment |
| \|\| | Bit string concatenation |
| $x^y$ | Replication of bit value *x* into a *y*-bit string. *x* is always a single-bit value |
| xy:z | Selection of bits *y* through *z* of bit string *x*. Little-endian bit notation is always used. If *y* is less than *z*, this expression is an empty (zero length) bit string |
| + | 2's complement or floating-point addition |
| - | 2's complement or floating-point subtraction |
| * | 2's complement or floating-point multiplication |
| div | 2's complement integer division |
| mod | 2's complement modulo |
| / | Floating-point division |
| < | 2's complement less than comparison |
| and | Bit-wise logical AND |
| or | Bit-wise logical OR |
| xor | Bit-wise logical XOR |
| nor | Bit-wise logical NOR |
| GPR [*x*] | General-Register *x*. The content of GPR [0] is always zero. Attempts to alter the content of GPR [0] have no effect. |
| CPR [*z, x*] | Coprocessor unit *z*, general register *x*. |
| CCR [*z, x*] | Coprocessor unit *z*, control register *x*. |
| COC [*z*] | Coprocessor unit *z* condition signal. |
| BigEndianMem | Big-endian mode as configured at reset (0 → Little, 1 → Big). Specifies the endianness of the memory interface (see LoadMemory and StoreMemory), and the endianness of Kernel and Supervisor mode execution. However, this value is always 0 since the VR4120A CPU supports the little endian order only. |
| ReverseEndian | Signal to reverse the endianness of load and store instructions. This feature is available in User mode only, and is effected by setting the RE bit of the Status register. Thus, ReverseEndian may be computed as (SR25 and User mode).However, this value is always 0 since the VR4120A CPU supports the little endian order only. |
| BigEndianCPU | The endianness for load and store instructions (0 → Little, 1 → Big). In User mode, this endianness may be reversed by setting RE bit. Thus, BigEndianCPU may be computed as BigEndianMem XOR ReverseEndian.However, this value is always 0 since the VR4120A CPU supports the little endian order only. |
| T + *i*: | Indicates the time steps between operations. Each of the statements within a time step are defined to be executed in sequential order (as modified by conditional and loop constructs). Operations which are marked *T + i:* are executed at instruction cycle *i* relative to the start of execution of the instruction. Thus, an instruction which starts at time *j* executes operations marked T + *i:* at time *i + j*. The interpretation of the order of execution between two instructions or two operations that execute at the same time should be pessimistic; the order is not defined. |

**(1) Instruction notation examples**

The following examples illustrate the application of some of the instruction notation conventions:

**Example 1:**

$$GPR\ [rt] \leftarrow immediate\ ||\ 0^{16}$$

Sixteen zero bits are concatenated with an immediate value (typically 16 bits), and the 32-bit string is assigned to general register *rt*.

**Example 2:**

$$(immediate15)^{16}\ ||\ immediate15...0$$

Bit 15 (the sign bit) of an immediate value is extended for 16-bit positions, and the result is concatenated with bits 15 through 0 of the immediate value to form a 32-bit sign extended value.

## A.2 Load and Store Instructions

In the VR4120A CPU implementation, the instruction immediately following a load may use the loaded contents of the register. In such cases, the hardware interlocks, requiring additional real cycles, so scheduling load delay slots is still desirable, although not required for functional code.

In the load and store descriptions, the functions listed in Table A-2 are used to summarize the handling of virtual addresses and physical memory.

**Table A-2. Load and Store Common Functions**

| Function | Meaning |
|---|---|
| AddressTranslation | Uses the TLB to find the physical address given the virtual address. The function fails and an exception is taken if the required translation is not present in the TLB. |
| LoadMemory | Uses the cache and main memory to find the contents of the word containing the specified physical address. The low-order three bits of the address and the Access Type field indicate which of each of the four bytes within the data word need to be returned. If the cache is enabled for this access, the entire word is returned and loaded into the cache. If the specified data is short of word length, the data position to which the contents of the specified data is stored is determined considering the endian mode and reverse endian mode. |
| StoreMemory | Uses the cache, write buffer, and main memory to store the word or part of word specified as data in the word containing the specified physical address. The low-order three bits of the address and the Access Type field indicate which of each of the four bytes within the data word should be stored. If the specified data is short of word length, the data position to which the contents of the specified data is stored is determined considering the endian mode and reverse endian mode. |

As shown in Table A-3, the Access Type field indicates the size of the data item to be loaded or stored. Regardless of access type or byte-numbering order (endian), the address specifies the byte that has the smallest byte address in the addressed field. This is the rightmost byte in the VR4120A CPU since it supports the little-endian order only.

**Table A-3. Access Type Specifications for Loads/Stores**

| Access Type | Meaning |
|---|---|
| DOUBLEWORD | 8 bytes (64 bits) |
| SEPTIBYTE | 7 bytes (56 bits) |
| SEXTIBYTE | 6 bytes (48 bits) |
| QUINTIBYTE | 5 bytes (40 bits) |
| WORD | 4 bytes (32 bits) |
| TRIPLEBYTE | 3 bytes (24 bits) |
| HALFWORD | 2 bytes (16 bits) |
| BYTE | 1 byte (8 bits) |

The bytes within the addressed doubleword that are used can be determined directly from the access type and the three low-order bits of the address.

## A.3 Jump and Branch Instructions

All jump and branch instructions have an architectural delay of exactly one instruction. That is, the instruction immediately following a jump or branch (that is, occupying the delay slot) is always executed while the target instruction is being fetched from storage. A delay slot may not itself be occupied by a jump or branch instruction; however, this error is not detected and the results of such an operation are undefined.

If an exception or interrupt prevents the completion of a legal instruction during a delay slot, the hardware sets the EPC register to point at the jump or branch instruction that precedes it. When the code is restarted, both the jump or branch instructions and the instruction in the delay slot are reexecuted.

Because jump and branch instructions may be restarted after exceptions or interrupts, they must be restartable. Therefore, when a jump or branch instruction stores a return link value, register *r31* (the register in which the link is stored) may not be used as a source register.

Since instructions must be word-aligned, a Jump Register or Jump and Link Register instruction must use a register which contains an address whose two low-order bits (low-order one bit in the 16-bit mode) are zero. If these low-order bits are not zero, an address exception will occur when the jump target instruction is subsequently fetched.

## A.4 System Control Coprocessor (CP0) Instructions

There are some special limitations imposed on operations involving CP0 that is incorporated within the CPU. Although load and store instructions to transfer data to/from coprocessors and to move control to/from coprocessor instructions are generally permitted by the MIPS architecture, CP0 is given a somewhat protected status since it has responsibility for exception handling and memory management. Therefore, the move to/from coprocessor instructions are the only valid mechanism for writing to and reading from the CP0 registers.

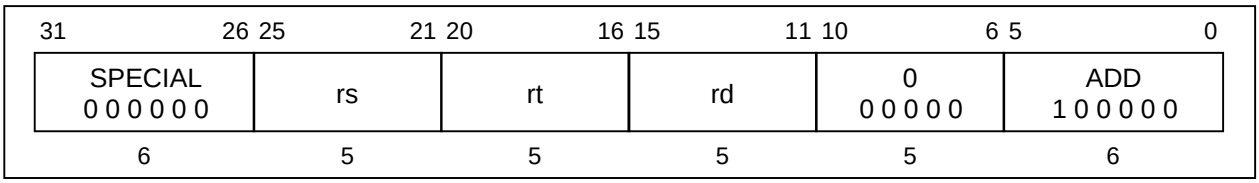
Several CP0 instructions are defined to directly read, write, and probe TLB entries and to modify the operating modes in preparation for returning to User mode or interrupt-enabled states.

## A.5 CPU Instruction

This section describes the functions of CPU instructions in detail for both 32-bit address mode and 64-bit address mode.

The exception that may occur by executing each instruction is shown in the last of each instruction's description. For details of exceptions and their processes, see **Section 2.6 Exception Processing**.

# ADD — Add — ADD




**Format:**

ADD rd, rs, rt

**Description:**

The contents of general register *rs* and the contents of general register *rt* are added to form the result. The result is placed into general register *rd*. In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

An overflow exception occurs if the carries out of bits 30 and 31 differ (2's complement overflow). The destination register *rd* is not modified when an integer overflow exception occurs.

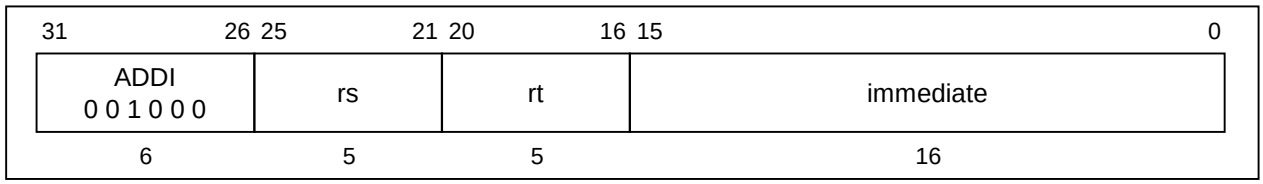
**Operation:**

```
32   T:   GPR [rd] ← GPR [rs] + GPR [rt]

64   T:   temp ← GPR [rs] + GPR [rt]
          GPR [rd] ← (temp31)^32 || temp31...0
```

**Exceptions:**

Integer overflow exception

# ADDI Add Immediate ADDI




**Format:**

ADDI rt, rs, immediate

**Description:**

The 16-bit *immediate* is sign-extended and added to the contents of general register *rs* to form the result. The result is placed into general register *rt.* In 64-bit mode, the operand must be valid sign-extended, 32-bit values.

An overflow exception occurs if carries out of bits 30 and 31 differ (2's complement overflow). The destination register *rt* is not modified when an integer overflow exception occurs.

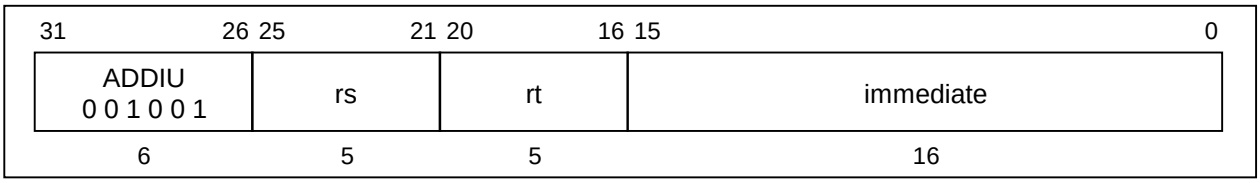
**Operation:**

```
32   T:   GPR [rt] ← GPR [rs] + (immediate15)16 || immediate15...0

64   T:   temp ← GPR [rs] + (immediate15)48 || immediate15...0
          GPR [rt] ← (temp31)32 || temp31...0
```

**Exceptions:**

Integer overflow exception

# ADDIU Add Immediate Unsigned ADDIU




**Format:**

ADDIU rt, rs, immediate

**Description:**

The 16-bit *immediate* is sign-extended and added to the contents of general register *rs* to form the result. The result is placed into general register *rt.* No integer overflow exception occurs under any circumstances. In 64-bit mode, the operand must be valid sign-extended, 32-bit values.

The only difference between this instruction and the ADDI instruction is that ADDIU never causes an integer overflow exception.

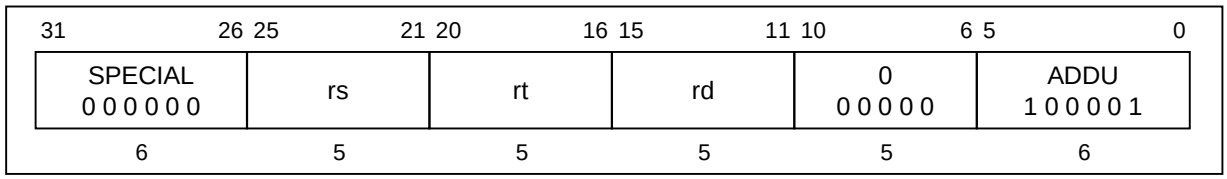
**Operation:**

```
32    T:    GPR [rt] ← GPR [rs] + (immediate 15)^16 || immediate 15...0

64    T:    temp ←GPR [ rs] + (immediate 15)^48 || immediate 15...0
            GPR [rt] ← (temp 31)^32 || temp 31...0
```

**Exceptions:**

None

# ADDU — Add Unsigned — ADDU




**Format:**

ADDU rd, rs, rt

**Description:**

The contents of general register *rs* and the contents of general register *rt* are added to form the result. The result is placed into general register *rd*. No integer overflow exception occurs under any circumstances. In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

The only difference between this instruction and the ADD instruction is that ADDU never causes an integer overflow exception.

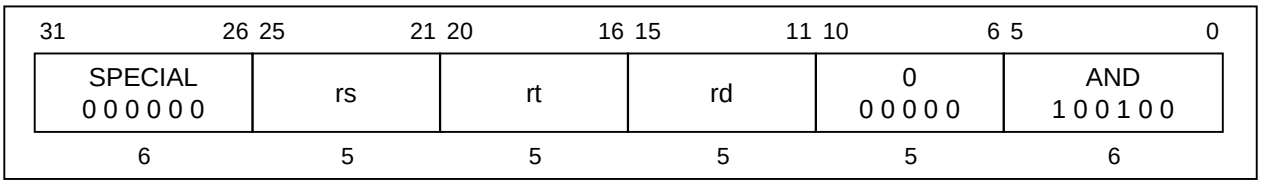
**Operation:**

```
32   T:   GPR [rd] ← GPR [rs] + GPR [rt]

64   T:   temp ← GPR [rs] + GPR [rt]
          GPR [rd] ← (temp31)^32 || temp31...0
```

**Exceptions:**

None

# AND And AND




**Format:**

AND rd, rs, rt

**Description:**

The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical AND operation. The result is placed into general register *rd*.

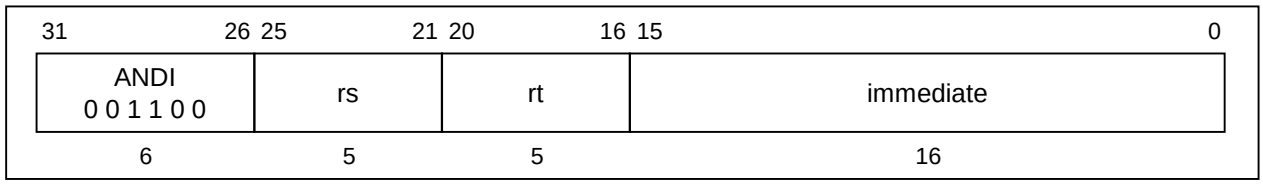
**Operation:**

```
32   T:   GPR [rd] ←GPR [rs] and GPR [rt]

64   T:   GPR [rd] ← GPR [rs] and GPR [rt]
```

**Exceptions:**

None

# ANDI And Immediate ANDI




**Format:**

ANDI rt, rs, immediate

**Description:**

The 16-bit *immediate* is zero-extended and combined with the contents of general register *rs* in a bit-wise logical AND operation. The result is placed into general register *rt*.

**Operation:**

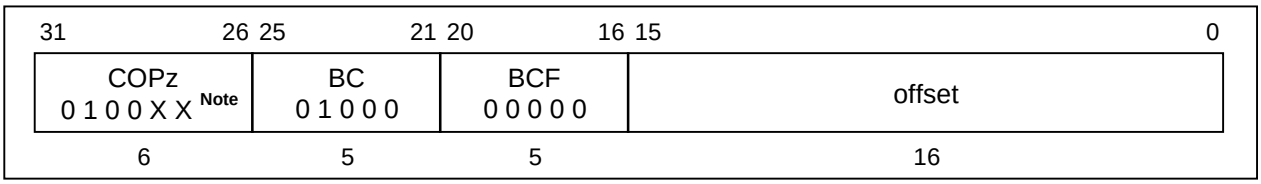
32 T: GPR [rt] $\leftarrow 0^{16}$ || (immediate and GPR [rs]$_{15...0}$)

64 T: GPR [rt] $\leftarrow 0^{48}$ || (immediate and GPR [rs]$_{15...0}$)

**Exceptions:**

None

# BC0F Branch On Coprocessor 0 False BC0F




**Format:**

BC0F offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. If contents of CP0's condition signal (CpCond), as sampled during the previous instruction, is false, then the program branches to the target address with a delay of one instruction. Because the condition line is sampled during the previous instruction, there must be at least one instruction between this instruction and a coprocessor instruction that changes the condition line.

**Operation:**

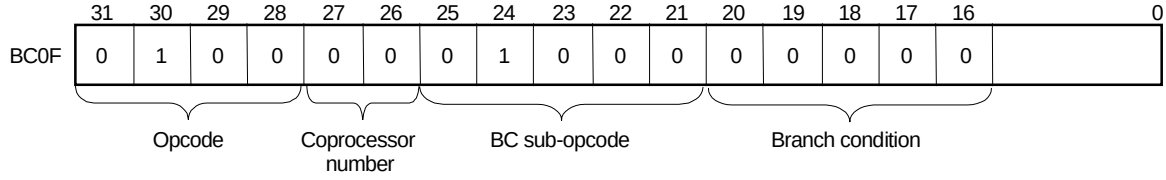
```
32    T-1:  condition ← not SR18
      T:    target ← (offset15)^14 || offset || 0^2
      T+1:  if condition then
                  PC ←  PC + target
            endif

64    T-1:  condition ← not SR18
      T:    target ← (offset15)^46 || offset || 0^2
      T+1:  if condition then
                  PC ← PC + target
            endif
```

**Exceptions:**

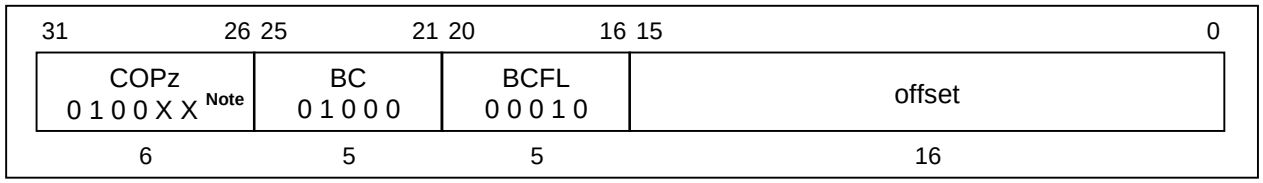
Coprocessor unusable exception

**Note** See the opcode table below, or **A.6 CPU Instruction Opcode Bit Encoding**.

**Opcode Table:**



# BC0FL Branch On Coprocessor 0 False Likely (1/2) BC0FL




**Format:**

BC0FL offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. If the contents of CP0's condition (CpCond) line, as sampled during the previous instruction, is false, the target address is branched to with a delay of one instruction.

If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

Because the condition line is sampled during the previous instruction, there must be at least one instruction between this instruction and a coprocessor instruction that changes the condition line.

**Operation:**

```
32  T-1:  condition ← not SR18
      T:  target ← (offset15)^14 || offset || 0^2
    T+1:  if condition then
              PC ← PC + target
          else
              NullifyCurrentInstruction
          endif

64  T-1:  condition ← not SR
      T:  target ← (offset15)^46 || offset || 0^2
    T+1:  if condition then
              PC ← PC + target
          else
              NullifyCurrentInstruction
          endif
```

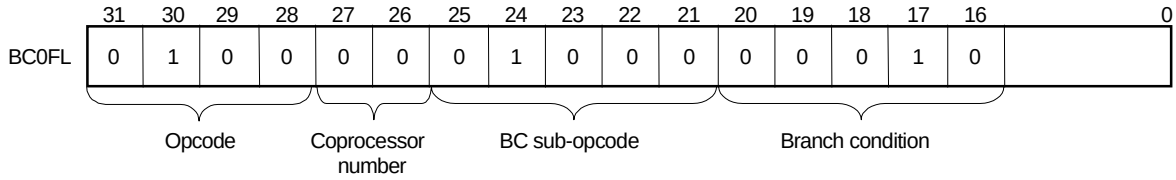
**Exceptions:**

Coprocessor unusable exception

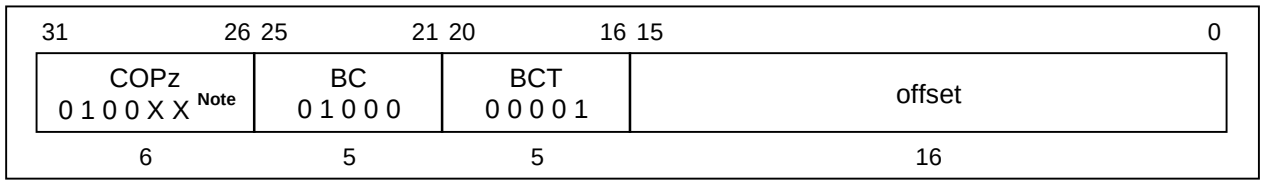
**Note** See the opcode table below, or **A.6 CPU Instruction Opcode Bit Encoding**.

# BC0FL Branch On Coprocessor 0 False Likely (2/2) BC0FL

**Opcode Table:**



# BC0T Branch On Coprocessor 0 True BC0T




**Format:**

BC0T offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. If the contents of CP0's condition signal (CpCond) is true, then the program branches to the target address, with a delay of one instruction.

Because the condition line is sampled during the previous instruction, there must be at least one instruction between this instruction and a coprocessor instruction that changes the condition line.

**Operation:**

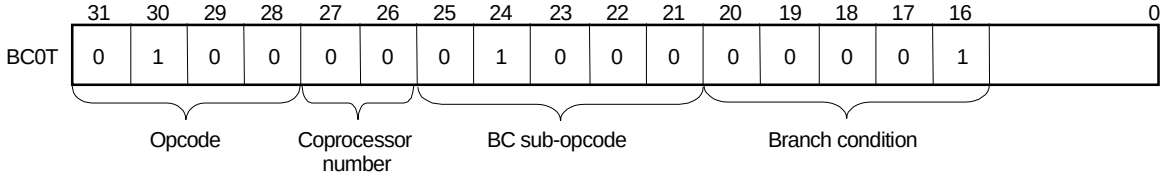
```
32   T-1:  condition ← SR18
      T:   target ← (offset15)^14 || offset || 0^2
      T+1: if condition then
                PC ← PC + target
           endif

64   T-1:  condition ← SR18
      T:   target ← (offset15)^46 || offset || 0^2
      T+1: if condition then
                PC ← PC + target
           endif
```

**Exceptions:**

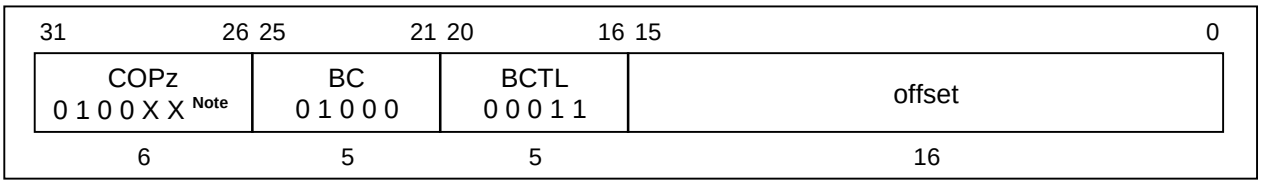
Coprocessor unusable exception

**Note** See the opcode table below, or **A.6 CPU Instruction Opcode Bit Encoding**.

**Opcode Table:**



# BC0TL Branch On Coprocessor 0 True Likely (1/2) BC0TL




**Format:**

BC0TL offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. If the contents of CP0's condition (CpCond) line, as sampled during the previous instruction, is true, the target address is branched to with a delay of one instruction.

If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

Because the condition line is sampled during the previous instruction, there must be at least one instruction between this instruction and a coprocessor instruction that changes the condition line.

**Operation:**

```
32  T-1:  condition ← SR18
      T:  target ← (offset15)^14 || offset || 0^2
    T+1:  if condition then
                PC ← PC + target
          else
                NullifyCurrentInstruction
          endif

64  T-1:  condition ← SR18
      T:  target ← (offset15)^46 || offset || 0^2
    T+1:  if condition then
                PC ← PC + target
          else
                NullifyCurrentInstruction
          endif
```

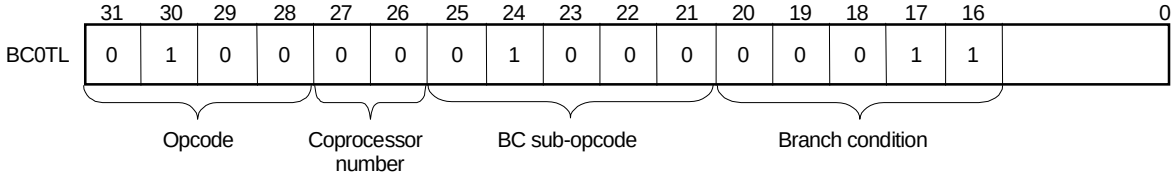
**Exceptions:**

Coprocessor unusable exception

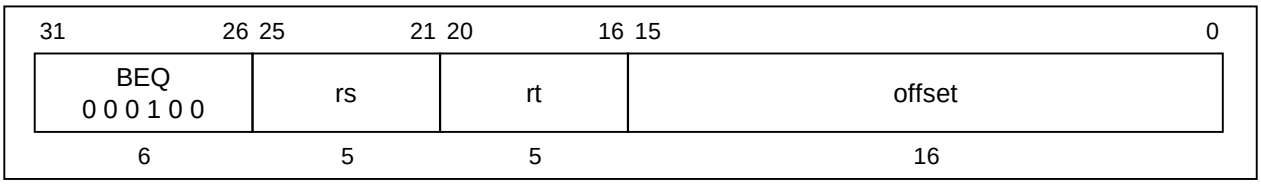
**Note** See the opcode table below, or **A.6 CPU Instruction Opcode Bit Encoding**.

# BC0TL Branch On Coprocessor 0 True Likely (2/2) BC0TL

**Opcode Table:**



# BEQ — Branch On Equal — BEQ




**Format:**

BEQ rs, rt, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. The contents of general register *rs* and the contents of general register *rt* are compared. If the two registers are equal, then the program branches to the target address, with a delay of one instruction.

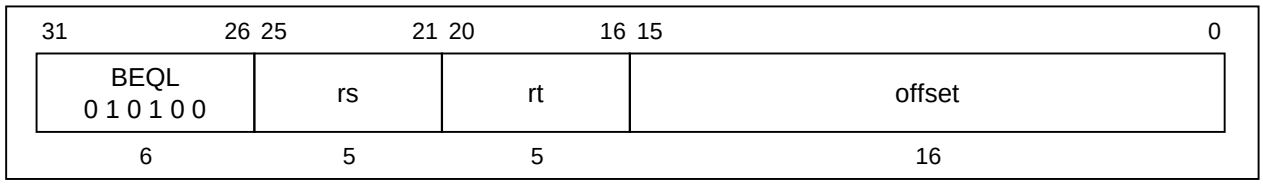
**Operation:**

```
32    T:    target ← (offset15)^14 || offset || 0^2
            condition ← (GPR [rs] = GPR [rt])
      T+1:  if condition then
                  PC ← PC + target
            endif

64    T:    target ← (offset15)^46 || offset || 0^2
            condition ← (GPR [rs] = GPR [rt])
      T+1:  if condition then
                  PC ← PC + target
            endif
```

**Exceptions:**

None

# BEQL Branch On Equal Likely BEQL




**Format:**

BEQL rs, rt, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit offset, shifted left two bits and sign-extended. The contents of general register *rs* and the contents of general register *rt* are compared. If the two registers are equal, the target address is branched to, with a delay of one instruction. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

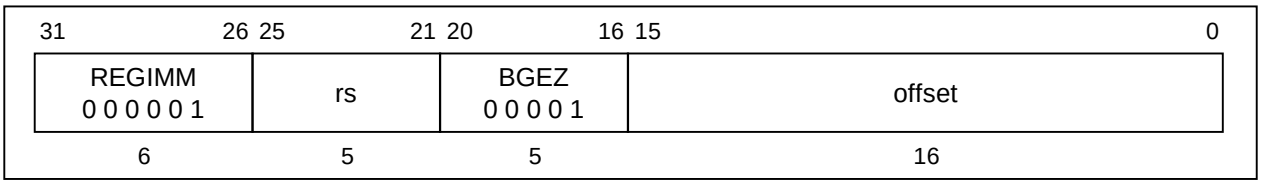
**Operation:**

```
32    T:    target ← (offset15)^14 || offset || 0^2
            condition ← (GPR [rs] = GPR [rt])
      T+1:  if condition then
                  PC ← PC + target
            else
                  NullifyCurrentInstruction
            endif

64    T:    target ← (offset15)^46 || offset || 0^2
            condition ← (GPR [rs] = GPR [rt])
      T+1:  if condition then
                  PC ← PC + target
            else
                  NullifyCurrentInstruction
            endif
```

**Exceptions:**

None

# BGEZ Branch On Greater Than Or Equal To Zero BGEZ




**Format:**

BGEZ rs, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. If the contents of general register *rs* are zero or greater when compared to zero, then the program branches to the target address, with a delay of one instruction.

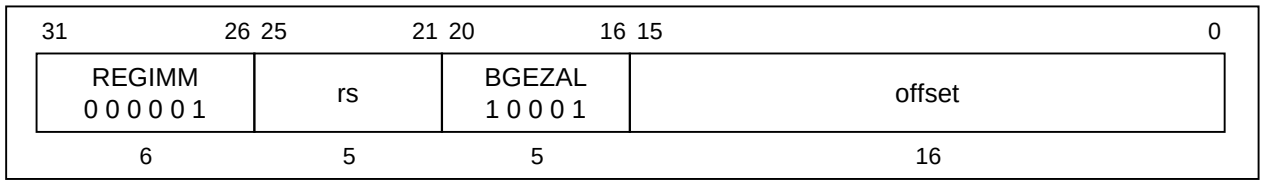
**Operation:**

```
32   T:   target ← (offset15)^14 || offset || 0^2
          condition ← (GPR [rs]31 = 0)
     T+1: if condition then
               PC ← PC + target
          endif

64   T:   target ← (offset15)^46 || offset || 0^2
          condition ← (GPR [rs]63 = 0)
     T+1: if condition then
               PC ← PC + target
          endif
```

**Exceptions:**

None

# BGEZAL Branch On Greater Than Or Equal To Zero And Link BGEZAL




**Format:**

BGEZAL rs, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset,* shifted left two bits and sign-extended. Unconditionally, the address of the instruction after the delay slot is placed in the link register, *r31*. If the contents of general register *rs* are zero or greater when compared to zero, then the program branches to the target address, with a delay of one instruction.

General register *rs* may not be general register *r31*, because such an instruction is not restartable. An attempt to execute this instruction is not trapped, however.

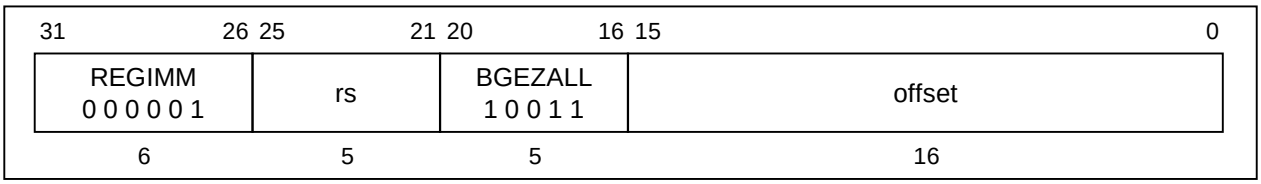
**Operation:**

```
32   T:   target ← (offset15)^14 || offset || 0^2
          condition ← (GPR [rs]31 = 0)
          GPR [31] ← PC + 8
     T+1: if condition then
               PC ← PC + target
          endif

64   T:   target ← (offset15)^46 || offset || 0^2
          condition ← (GPR [rs]63 = 0)
          GPR [31] ← PC + 8
     T+1: if condition then
               PC ← PC + target
          endif
```

**Exceptions:**

None

# BGEZALL Branch On Greater Than Or Equal To Zero And Link Likely BGEZALL




**Format:**

BGEZALL rs, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset,* shifted left two bits and sign-extended. Unconditionally, the address of the instruction after the delay slot is placed in the link register, *r31*. If the contents of general register *rs* are zero or greater when compared to zero, then the program branches to the target address, with a delay of one instruction. General register *r31* should not be specified as general register *rs*. If register *r31* is specified, restarting may be impossible due to the destruction of *rs* contents caused by storing a link address. Even such instructions are executed, an exception does not result.

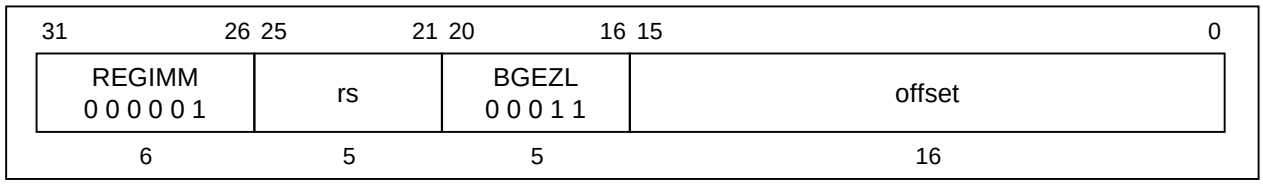
**Operation:**

```
32    T:    target ← (offset15)^14 || offset || 0^2
            condition ← (GPR [rs]31 = 0)
            GPR [31] ← PC + 8
      T+1:  if condition then
                  PC ← PC + target
            else
                  NullifyCurrentInstruction
            endif

64    T:    target ← (offset15)^46 || offset || 0^2
            condition ← (GPR [rs]63 = 0)
            GPR [31] ← PC + 8
      T+1:  if condition then
                  PC ← PC + target
            else
                  NullifyCurrentInstruction
            endif
```

**Exceptions:**

None

# BGEZL Branch On Greater Than Or Equal To Zero Likely BGEZL




**Format:**

BGEZL rs, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. If the contents of general register *rs* are zero or greater when compared to zero, then the program branches to the target address, with a delay of one instruction. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

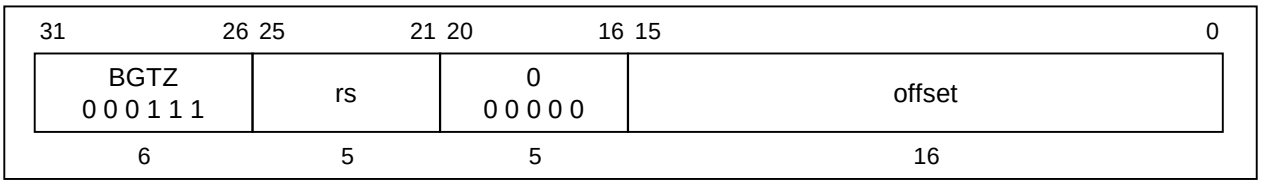
**Operation:**

```
32    T:    target ← (offset15)^14 || offset || 0^2
            condition ← (GPR [rs]31 = 0)
      T+1:  if condition then
                  PC ← PC + target
            else
                  NullifyCurrentInstruction
            endif

64    T:    target ← (offset15)^46 || offset || 0^2
            condition ← (GPR [rs]63 = 0)
      T+1:  if condition then
                  PC ← PC + target
            else
                  NullifyCurrentInstruction
            endif
```

**Exceptions:**

None

# BGTZ Branch On Greater Than Zero BGTZ




**Format:**

BGTZ rs, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. If the contents of general register *rs* are zero or greater when compared to zero, then the program branches to the target address, with a delay of one instruction.

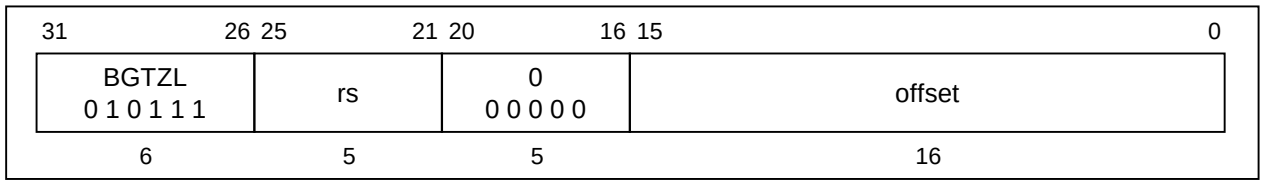
**Operation:**

```
32   T:    target ← (offset15)^14 || offset || 0^2
           condition ← (GPR [rs] 31 = 0) and (GPR [rs] ≠ 0^32)
      T+1: if condition then
                PC ← PC + target
           endif

64   T:    target ← (offset15)^46 || offset || 0^2
           condition ← (GPR [rs]63 = 0) and (GPR [rs] ≠ 0^64)
      T+1: if condition then
                PC ← PC + target
           endif
```

**Exceptions:**

None

# BGTZL Branch On Greater Than Zero Likely BGTZL




**Format:**

BGTZL rs, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. The contents of general register *rs* are compared to zero. If the contents of general register *rs* are greater than zero, then the program branches to the target address, with a delay of one instruction. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

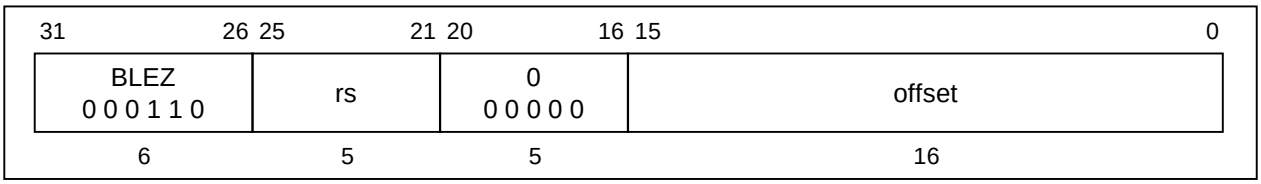
**Operation:**

```
32   T:   target ← (offset15)^14 || offset || 0^2
          condition ← (GPR [rs]31 = 0) and (GPR [rs] ≠ 0^32)
     T+1: if condition then
               PC ← PC + target
          else
               NullifyCurrentInstruction
          endif

64   T:   target ← (offset15)^46 || offset || 0^2
          condition ← (GPR [rs]63 = 0) and (GPR [rs] ≠ 0^64)
     T+1: if condition then
               PC ← PC + target
          else
               NullifyCurrentInstruction
          endif
```

**Exceptions:**

None

# BLEZ Branch On Less Than Or Equal To Zero BLEZ




**Format:**

BLEZ rs, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. The contents of general register *rs* are compared to zero. If the contents of general register *rs* are zero or smaller than zero, then the program branches to the target address, with a delay of one instruction.

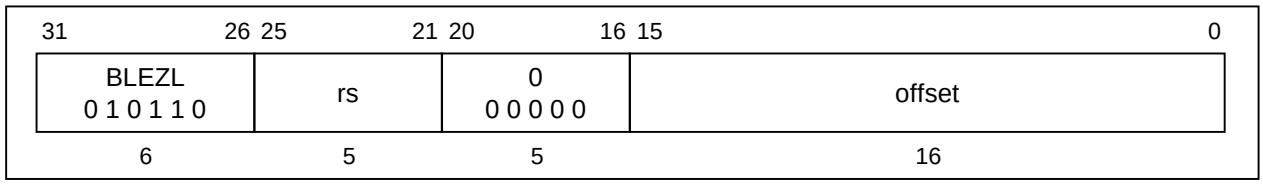
**Operation:**

```
32   T:    target ← (offset15)^14 || offset || 0^2
           condition ← (GPR [rs]31 = 1) or (GPR [rs] = 0^32)
      T+1: if condition then
                PC ← PC + target
           endif

64   T:    target ← (offset15)^46 || offset || 0^2
           condition ← (GPR [rs]63 = 1) or (GPR [rs] = 0^64)
      T+1: if condition then
                PC ← PC + target
           endif
```

**Exceptions:**

None

# BLEZL Branch On Less Than Or Equal To Zero Likely BLEZL




**Format:**

BLEZL rs, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. The contents of general register *rs* is compared to zero. If the contents of general register *rs* are zero or smaller than zero, then the program branches to the target address, with a delay of one instruction.

If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

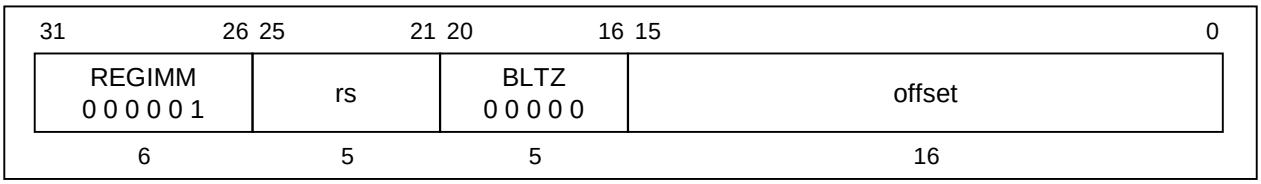
**Operation:**

```
32   T:    target ← (offset15)^14 || offset || 0^2
           condition ← (GPR [rs]31 = 1) or (GPR [rs] = 0^32)
      T+1: if condition then
                PC ← PC + target
           else
                NullifyCurrentInstruction
           endif

64   T:    target ← (offset15)^46 || offset || 0^2
           condition ← (GPR [rs]63 = 1) or (GPR [rs] = 0^64)
      T+1: if condition then
                PC ← PC + target
           else
                NullifyCurrentInstruction
           endif
```

**Exceptions:**

None

# BLTZ Branch On Less Than Zero BLTZ




**Format:**

BLTZ rs, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. If the contents of general register *rs* are smaller than zero, then the program branches to the target address, with a delay of one instruction.

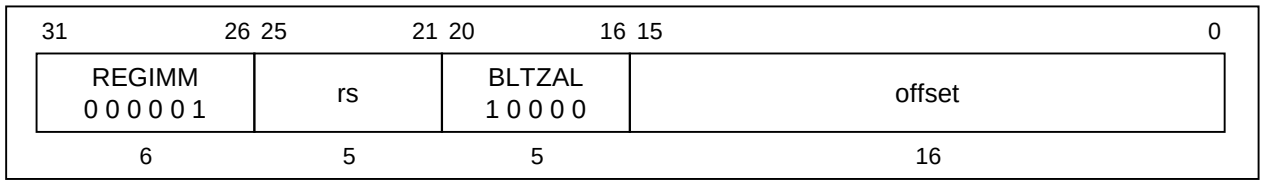
**Operation:**

```
32  T:    target ← (offset15)^14 || offset || 0^2
          condition ← (GPR [rs]31 = 1)
      T+1: if condition then
               PC ← PC + target
          endif

64  T:    target ← (offset15)^46 || offset || 0^2
          condition ← (GPR [rs]63 = 1)
      T+1: if condition then
               PC ← PC + target
          endif
```

**Exceptions:**

None

# BLTZAL Branch On Less Than Zero And Link BLTZAL




**Format:**

BLTZAL rs, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset,* shifted left two bits and sign-extended. Unconditionally, the address of the instruction after the delay slot is placed in the link register, *r31*. If the contents of general register *rs* are smaller than zero when compared to zero, then the program branches to the target address, with a delay of one instruction.

General register *r31* should not be specified as general register *rs*. If register *r31* is specified, restarting may be impossible due to the destruction of *rs* contents caused by storing a link address. Even such instructions are executed, an exception does not result.

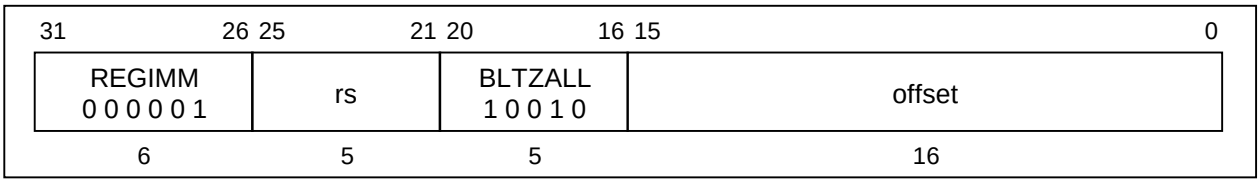
**Operation:**

```
32   T:    target ← (offset15)^14 || offset || 0^2
           condition ← (GPR [rs]31 = 1)
           GPR [31] ← PC + 8
      T+1: if condition then
                PC ← PC + target
           endif

64   T:    target ← (offset15)^46 || offset || 0^2
           condition ← (GPR [rs]63 = 1)
           GPR [31] ← PC + 8
      T+1: if condition then
                PC ← PC + target
           endif
```

**Exceptions:**

None

# BLTZALL Branch On Less Than Zero And Link Likely BLTZALL




**Format:**

BLTZALL rs, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset,* shifted left two bits and sign-extended. Unconditionally, the address of the instruction after the delay slot is placed in the link register, *r31*. If the contents of general register *rs* are smaller than zero when compared to zero, then the program branches to the target address, with a delay of one instruction.

General register *r31* should not be specified as general register *rs*. If register *r31* is specified, restarting may be impossible due to the destruction of *rs* contents caused by storing a link address. Even such instructions are executed, an exception does not result.

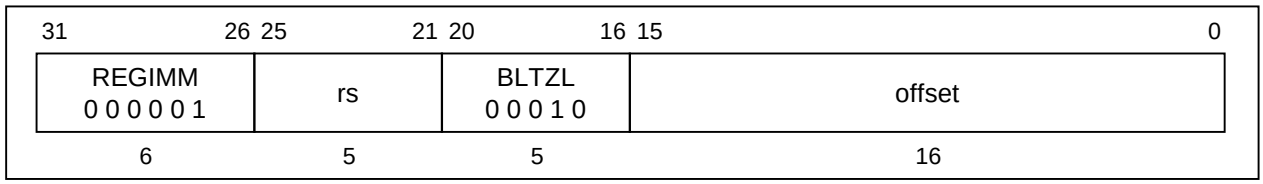
**Operation:**

```
32    T:    target ← (offset15)^14 || offset || 0^2
            condition ← (GPR [rs]31 = 1)
            GPR [31] ← PC + 8
      T+1:  if condition then
                  PC ← PC + target
            else
                  NullifyCurrentInstruction
            endif

64    T:    target ← (offset15)^46 || offset || 0^2
            condition ← (GPR [rs]63 = 1)
            GPR [31] ← PC + 8
      T+1:  if condition then
                  PC ← PC + target
            else
                  NullifyCurrentInstruction
            endif
```

**Exceptions:**

None

# BLTZL Branch On Less Than Zero Likely BLTZL




**Format:**

BLTZ rs, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. If the contents of general register *rs* are smaller than zero when compared to zero, then the program branches to the target address, with a delay of one instruction. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

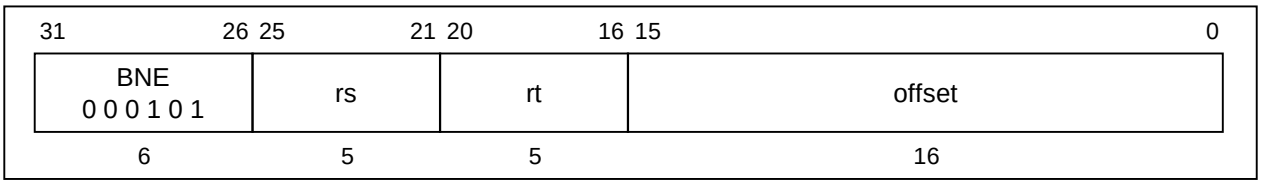
**Operation:**

```
32    T:    target ← (offset15)^14 || offset || 0^2
            condition ← (GPR [rs]31 = 1)
      T+1:  if condition then
                  PC ← PC + target
            else
                  NullifyCurrentInstruction
            endif

64    T:    target ← (offset15)^46 || offset || 0^2
            condition ← (GPR [rs]63 = 1)
      T+1:  if condition then
                  PC ← PC + target
            else
                  NullifyCurrentInstruction
            endif
```

**Exceptions:**

None

# BNE Branch On Not Equal BNE




**Format:**

BNE rs, rt, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset,* shifted left two bits and sign-extended. The contents of general register *rs* and the contents of general register *rt* are compared. If the two registers are not equal, then the program branches to the target address, with a delay of one instruction.

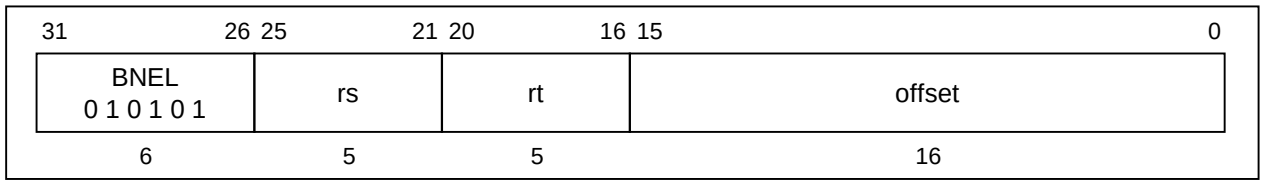
**Operation:**

```
32    T:    target ← (offset15)^14 || offset || 0^2
            condition ← (GPR [rs] ≠ GPR [rt])
      T+1:  if condition then
                  PC ← PC + target
            endif

64    T:    target ← (offset15)^46 || offset || 0^2
            condition ← (GPR [rs] ≠ GPR [rt])
      T+1:  if condition then
                  PC ← PC + target
            endif
```

**Exceptions:**

None

# BNEL Branch On Not Equal Likely BNEL




**Format:**

BNEL rs, rt, offset

**Description:**

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset,* shifted left two bits and sign-extended. The contents of general register *rs* and the contents of general register *rt* are compared. If the two registers are not equal, then the program branches to the target address, with a delay of one instruction.

If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

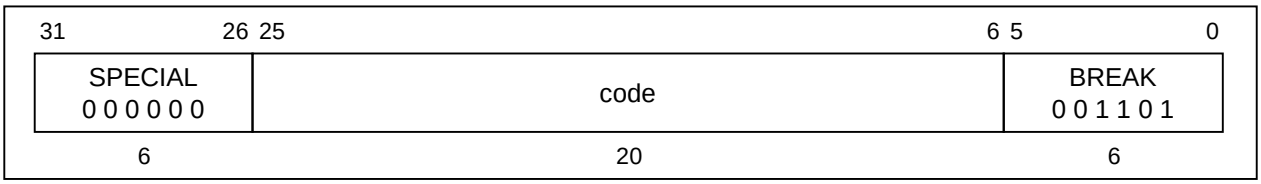
**Operation:**

```
32    T:    target ← (offset15)^14 || offset || 0^2
            condition ← (GPR [rs] ≠ GPR [rt])
      T+1:  if condition then
                  PC ← PC + target
            else
                  NullifyCurrentInstruction
            endif

64    T:    target ← (offset15)^46 || offset || 0^2
            condition ← (GPR [rs] ≠ GPR [rt])
      T+1:  if condition then
                  PC ← PC + target
            else
                  NullifyCurrentInstruction
            endif
```

**Exceptions:**

None

# BREAK — Breakpoint — BREAK




**Format:**

BREAK

**Description:**

A breakpoint trap occurs, immediately and unconditionally transferring control to the exception handler.

The code field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

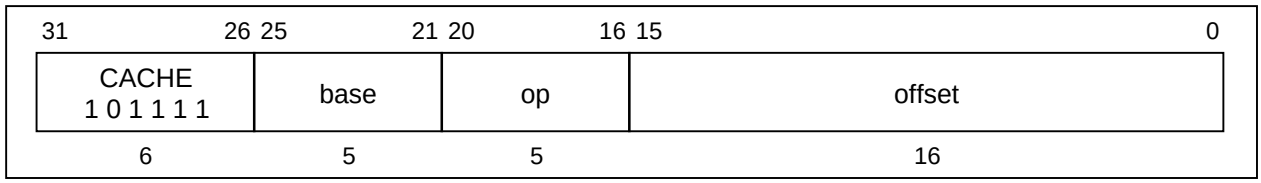
**Operation:**

```
32, 64 T:    BreakpointException
```

**Exceptions:**

Breakpoint exception

# CACHE — Cache (1/4) — CACHE




**Format:**

CACHE op, offset (base)

**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The virtual address is translated to a physical address using the TLB, and the 5-bit sub-opcode specifies a cache operation for that address.

If CP0 is not usable (User or Supervisor mode) and the CP0 enable bit in the Status register is clear, a coprocessor unusable exception is taken. The operation of this instruction on any operation/cache combination not listed below, or on a secondary cache that is not incorporated in VR4120A CPU, is undefined. The operation of this instruction on uncached addresses is also undefined.

The Index operation uses part of the virtual address to specify a cache block.

For a primary cache of $2^{CACHEBITS}$ bytes with $2^{LINEBITS}$ bytes per tag, vAddrCACHEBITS...LINEBITS specifies the block. Index_Load_Tag also uses vAddrLINEBITS...3 to select the doubleword for reading parity. When the *CE* bit of the Status register is set, Fill Cache op uses the PErr register to store parity values into the cache.

The Hit operation accesses the specified cache as normal data references, and performs the specified operation if the cache block contains valid data with the specified physical address (a hit). If the cache block is invalid or contains a different address (a miss), no operation is performed.

# CACHE Cache (2/4) CACHE

Write back from a cache goes to main memory.
The main memory address to be written is specified by the cache tag and not the physical address translated using TLB.
TLB Refill and TLB Invalid exceptions can occur on any operation. For Index operations[Note] for addresses in the unmapped areas, unmapped addresses may be used to avoid TLB exceptions. Index operations never cause a TLB Modified exception. Bits 17 and 16 of the instruction code specify the cache for which the operation is to be performed as follows.

| Code | Name | Cache |
|---|---|---|
| 0 | I | Instruction cache |
| 1 | D | Data cache |
| 2 | — | Reserved |
| 3 | — | Reserved |

**Note** Physical addresses here are used to index the cache, and they do not need to match the cache tag.

Bits 20 to 18 of this instruction specify the contents of cache operaiton. Details are provided from the next page.

# CACHE Cache (3/4) CACHE

| Code | Cache | Name | Operation |
|---|---|---|---|
| 0 | I | Index_Invalidate | Set the cache state of the cache block to Invalid. |
| 0 | D | Index_Write_ Back_Invalidate | Examine the cache state and W bit of the primary data cache block at the index specified by the virtual address. If the state is not Invalid and the W bit is set, then write back the block to memory. The address to write is taken from the primary cache tag. Set cache state of primary cache block to Invalid. |
| 1 | I, D | Index_Load_Tag | Read the tag for the cache block at the specified index and place it into the TagLo CP0 registers, ignoring parity errors. Also load the data parity bits into the ECC register. |
| 2 | I, D | Index_Store_ Tag | Write the tag for the cache block at the specified index from the TagLo and TagHi CP0 registers. |
| 3 | D | Create_Dirty_ Exclusive | This operation is used to avoid loading data needlessly from memory when writing new contents into an entire cache block. If the cache block does not contain the specified address, and the block is dirty, write it back to the memory. In all cases, set the cache state to Dirty. |
| 4 | I, D | Hit_Invalidate | If the cache block contains the specified address, mark the cache block invalid. |
| 5 | D | Hit_Write_Back Invalidate | If the cache block contains the specified address, write back the data if it is dirty, and mark the cache block invalid. |
| 5 | I | Fill | Fill the primary instruction cache block from memory. If the CE bit of the Status register is set, the contents of the ECC register is used instead of the computed parity bits for addressed doubleword when written to the instruction cache. |
| 6 | D | Hit_Write_Back | If the cache block contains the specified address, and the W bit is set, write back the data to memory and clear the W bit. |
| 6 | I | Hit_Write_Back | If the cache block contains the specified address, write back the data unconditionally. |

# CACHE — Cache (4/4) — CACHE

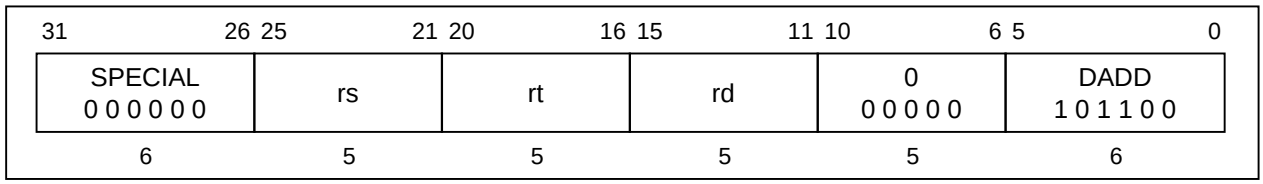
**Operation:**

```
32, 64 T:   vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            CacheOp (op, vAddr, pAddr)
```

**Exceptions:**

Coprocessor unusable exception
TLB Refill exception
TLB Invalid exception
Bus Error exception
Address Error exception
Cache Error exception

# DADD Doubleword Add DADD




**Format:**

DADD rd, rs, rt

**Description:**

The contents of general register *rs* and the contents of general register *rt* are added to form the result. The result is placed into general register *rd*.

An overflow exception occurs if the carries out of bits 62 and 63 differ (2's complement overflow). The destination register *rd* is not modified when an integer overflow exception occurs.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

**Operation:**

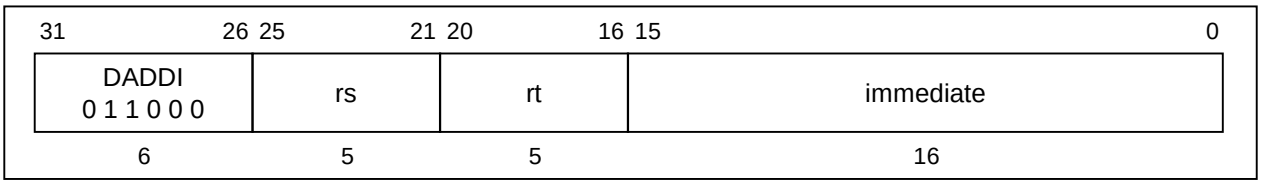
```
64   T:   GPR [rd] ← GPR [rs] + GPR [rt]
```

**Exceptions:**

Integer overflow exception

Reserved instruction exception (32-bit user mode/supervisor mode)

# DADDI Doubleword Add Immediate DADDI




**Format:**

DADDI rt, rs, immediate

**Description:**

The 16-bit *immediate* is sign-extended and added to the contents of general register *rs* to form the result. The result is placed into general register *rt.*

An overflow exception occurs if carries out of bits 62 and 63 differ (2's complement overflow). The destination register *rt* is not modified when an integer overflow exception occurs.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

**Operation:**

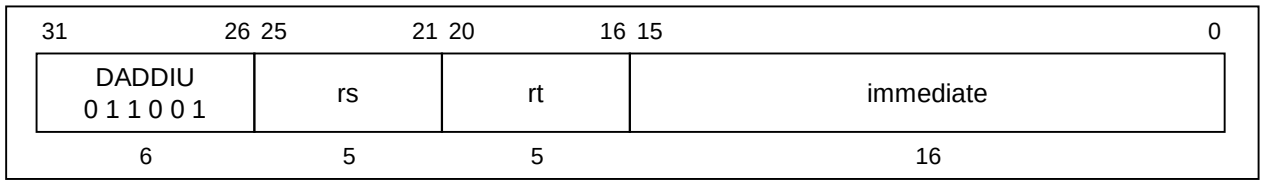
```
64    T:    GPR [rt] ← GPR [rs] + (immediate15)48 || immediate15...0
```

**Exceptions:**

Integer overflow exception

Reserved instruction exception (32-bit user mode/supervisor mode)

# DADDIU Doubleword Add Immediate Unsigned DADDIU




**Format:**

DADDIU rt, rs, immediate

**Description:**

The 16-bit *immediate* is sign-extended and added to the contents of general register *rs* to form the result. The result is placed into general register *rt.* No integer overflow exception occurs under any circumstances.

The only difference between this instruction and the DADDI instruction is that DADDIU never causes an overflow exception.

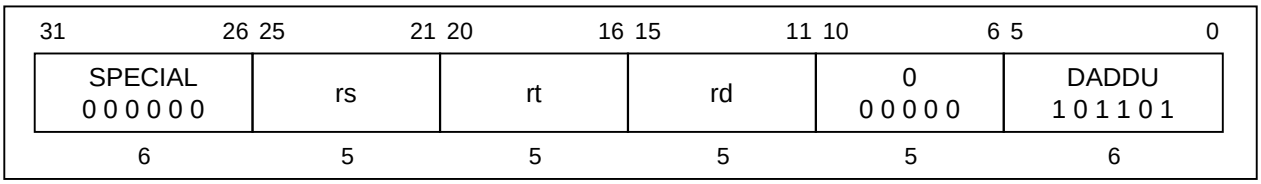
**Operation:**

```
64    T:    GPR [rt] ← GPR [rs] + (immediate15)48 || immediate15...0
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# DADDU Doubleword Add Unsigned DADDU




**Format:**

DADDU rd, rs, rt

**Description:**

The contents of general register *rs* and the contents of general register *rt* are added to form the result. The result is placed into general register *rd*.

No overflow exception occurs under any circumstances.

The only difference between this instruction and the DADD instruction is that DADDU never causes an overflow exception.

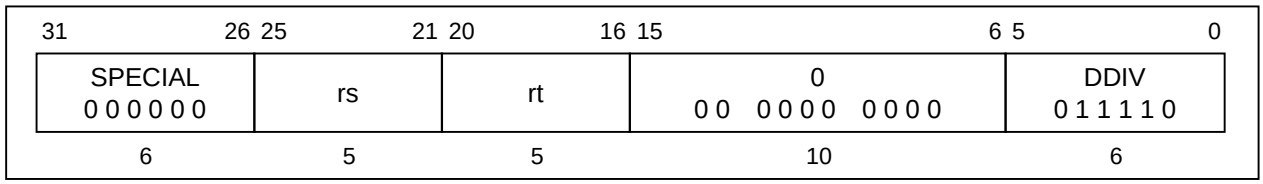
**Operation:**

```
64    T:    GPR [rd] ← GPR [rs] + GPR [rt]
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# DDIV — Doubleword Divide — DDIV




**Format:**

DDIV rs, rt

**Description:**

The contents of general register *rs* are divided by the contents of general register *rt,* treating both operands as 2's complement values. No overflow exception occurs under any circumstances, and the result of this operation is undefined when the divisor is zero.

This instruction is typically followed by additional instructions to check for a zero divisor and for overflow.

When the operation completes, the quotient word of the double result is loaded into special register *LO*, and the remainder word of the double result is loaded into special register *HI*.

If either of the two preceding instructions is MFHI or MFLO, the results of those instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by two or more instructions.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

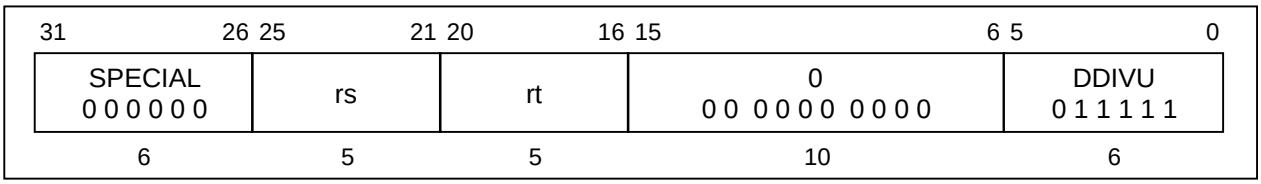
**Operation:**

```
64   T-2:  LO  ← undefined
           HI  ← undefined
      T-1:  LO  ← undefined
           HI  ← undefined
      T:    LO  ← GPR [rs] div GPR [rt]
           HI  ← GPR [rs] mod GPR [rt]
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# DDIVU Doubleword Divide Unsigned DDIVU




**Format:**

DDIVU rs, rt

**Description:**

The contents of general register *rs* are divided by the contents of general register *rt,* treating both operands as unsigned values. No integer overflow exception occurs under any circumstances, and the result of this operation is undefined when the divisor is zero.

This instruction may be followed by additional instructions to check for a zero divisor, inserted by the programmer.

When the operation completes, the quotient word of the double result is loaded into special register *LO*, and the remainder word of the double result is loaded into special register *HI*.

If either of the two preceding instructions is MFHI or MFLO, the results of those instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by two or more instructions.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

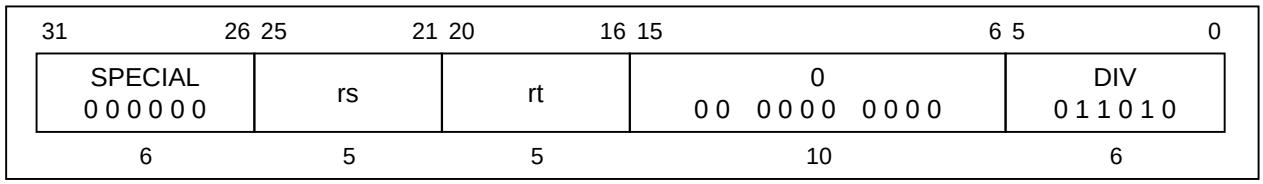
**Operation:**

```
64      T-2:  LO   ← undefined
              HI   ← undefined
        T-1:  LO   ← undefined
              HI   ← undefined
        T:    LO   ←  (0 || GPR [rs]) div (0 || GPR [rt])
              HI   ← (0 || GPR [rs]) mod (0 || GPR [rt])
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# DIV — Divide — DIV




**Format:**

DIV rs, rt

**Description:**

The contents of general register *rs* are divided by the contents of general register *rt,* treating both operands as 2's complement values. No overflow exception occurs under any circumstances, and the result of this operation is undefined when the divisor is zero.

In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

This instruction is typically followed by additional instructions to check for a zero divisor and for overflow.

When the operation completes, the quotient word of the double result is loaded into special register *LO*, and the remainder word of the double result is loaded into special register *HI.*

If either of the two preceding instructions is MFHI or MFLO, the results of those instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by two or more instructions.

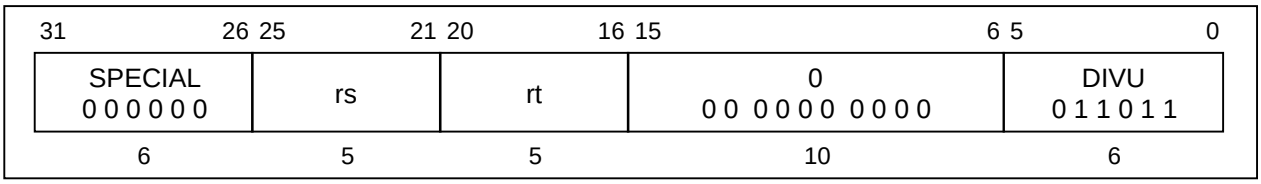
**Operation:**

```
32    T-2:  LO   ← undefined
            HI   ← undefined
      T-1:  LO   ← undefined
            HI   ← undefined
      T:    LO   ← GPR [rs] div GPR [rt]
            HI   ← GPR [rs] mod GPR [rt]

64    T-2:  LO   ← undefined
            HI   ← undefined
      T-1:  LO   ← undefined
            HI   ← undefined
      T:    q    ← GPR [rs]31..0 div GPR [rt]31..0
            r    ← GPR [rs]31..0 mod GPR [rt]31..0
            LO   ← (q31)32 || q31..0
            HI   ← (r31)32 || r31..0
```

**Exceptions:**

None

# DIVU Divide Unsigned DIVU




**Format:**

DIVU rs, rt

**Description:**

The contents of general register *rs* are divided by the contents of general register *rt,* treating both operands as unsigned values. No integer overflow exception occurs under any circumstances, and the result of this operation is undefined when the divisor is zero.

In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

This instruction is typically followed by additional instructions to check for a zero divisor.

When the operation completes, the quotient word of the double result is loaded into special register *LO*, and the remainder word of the double result is loaded into special register *HI*.

If either of the two preceding instructions is MFHI or MFLO, the results of those instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by two or more instructions.

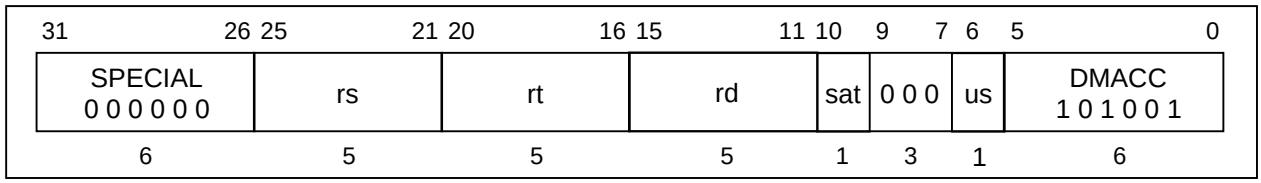
**Operation:**

```
32    T-2:  LO   ← undefined
            HI   ← undefined
      T-1:  LO   ← undefined
            HI   ← undefined
      T:    LO   ← (0 || GPR [rs]) div (0 || GPR [rt])
            HI   ←  0 || GPR [rs]) mod (0 || GPR [rt])

64    T-2:  LO   ← undefined
            HI   ← undefined
      T-1:  LO   ← undefined
            HI   ← undefined
      T:    q    ← (0 || GPR [rs]31..0) div (0 || GPR [rt]31..0)
            r    ← (0 || GPR [rs]31..0) mod (0 || GPR [rt]31..0)
            LO   ← (q31)32 || q31..0
            HI   ← (r31)32 || r31..0
```

**Exceptions:**

None

# DMACC Doubleword Multiply and Accumulate (1/3) DMACC




**Format:**

DMACC rd, rs, rt
DMACCU rd, rs, rt
DMACCS rd, rs, rt
DMACCUS rd, rs, rt

**Description:**

DMACC instruction differs mnemonics by each setting of op codes sat, hi and us as follows.

| Mnemonic | sat | us |
|---|---|---|
| DMACC | 0 | 0 |
| DMACCU | 0 | 1 |
| DMACCS | 1 | 0 |
| DMACCUS | 1 | 1 |

The number of significant bits in the operands of the DMACC instruction differ depending on whether saturation processing is executed (sat = 1) or not executed (sat = 0).

- When saturation processing is executed (sat = 1): DMACCS, DMACCUS instructions
  The contents of general register *rs* is multiplied by the contents of general register *rt*. If both operands are set as "us = 1" (DMACCUS instruction), the contents are handled as 16 bit unsigned data. If they are set as "us = 0" (DMACCS instruction), the contents are handled as 16 bit signed integers. Sign/zero expansion by software is required for any bits exceeding 16 bits in the operands.
  The product of this multiply operation is added to the value in the LO special register. If us = 1, this add operation handles the values being added as 32 bit unsigned data. If us = 0, the values are handled as 32 bit signed integers. Sign/zero expansion by software is required for any bits exceeding 32 bits in the LO special register.
  After saturation processing to 32 bits has been performed (see the table below), the sum from this add operation is loaded to the LO special register. When hi = 1, data that is the same as the data loaded to the HI special register is also loaded to the rd general register. When hi = 0, data that is the same as the data loaded to the LO special register is also loaded to the rd general register. Overflow exceptions do not occur.

# DMACC Doubleword Multiply and Accumulate (2/3) DMACC

- When saturation processing is not executed (sat = 0): DMACC, DMACCU instructions
  The contents of general register *rs* is multiplied by the contents of general register *rt*. If both operands are set as "us = 1" (DMACCU instruction), the contents are handled as 32 bit unsigned data. If they are set as "us = 0" (DMACC instruction), the contents are handled as 32 bit signed integers. Sign/zero expansion by software is required for any bits exceeding 32 bits in the operands.
  The product of this multiply operation is added to the value in the LO special register. If us = 1, this add operation handles the values being added as 64 bit unsigned data. If us = 0, the values are handled as 64 bit signed integers.
  The sum from this add operation is loaded to the LO special register. When hi = 1, data that is the same as the data loaded to the HI special register is also loaded to the rd general register. When hi = 0, data that is the same as the data loaded to the LO special register is also loaded to the rd general register. Overflow exceptions do not occur.

These operations are defined for 64 bit mode and 32 bit kernel mode. A reserved instruction exception occurs if one of these instructions is executed during 32 bit user/supervisor mode.
The correspondence of us and sat settings and values stored during saturation processing is shown below, along with the hazard cycles required between execution of the instruction for manipulating the HI and LO registers and execution of the DMACC instruction.

**Values Stored during Saturation Processing**

| us | sat | Overflow | Underflow |
|---|---|---|---|
| 0 | 0 | Store calculation result as is | Store calculation result as is |
| 1 | 0 | Store calculation result as is | Store calculation result as is |
| 0 | 1 | 0x0000 0000 7FFF FFFF | 0xFFFF FFFF 8000 0000 |
| 1 | 1 | 0xFFFF FFFF FFFF FFFF | None |

**Hazard Cycle Counts**

| Instruction | Cycle count |
|---|---|
| MULT, MULTU | 1 |
| DMULT, DMULTU | 3 |
| DIV, DIVU | 36 |
| DDIV, DDIVU | 68 |
| MFHI, MFLO | 2 |
| MTHI, MTLO | 0 |
| MACC | 0 |
| DMACC | 0 |

# DMACC Doubleword Multiply and Accumulate (3/3) DMACC

**Operation:**

64, sat=0, us=0 (DMACC instruction)

```
T:    temp1 ← ((GPR[rs]31)^32 || GPR [rs]) * ((GPR[rt]31)^32 || GPR [rt])
      temp2 ← temp1 + LO
      LO ← temp2
      GPR[rd] ← LO
```

64, sat=0, us=1 (DMACCU instruction)

```
T:    temp1 ← (0^32 || GPR [rs]) * (0^32 || GPR [rt])
      temp2 ← temp1 + LO
      LO ← temp2
      GPR[rd] ← LO
```

64, sat=1, us=0 (DMACCS instruction)

```
T:    temp1 ← ((GPR[rs]31)^32 || GPR [rs]) * ((GPR[rt]31)^32 || GPR [rt])
      temp2 ← saturation(temp1 + LO)
      LO ← temp2
      GPR[rd] ← LO
```

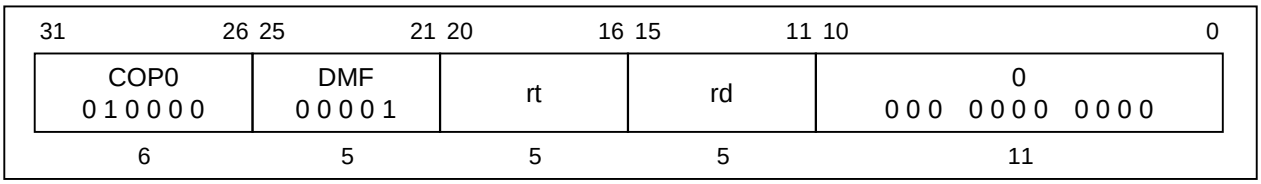
64, sat=1, us=1 (DMACCUS instruction)

```
T:    temp1 ← (0^32 || GPR [rs]) * (0^32 || GPR [rt])
      temp2 ← saturation(temp1 + LO)
      LO ← temp2
      GPR[rd] ← LO
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# DMFC0 Doubleword Move From System Control Coprocessor DMFC0




**Format:**

DMFC0 rt, rd

**Description:**

The contents of coprocessor register *rd* of the CP0 are loaded into general register *rt.*

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception. All 64-bits of the general register destination are written from the coprocessor register source. The operation of DMFC0 on a 32-bit coprocessor 0 register is undefined.

**Operation:**

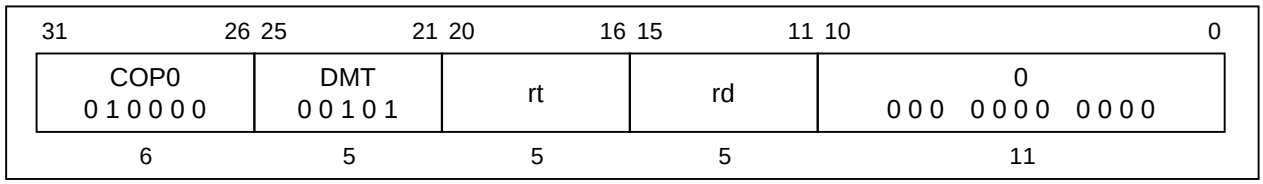
```
64    T:    data ← CPR [0, rd]
      T+1:  GPR [rt] ← data
```

**Exceptions:**

Coprocessor unusable exception (user mode and supervisor mode if CP0 not enabled)

Reserved instruction exception (32-bit user mode/supervisor mode)

# DMTC0 Doubleword Move To System Control Coprocessor DMTC0




**Format:**

DMTC0 rt, rd

**Description:**

The contents of general register *rt* are loaded into coprocessor register *rd* of the CP0.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

All 64-bits of the coprocessor 0 register are written from the general register source. The operation of DMTC0 on a 32-bit coprocessor 0 register is undefined.

Because the state of the virtual address translation system may be altered by this instruction, the operation of load instructions, store instructions, and TLB operations immediately prior to and after this instruction are undefined.

**Operation:**

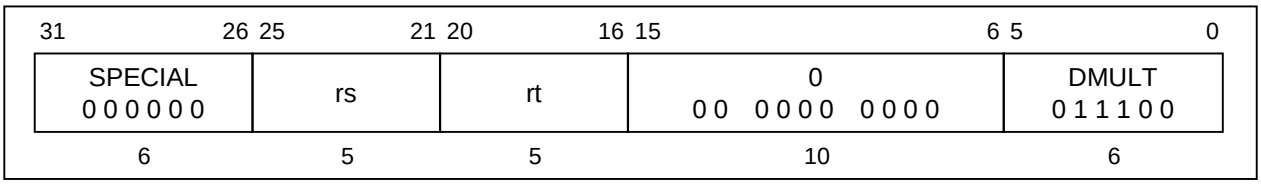
```
64   T:    data ← GPR [rt]
     T+1:  CPR [0, rd] ← data
```

**Exceptions:**

Coprocessor unusable exception (In 64-bit/32-bit user and supervisor mode if CP0 not enabled)

Reserved instruction exception (32-bit user mode/supervisor mode)

# DMULT Doubleword Multiply DMULT




**Format:**

DMULT rs, rt

**Description:**

The contents of general registers *rs* and *rt* are multiplied, treating both operands as 2's complement values. No integer overflow exception occurs under any circumstances.

When the operation completes, the low-order word of the double result is loaded into special register *LO*, and the high-order word of the double result is loaded into special register *HI*.

If either of the two preceding instructions is MFHI or MFLO, the results of these instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by a minimum of two other instructions.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

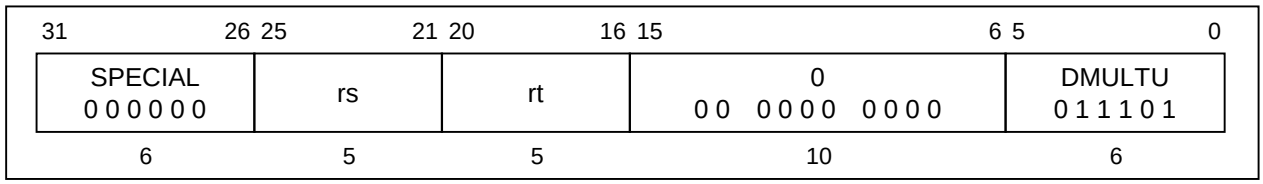
**Operation:**

```
64    T-2:  LO  ← undefined
            HI  ← undefined
      T-1:  LO  ← undefined
            HI  ← undefined
      T:    t   ← GPR [rs] * GPR [rt]
            LO  ← t63..0
            HI  ← t127..64
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# DMULTU Doubleword Multiply Unsigned DMULTU




**Format:**

DMULTU rs, rt

**Description:**

The contents of general register *rs* and the contents of general register *rt* are multiplied, treating both operands as unsigned values. No overflow exception occurs under any circumstances.

When the operation completes, the low-order word of the double result is loaded into special register *LO*, and the high-order word of the double result is loaded into special register *HI*.

If either of the two preceding instructions is MFHI or MFLO, the results of these instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by a minimum of two instructions.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

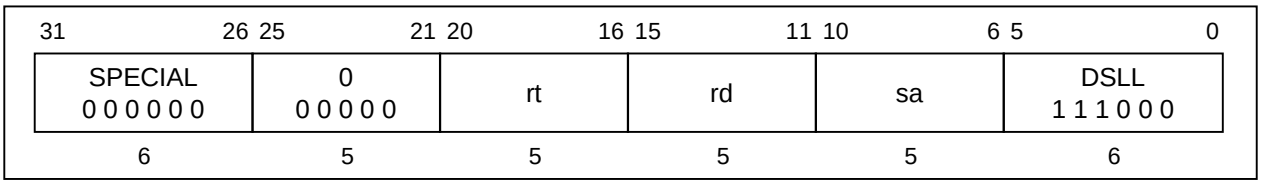
**Operation:**

```
64   T-2:  LO  ← undefined
           HI  ← undefined
      T-1:  LO  ← undefined
           HI  ← undefined
      T:    t   ← (0 || GPR [rs]) * (0 || GPR [rt])
           LO  ← t63..0
           HI  ← t127..64
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# DSLL Doubleword Shift Left Logical DSLL




**Format:**

DSLL rd, rt, sa

**Description:**

The contents of general register *rt* are shifted left by *sa* bits, inserting zeros into the low-order bits. The result is placed in register *rd.*

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

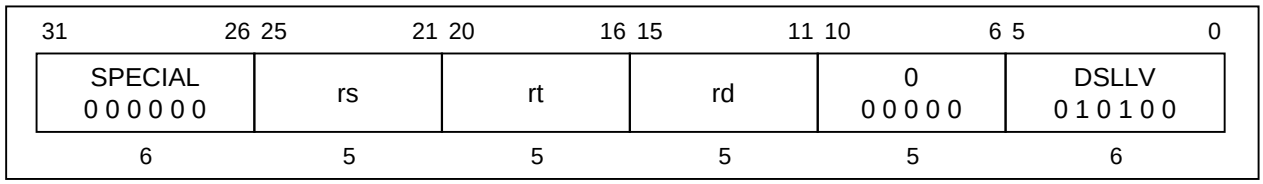
**Operation:**

```
64    T:    s ← 0 || sa
            GPR [rd] ← GPR [rt] (63 - s)..0 || 0^s
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# DSLLV Doubleword Shift Left Logical Variable DSLLV




**Format:**

DSLLV rd, rt, rs

**Description:**

The contents of general register *rt* are shifted left by the number of bits specified by the low-order six bits contained in general register *rs*, inserting zeros into the low-order bits. The result is placed in register *rd*.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

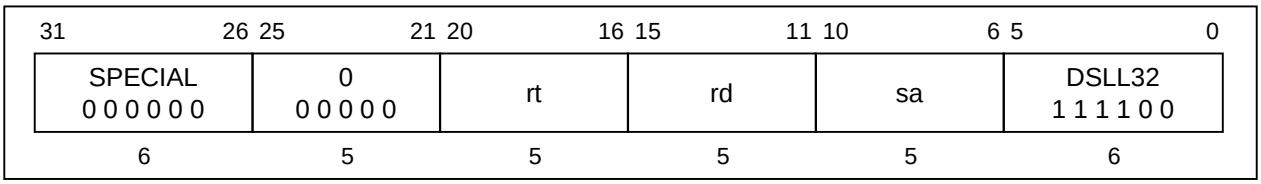
**Operation:**

```
64    T:    s ← GPR [rs]5..0
            GPR [rd] ← GPR [rt](63 - s)..0 || 0^s
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# DSLL32 Doubleword Shift Left Logical + 32 DSLL32




**Format:**

DSLL32 rd, rt, sa

**Description:**

The contents of general register *rt* are shifted left by *32 + sa* bits, inserting zeros into the low-order bits. The result is placed in register *rd*.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

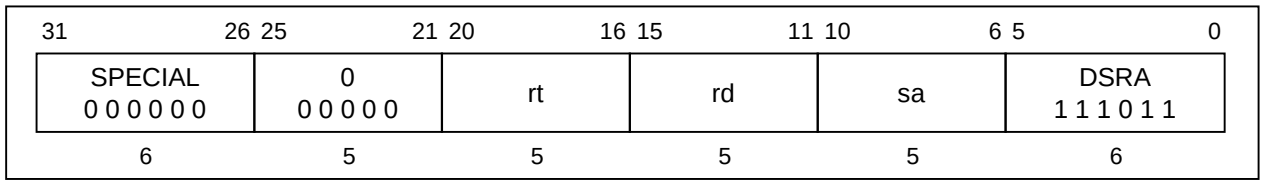
**Operation:**

```
64    T:    s ← 1 || sa
            GPR [rd] ← GPR [rt](63 - s)..0 || 0^s
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# DSRA Doubleword Shift Right Arithmetic DSRA




**Format:**

DSRA rd, rt, sa

**Description:**

The contents of general register *rt* are shifted right by *sa* bits, sign-extending the high-order bits. The result is placed in register *rd*.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

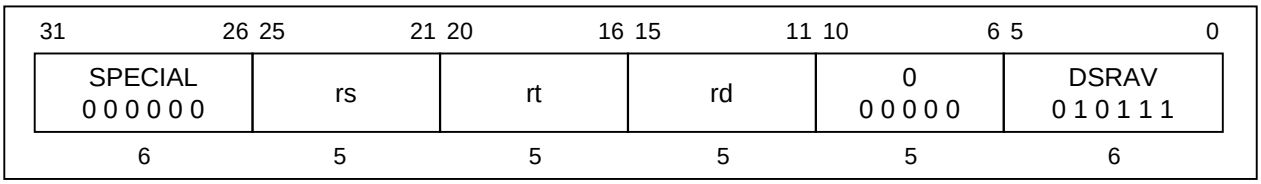
**Operation:**

```
64    T:    s ← 0 || sa
            GPR [rd] ← (GPR [rt] 63)^s || GPR [rt] 63..s
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# DSRAV Doubleword Shift Right Arithmetic Variable DSRAV




**Format:**

DSRAV rd, rt, rs

**Description:**

The contents of general register *rt* are shifted right by the number of bits specified by the low-order six bits of general register *rs*, sign-extending the high-order bits. The result is placed in register *rd*.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

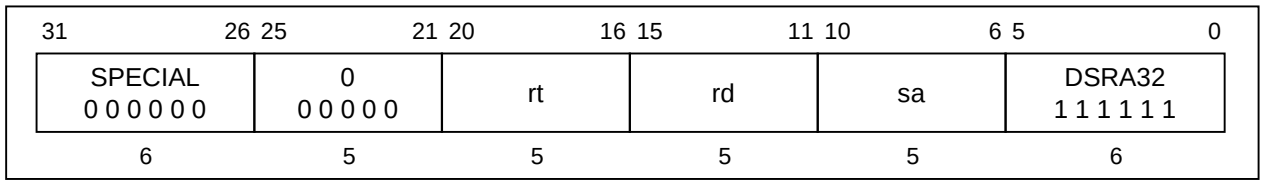
**Operation:**

```
64    T:    s ← GPR [rs]5..0
            GPR [rd] ← (GPR [rt] 63)^s || GPR [rt] 63..s
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# DSRA32 Doubleword Shift Right Arithmetic + 32 DSRA32




**Format:**

DSRA32 rd, rt, sa

**Description:**

The contents of general register *rt* are shifted right by *32 + sa* bits, sign-extending the high-order bits. The result is placed in register *rd*.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

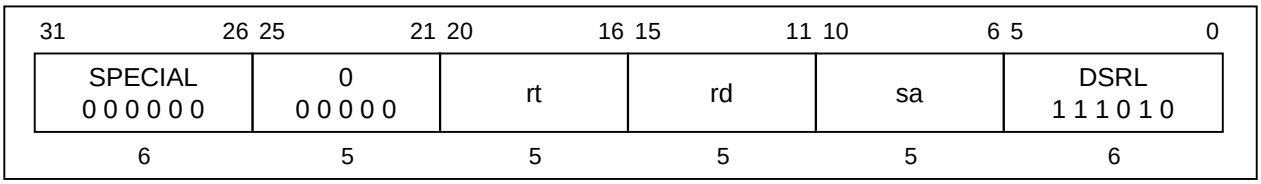
**Operation:**

```
64    T:    s ← 1 || sa
            GPR [rd] ← (GPR [rt]63)^s || GPR [rt]63..s
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# DSRL Doubleword Shift Right Logical DSRL




**Format:**

DSRL rd, rt, sa

**Description:**

The contents of general register *rt* are shifted right by *sa* bits, inserting zeros into the high-order bits. The result is placed in register *rd*.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

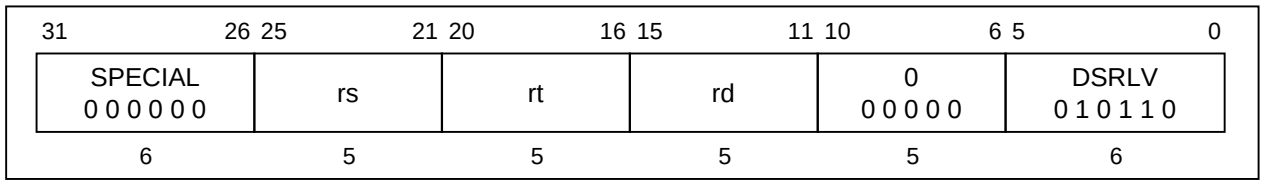
**Operation:**

```
64    T:    s ← 0 || sa
            GPR [rd] ← 0^s || GPR [rt]63..s
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# DSRLV Doubleword Shift Right Logical Variable DSRLV




**Format:**

DSRLV rd, rt, rs

**Description:**

The contents of general register *rt* are shifted right by the number of bits specified by the low-order six bits of general register *rs,* inserting zeros into the high-order bits. The result is placed in register *rd*.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

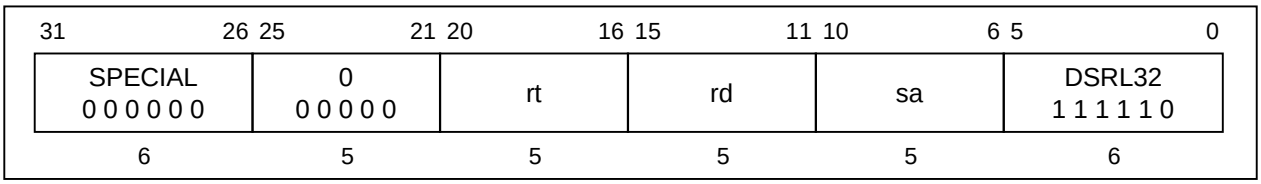
**Operation:**

```
64    T:    s ← GPR [rs]5..0
            GPR [rd] ← 0^s || GPR [rt]63..s
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# DSRL32 Doubleword Shift Right Logical + 32 DSRL32




**Format:**

DSRL32 rd, rt, sa

**Description:**

The contents of general register *rt* are shifted right by *32* + *sa* bits, inserting zeros into the high-order bits. The result is placed in register *rd*.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

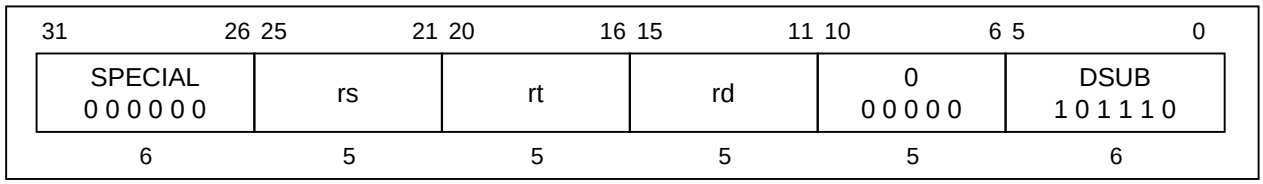
**Operation:**

```
64    T:    s ← 1 || sa
            GPR [rd] ← 0^s || GPR [rt]63..s
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# DSUB Doubleword Subtract DSUB




**Format:**

DSUB rd, rs, rt

**Description:**

The contents of general register *rt* are subtracted from the contents of general register *rs* to form a result. The result is placed into general register *rd.*

An integer overflow exception takes place if the carries out of bits 62 and 63 differ (2's complement overflow). The destination register *rd* is not modified when an integer overflow exception occurs.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

**Operation:**

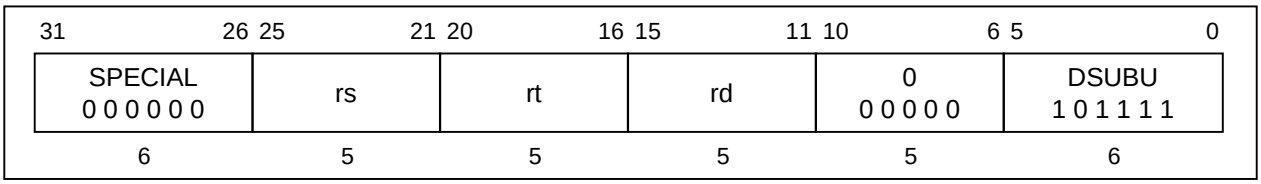
```
64    T:    GPR [rd] ← GPR [rs] − GPR [rt]
```

**Exceptions:**

Integer overflow exception

Reserved instruction exception (32-bit user mode/supervisor mode)

# DSUBU Doubleword Subtract Unsigned DSUBU




**Format:**

DSUBU rd, rs, rt

**Description:**

The contents of general register *rt* are subtracted from the contents of general register *rs* to form a result. The result is placed into general register *rd*.

The only difference between this instruction and the DSUB instruction is that DSUBU never traps on overflow. No integer overflow exception occurs under any circumstances.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

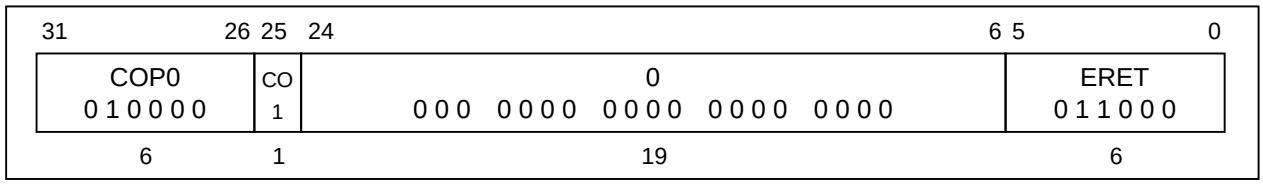
**Operation:**

```
64    T:    GPR [rd] ← GPR [rs] – GPR [rt]
```

**Exceptions:**

Reserved instruction exception (32-bit user mode/supervisor mode)

# ERET Exception Return ERET




**Format:**

ERET

**Description:**

ERET is the instruction for returning from an interrupt, exception, or error trap. Unlike a branch or jump instruction, ERET does not execute the next instruction.

ERET must not itself be placed in a branch delay slot.

If the processor is servicing an error trap (*SR2* = 1), then load the PC from the ErrorEPC register and clear the *ERL* bit of the Status register (*SR2*). Otherwise (*SR2* = 0), load the PC from the EPC register, and clear the *EXL* bit of the Status register (*SR1* = 0).

When a MIPS16 instruction can be executed, the value of clearing the least significant bit of the EPC or error EPC register to 0 is loaded to PC. This means the content of the least significant bit is reflected on the ISA mode bit (internal).

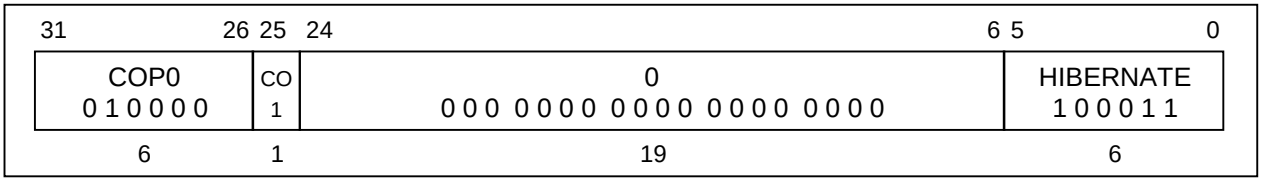
**Operation:**

```
32, 64   T:   if SR2 = 1 then
                if MIPS16EN = 1 then
                  PC ← ErrorEPC63..1 || 0
                  ISA MODE ← ErrorEPC0
                else
                  PC ← ErrorEPC
                endif
                  SR ← SR31..3 || 0 || SR1..0
              else
                if MIPS16EN = 1 then
                  PC ← EPC63..1 || 0
                  ISA MODE ← EPC0
                else
                  PC ← EPC
                endif
                  SR ← SR31..2 || 0 || SR0
              endif
```

**Exceptions:**

Coprocessor unusable exception

# HIBERNATE Hibernate HIBERNATE




**Format:**

HIBERNATE

**Description:**

HIBERNATE instruction starts mode transition from Fullspeed mode to Hibernate mode.

When the HIBERNATE instruction finishes the WB stage, the processor wait by the SysAD bus is idle state, after then the internal clocks and the system interface clocks will shut down, thus freezing the pipeline.

Cold Reset causes the Hibernate mode to the Fullspeed mode transition.

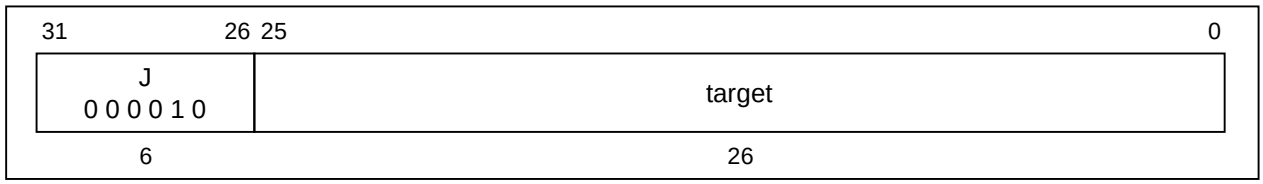
**Operation:**

```
32, 64  T:
        T+1:  Hibernate operation ()
```

**Exceptions:**

Coprocessor unusable exception

# J — Jump — J




**Format:**

J target

**Description:**

The 26-bit target address is shifted left two bits and combined with the high-order four bits of the address of the delay slot. The program unconditionally jumps to this calculated address with a delay of one instruction.

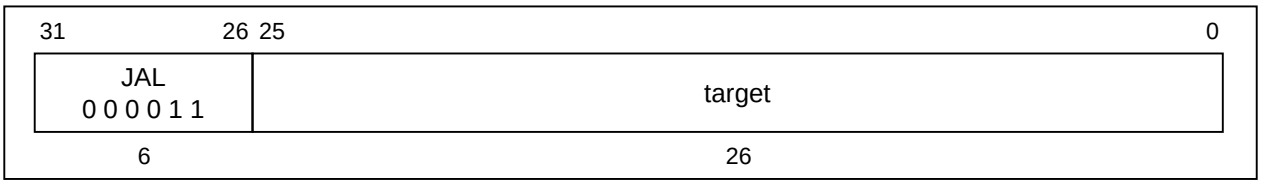
**Operation:**

```
32    T:    temp ← target
      T+1:  PC ← PC31..28 || temp || 0^2

64    T:    temp ← target
      T+1:  PC ← PC63..28 || temp || 0^2
```

**Exceptions:**

None

# JAL — Jump And Link — JAL




**Format:**

JAL target

**Description:**

The 26-bit target address is shifted left two bits and combined with the high-order four bits of the address of the delay slot. The program unconditionally jumps to this calculated address with a delay of one instruction. The address of the instruction after the delay slot is placed in the link register, *r31*. The address of the instruction immediately after a delay slot is placed in the link register (r31). When a MIPS16 instruction can be executed, the value of bit 0 of r31 indicates the ISA mode bit before jump.

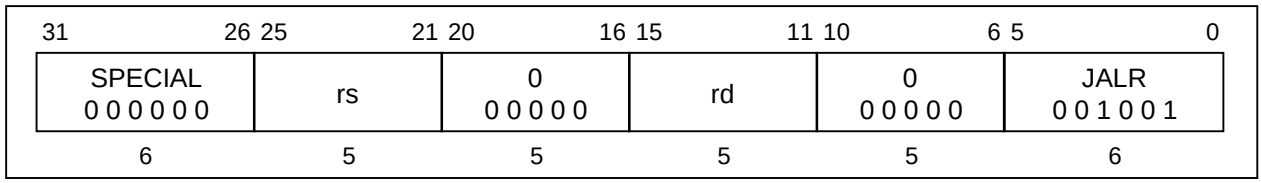
**Operation:**

```
32    T:    temp ← target
            If MIPS16En = 1 then
                GPR[31] ← (PC+8)31..1 || ISA MODE
            else
                GPR[31] ← PC+8
            endif
      T+1:  PC ← PC31..28 || temp || 0^2

64    T:    temp ← target
            If MIPS16EN = 1 then
                GPR[31] ← (PC+8)63..1 || ISA MODE
            else
                GPR[31] ← PC+8
            endif
      T+1:  PC ← PC63..28 || temp || 0^2
```

**Exceptions:**

None

# JALR Jump And Link Register JALR




**Format:**

JALR rs

JALR rd, rs

**Description:**

The program unconditionally jumps to the address contained in general register *rs*, with a delay of one instruction. When a MIPS16 instruction can be executed, the program unconditionally jumps with a delay of one instruction to the address indicated by the value of clearing the least significant bit of the general-purpose register *rs* to 0. Then, the content of the least significant bit of the general-purpose register *rs* is set to the ISA mode bit (internal). The address of the instruction after the delay slot is placed in general register *rd*. The default value of *rd*, if omitted in the assembly language instruction, is 31. When a MIPS16 instruction can be executed, the value of bit 0 of *rd* indicates the ISA mode bit before jump. Register specifiers *rs* and *rd* may not be equal, because such an instruction does not have the same effect when re-executed. Because storing a link address destroys the contents of *rs* if they are equal. However, an attempt to execute this instruction is *not* trapped, and the result of executing such an instruction is undefined.

Since 32-bit length instructions must be word-aligned, a **JALR** instruction must specify a target register (*rs*) that contains an address whose two low-order bits are zero when a MIPS16 instruction can be executed. If these low-order bits are not zero, an address error exception will occur when the jump target instruction is subsequently fetched.

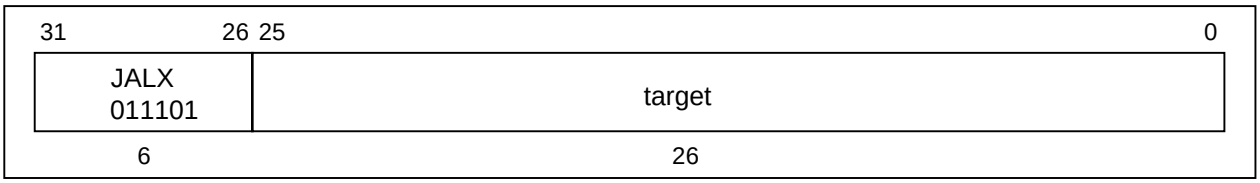
**Operation:**

```
32, 64   T:     temp ← GPR [rs]
                If MIPS16EN = 1 then
                    GPR [rd] ← (PC + 8)63..1 || ISA MODE
                else
                    GPR [rd] ← PC + 8
                endif
         T+1:   If MIPS16EN = 1 then
                    PC ← temp63..1 || 0
                    ISA MODE ← temp0
                else
                    PC ← temp
                endif
```

**Exceptions:**

None

# JALX Jump And Link Exchange JALX




**Format:**

JALX target

**Description:**

When a MIPS16 instruction can be executed, a 26-bit target is shifted to left by 2 bits and then added to higher 4 bits of the delay slot's address to make a target address. The program unconditionally jumps to the target address with a delay of one instruction. The address of the instruction that follows the delay slot is stored to the link register (r31). The ISA mode bit is inverted with a delay of one instruction. The value of bit 0 of the link register (r31) indicates the ISA mode bit before jump.

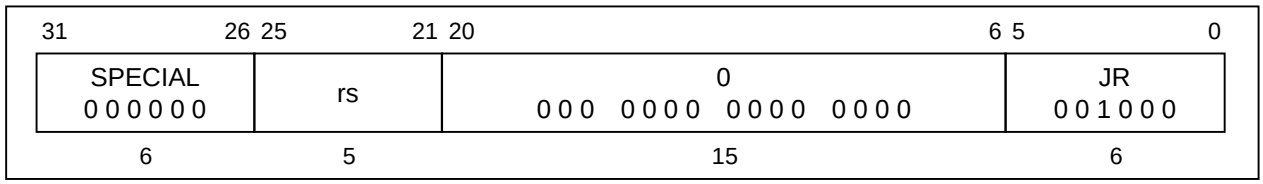
**Operation:**

```
32    T:    temp ← target
            GPR [31] ← (PC + 8)31..1 || ISA MODE
      T+1:  PC ← PC31..28 || temp || 0^2
            ISA MODE toggle

64    T:    temp ← target
            GPR [31] ← (PC + 8)63..1 || ISA MODE
      T+1:  PC ← PC63..28 || temp || 0^2
            ISA MODE toggle
```

**Exceptions:**

Reserved instruction exception (when MIPS16 instruction execution disabled)

# JR Jump Register JR




**Format:**

JR rs

**Description:**

The program unconditionally jumps to the address contained in general register *rs*, with a delay of one instruction.

When a MIPS16 instruction can be executed, the program unconditionally jumps with a delay of one instruction to the address indicated by the value of clearing the least significant bit of the general register *rs* to 0. Then, the content of the least significant bit of the general register *rs* is set to the ISA mode bit (internal).

Since 32-bit length instructions must be word-aligned, a JR instruction must specify a target register (*rs*) that contains an address whose two low-order bits are zero when a MIPS16 instruction can be executed. If these low-order bits are not zero, an address error exception will occur when the jump target instruction is subsequently fetched.

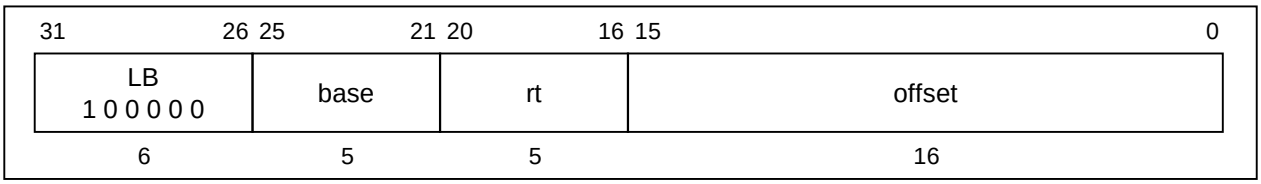
**Operation:**

```
32, 64   T:      temp ← GPR [rs]
         T+1:    If MIPS16EN = 1 then
                     PC ← temp63..1 || 0
                     ISA MODE ← temp0
                 else
                     PC ← temp
                 endif
```

**Exceptions:**

None

# LB Load Byte LB




**Format:**

LB rt, offset (base)

**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The contents of the byte at the memory location specified by the effective address are sign-extended and loaded into general register *rt*.

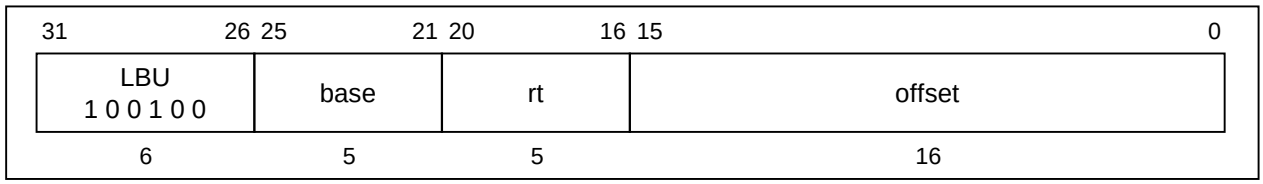
**Operation:**

```
32    T:   vAddr ← ((offset15)^16 || offset15..0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1..3 || (pAddr2..0 xor ReverseEndian^3)
           mem ← LoadMemory (uncached, BYTE, pAddr, vAddr, DATA)
           byte ← vAddr2..0 xor BigEndianCPU^3
           GPR [rt] ← (mem7 + 8* byte)^24 || mem7 + 8* byte..8* byte

64    T:   vAddr ← ((offset15)^48 || offset15..0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1..3 || (pAddr2..0 xor ReverseEndian^3)
           mem ← LoadMemory (uncached, BYTE, pAddr, vAddr, DATA)
           byte ← vAddr2..0 xor BigEndianCPU^3
           GPR [rt] ← (mem7 + 8* byte)^56 || mem7 + 8* byte..8* byte
```

**Exceptions:**

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

# LBU Load Byte Unsigned LBU




**Format:**

LBU rt, offset (base)

**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The contents of the byte at the memory location specified by the effective address are zero-extended and loaded into general register *rt*.

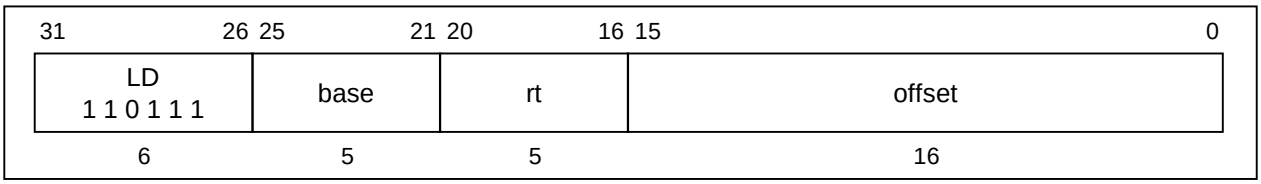
**Operation:**

```
32    T:   vAddr ← ((offset15)^16 || offset15..0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1..3 || (pAddr2..0 xor ReverseEndian^3)
           mem ← LoadMemory (uncached, BYTE, pAddr, vAddr, DATA)
           byte ← vAddr2..0 xor BigEndianCPU^3
           GPR [rt] ← 0^24 || mem7 + 8* byte..8* byte

64    T:   vAddr ← ((offset15)^48 || offset15..0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1..3 || (pAddr2..0 xor ReverseEndian^3)
           mem ← LoadMemory (uncached, BYTE, pAddr, vAddr, DATA)
           byte ← vAddr2..0 xor BigEndianCPU^3
           GPR [rt] ← 0^56 || mem7 + 8* byte..8* byte
```

**Exceptions:**

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

# LD Load Doubleword LD




**Format:**

LD rt, offset (base)

**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The contents of the 64-bit doubleword at the memory location specified by the effective address are loaded into general register *rt*.

If any of the three least-significant bits of the effective address are non-zero, an address error exception occurs.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

**Operation:**

```
64    T:    vAddr ← ((offset15)^48 || offset15..0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            data ← LoadMemory (uncached, DOUBLEWORD, pAddr, vAddr, DATA)
            GPR [rt] ← data
```

**Exceptions:**

TLB refill exception

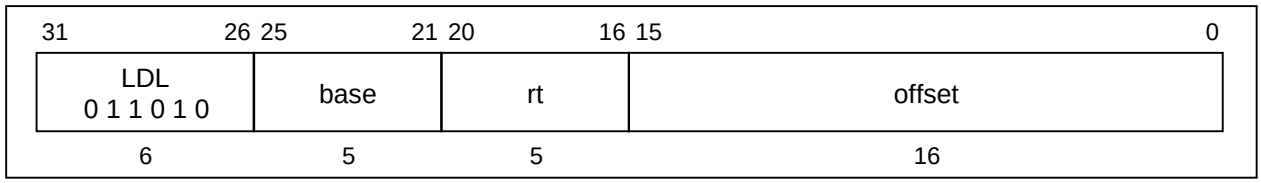
TLB invalid exception

Bus error exception

Address error exception

Reserved instruction exception (32-bit user mode/supervisor mode)

# LDL — Load Doubleword Left (1/3) — LDL




**Format:**

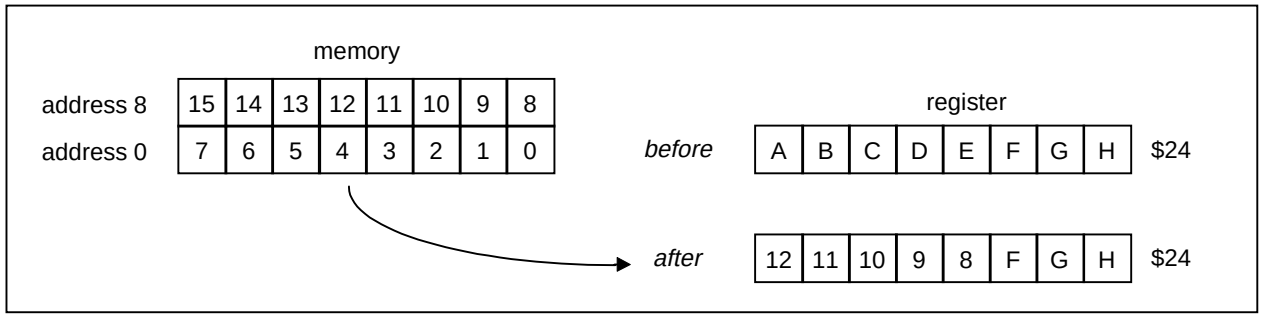
LDL rt, offset (base)

**Description:**

This instruction can be used in combination with the LDR instruction to load a register with eight consecutive bytes from memory, when the bytes cross a doubleword boundary. LDL loads the left portion of the register with the appropriate part of the high-order doubleword; LDR loads the right portion of the register with the appropriate part of the low-order doubleword.

The LDL instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address that can specify an arbitrary byte. It reads bytes only from the doubleword in memory that contains the specified starting byte. From one to eight bytes will be loaded, depending on the starting byte specified.

Conceptually, it starts at the specified byte in memory and loads that byte into the high-order (left-most) byte of the register; then it loads bytes from memory into the register until it reaches the low-order byte of the doubleword in memory. The least-significant (right-most) byte(s) of the register will not be changed.



# LDL Load Doubleword Left (2/3) LDL

The contents of general register *rt* are internally bypassed within the processor so that no NOP is needed between an immediately preceding load instruction which specifies register *rt* and a following LDL (or LDR) instruction which also specifies register *rt*.

No address error exceptions due to alignment are possible.

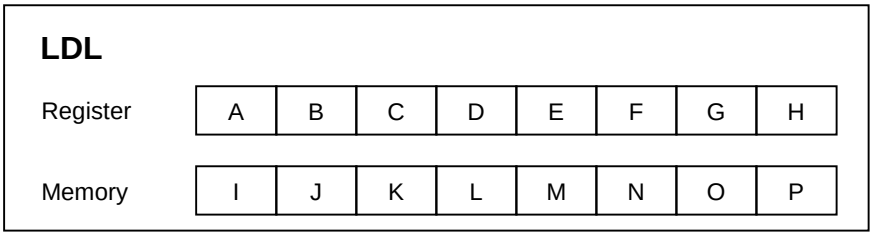
This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

**Operation:**

```
64    T:    vAddr ← ((offset15)^48 || offset15..0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE - 1..3 || (pAddr2..0 xor ReverseEndian^3)
            if BigEndianMem = 0 then
                  pAddr ← pAddrPSIZE - 1..3 || 0^3
            endif
            byte ← vAddr2..0 xor BigEndianCPU^3
            mem ← LoadMemory (uncached, byte, pAddr, vAddr, DATA)
            GPR [rt] ← mem7 + 8 * byte..0 || GPR [rt]55 – 8 * byte..0
```

# LDL Load Doubleword Left (3/3) LDL

Given a doubleword in a register and a doubleword in memory, the operation of LDL is as follows:




| vAddr2..0 | Destination | Type | Offset (LEM) |
|---|---|---|---|
| 0 | P B C D E F G H | 0 | 0 |
| 1 | O P C D E F G H | 1 | 0 |
| 2 | N O P D E F G H | 2 | 0 |
| 3 | M N O P E F G H | 3 | 0 |
| 4 | L M N O P F G H | 4 | 0 |
| 5 | K L M N O P G H | 5 | 0 |
| 6 | J K L M N O P H | 6 | 0 |
| 7 | I J K L M N O P | 7 | 0 |

**Remark** *LEM* Little-endian memory (BigEndianMem = 0)

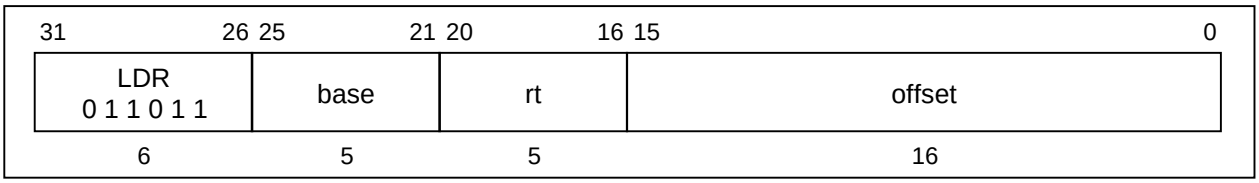
*Type* AccessType (see **Table 2-3. Byte Specification Related to Load and Store Instructions**) sent to memory

*Offset* pAddr2..0 sent to memory

**Exceptions:**

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception
Reserved instruction exception (32-bit user mode/supervisor mode)

# LDR Load Doubleword Right (1/3) LDR




**Format:**

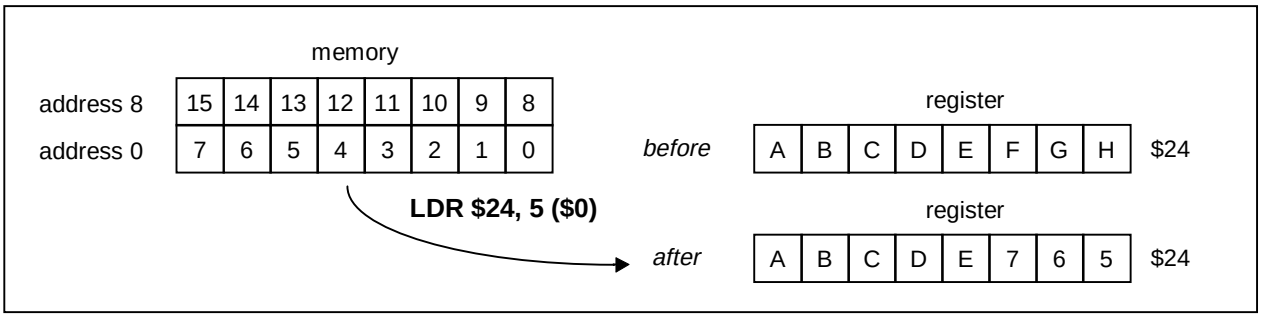
LDR rt, offset (base)

**Description:**

This instruction can be used in combination with the LDL instruction to load a register with eight consecutive bytes from memory, when the bytes cross a doubleword boundary. LDL instruction loads the high-order portion of data and LDR instruction loads the low-order portion of data.

The LDR instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address that can specify an arbitrary byte. It reads bytes only from the doubleword in memory that contains the specified starting byte. From one to eight bytes will be loaded, depending on the starting byte specified.

Conceptually, it starts at the specified byte in memory and loads that byte into the low-order (right-most) byte of the register; then it loads bytes from memory into the register until it reaches the high-order byte of the doubleword in memory. The most significant (left-most) byte(s) of the register will not be changed.



# LDR Load Doubleword Right (2/3) LDR

The contents of general register *rt* are internally bypassed within the processor so that no NOP is needed between an immediately preceding load instruction which specifies register *rt* and a following LDR (or LDL) instruction which also specifies register *rt*.

No address error exceptions due to alignment are possible.

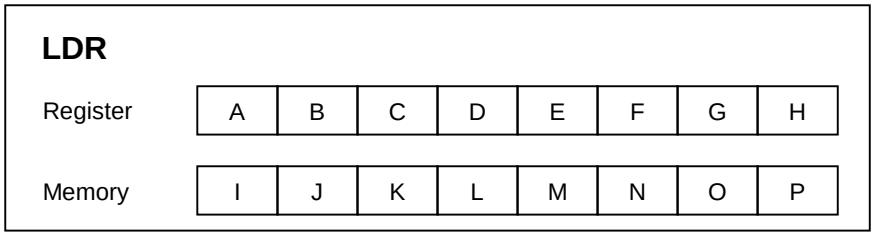
This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

**Operation:**

```
64    T:    vAddr ← ((offset15)^48 || offset15..0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE - 1..3 || (pAddr2..0 xor ReverseEndian^3)
            if BigEndianMem = 1 then
                  pAddr ← pAddrPSIZE - 1..3 || 0^3
            endif
            byte ← vAddr2..0 xor BigEndianCPU^3
            mem ← LoadMemory (uncached, DOUBLEWORD-byte, pAddr, vAddr, DATA)
            GPR [rt] ← GPR [rt]63..64 – 8 * byte || mem63..8 * byte
```

# LDR Load Doubleword Right (3/3) LDR

Given a doubleword in a register and a doubleword in memory, the operation of LDR is as follows:



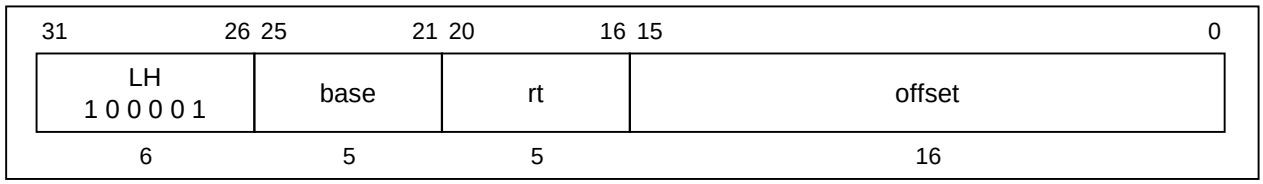

| vAddr2..0 | Destination | Type | Offset (LEM) |
|---|---|---|---|
| 0 | I J K L M N O P | 7 | 0 |
| 1 | A I J K L M N O | 6 | 1 |
| 2 | A B I J K L M N | 5 | 2 |
| 3 | A B C I J K L M | 4 | 3 |
| 4 | A B C D I J K L | 3 | 4 |
| 5 | A B C D E I J K | 2 | 5 |
| 6 | A B C D E F I J | 1 | 6 |
| 7 | A B C D E F G I | 0 | 7 |

**Remark** *LEM* Little-endian memory (BigEndianMem = 0)
*Type* AccessType (see **Table 2-3. Byte Specification Related to Load and Store Instructions**) sent to memory
*Offset* pAddr2..0 sent to memory

**Exceptions:**

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception
Reserved instruction exception (32-bit user mode/supervisor mode)

# LH Load Halfword LH




**Format:**

LH rt, offset (base)

**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The contents of the halfword at the memory location specified by the effective address are sign-extended and loaded into general register *rt*.

If the least-significant bit of the effective address is non-zero, an address error exception occurs.

**Operation:**

```
32    T:   vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor (ReverseEndian^2 || 0))
           mem ← LoadMemory (uncached, HALFWORD, pAddr, vAddr, DATA)
           byte ← vAddr2...0 xor (BigEndianCPU^2 || 0)
           GPR [rt] ← (mem15 + 8 * byte)^16 || mem15 + 8 * byte...8 * byte

64    T:   vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor (ReverseEndian^2 || 0))
           mem ← LoadMemory (uncached, HALFWORD, pAddr, vAddr, DATA)
           byte ← vAddr2...0 xor (BigEndianCPU^2 || 0)
           GPR [rt] ← (mem15 + 8 * byte)^48 || mem15 + 8 * byte...8 * byte
```

**Exceptions:**

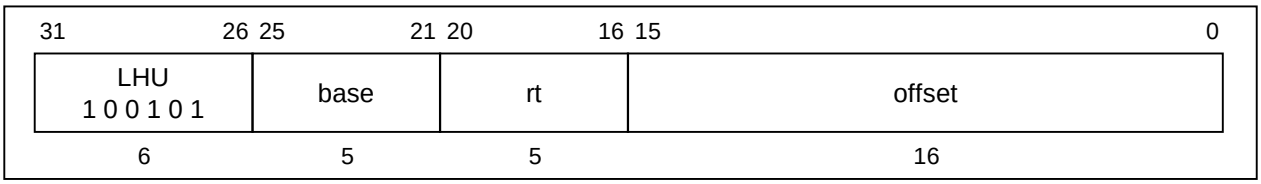
TLB refill exception

TLB invalid exception

Bus error exception

Address error exception

# LHU Load Halfword Unsigned LHU




**Format:**

LHU rt, offset (base)

**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The contents of the halfword at the memory location specified by the effective address are zero-extended and loaded into general register *rt*.

If the least-significant bit of the effective address is non-zero, an address error exception occurs.

**Operation:**

```
32    T:   vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor (ReverseEndian^2 || 0))
           mem ← LoadMemory (uncached, HALFWORD, pAddr, vAddr, DATA)
           byte ← vAddr2...0 xor (BigEndianCPU^2 || 0)
           GPR [rt] ← 0^16 || mem15 + 8 * byte...8 * byte

64    T:   vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor (ReverseEndian^2 || 0))
           mem ← LoadMemory (uncached, HALFWORD, pAddr, vAddr, DATA)
           byte ← vAddr2...0 xor (BigEndianCPU^2 || 0)
           GPR [rt] ← 0^48 || mem15 + 8 * byte...8 * byte
```

**Exceptions:**

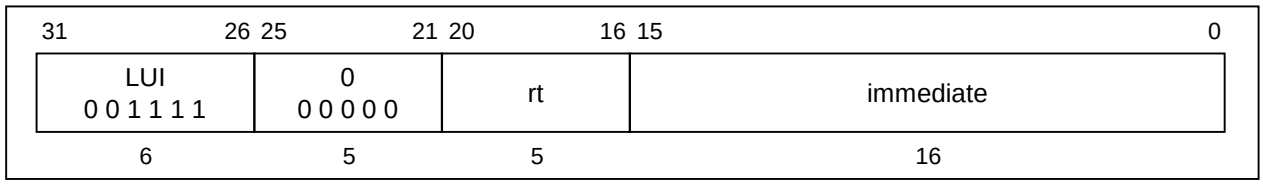
TLB refill exception

TLB invalid exception

Bus Error exception

Address error exception

# LUI Load Upper Immediate LUI




**Format:**

LUI rt, immediate

**Description:**

The 16-bit *immediate* is shifted left 16 bits and concatenated to 16 bits of zeros. The result is placed into general register *rt*. In 64-bit mode, the loaded word is sign-extended.

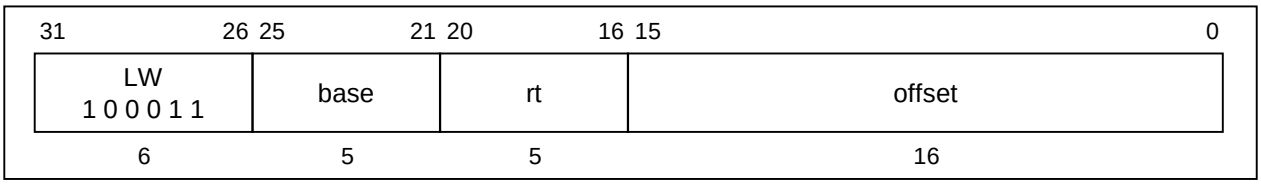
**Operation:**

```
32    T:    GPR [rt] ← immediate || 0^16

64    T:    GPR [rt] ← (immediate15)^32 || immediate || 0^16
```

**Exceptions:**

None

# LW Load Word LW




**Format:**

LW rt, offset (base)

**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The contents of the word at the memory location specified by the effective address are loaded into general register *rt*. In 64-bit mode, the loaded word is sign-extended.

If either of the two least-significant bits of the effective address is non-zero, an address error exception occurs.

**Operation:**

```
32    T:   vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor (ReverseEndian || 0^2))
           mem ← LoadMemory (uncached, WORD, pAddr, vAddr, DATA)
           byte ← vAddr2...0 xor (BigEndianCPU || 0^2)
           GPR [rt] ← mem31 + 8 * byte...8 * byte

64    T:   vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor (ReverseEndian || 0^2))
           mem ← LoadMemory (uncached, WORD, pAddr, vAddr, DATA)
           byte ← vAddr2...0 xor (BigEndianCPU || 0^2)
           GPR [rt] ← (mem31 + 8 * byte)^32 || mem31 + 8 * byte...8 * byte
```

**Exceptions:**

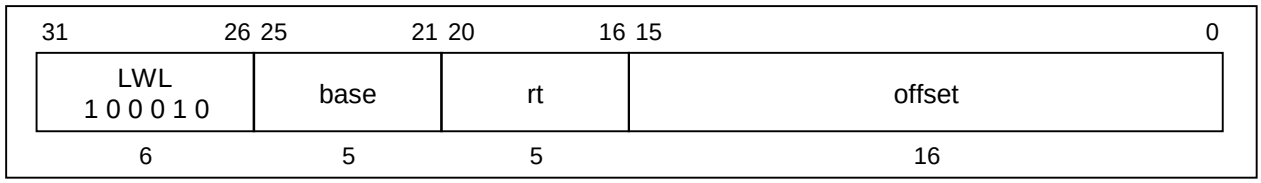
TLB refill exception

TLB invalid exception

Bus error exception

Address error exception

# LWL Load Word Left (1/3) LWL




**Format:**

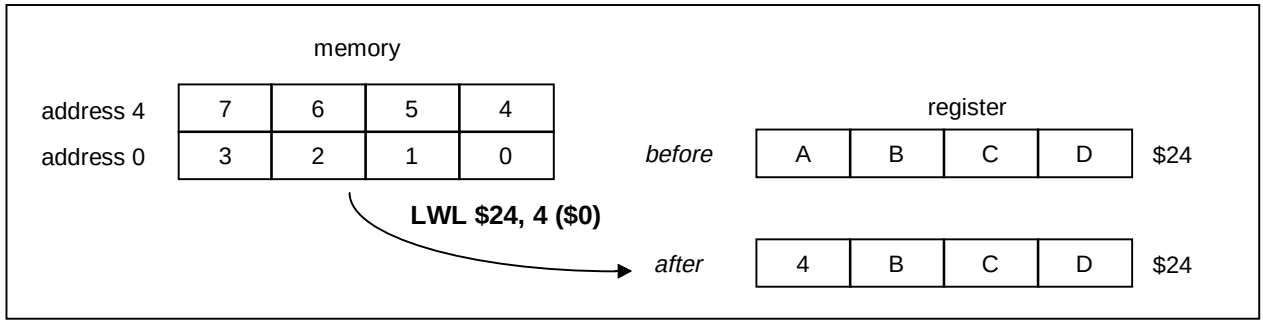
LWL rt, offset (base)

**Description:**

This instruction can be used in combination with the LWR instruction to load a register with four consecutive bytes from memory, when the bytes cross a word boundary. LWL loads the left portion of the register with the appropriate part of the high-order word; LWR loads the right portion of the register with the appropriate part of the low-order word.

The LWL instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address that can specify an arbitrary byte. It reads bytes only from the word in memory that contains the specified starting byte. From one to four bytes will be loaded, depending on the starting byte specified. In 64-bit mode, the loaded word is sign-extended.

Conceptually, it starts at the specified byte in memory and loads that byte into the high-order (left-most) byte of the register; then it loads bytes from memory into the register until it reaches the low-order byte of the word in memory. The least-significant (right-most) byte(s) of the register will not be changed.



# LWL Load Word Left (2/3) LWL

The contents of general register *rt* are internally bypassed within the processor so that no NOP is needed between an immediately preceding load instruction which specifies register *rt* and a following LWL (or LWR) instruction which also specifies register *rt*.

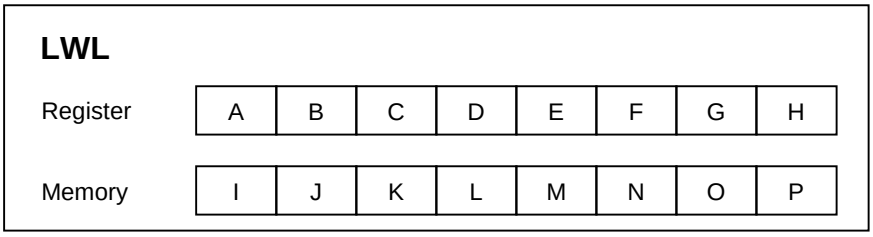
No address error exceptions due to alignment are possible.

**Operation:**

```
32    T:    vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 0 then
                  pAddr ← pAddrPSIZE - 1...2 || 0^2
            endif
            byte ← vAddr1...0 xor BigEndianCPU^2
            word ← vAddr2 xor BigEndianCPU
            mem ← LoadMemory (uncached, byte, pAddr, vAddr, DATA)
            temp ← mem32 * word + 8 * byte + 7...32 * word || GPR [rt]23 – 8 * byte...0
            GPR [rt] ← temp

64    T:    vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 0 then
                  pAddr ← pAddrPSIZE - 1...2 || 0^2
            endif
            byte ← vAddr1...0 xor BigEndianCPU^2
            word ← vAddr2 xor BigEndianCPU
            mem ← LoadMemory (uncached, 0 || byte, pAddr, vAddr, DATA)
            temp ← mem32 * word + 8 * byte + 7...32 * word || GPR [rt]23 – 8 * byte...0
            GPR [rt] ← (temp31)^32 || temp
```

# LWL Load Word Left (3/3) LWL

Given a doubleword in a register and a doubleword in memory, the operation of LWL is as follows:



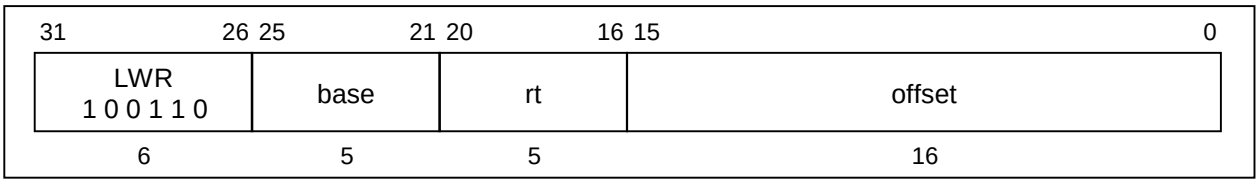

| vAddr2..0 | Destination | Type | Offset (LEM) |
|---|---|---|---|
| 0 | S S S S P F G H | 0 | 0 |
| 1 | S S S S O P G H | 1 | 0 |
| 2 | S S S S N O P H | 2 | 0 |
| 3 | S S S S M N O P | 3 | 0 |
| 4 | S S S S L F G H | 0 | 4 |
| 5 | S S S S K L G H | 1 | 4 |
| 6 | S S S S J K L H | 2 | 4 |
| 7 | S S S S I J K L | 3 | 4 |

**Remark** *LEM* Little-endian memory (BigEndianMem = 0)
*Type* AccessType (see **Table 2-3. Byte Specification Related to Load and Store Instructions**) sent to memory
*Offset* pAddr2..0 sent to memory
*S* sign-extend of destination bit 31

**Exceptions:**

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

# LWR — Load Word Right (1/3) — LWR




**Format:**

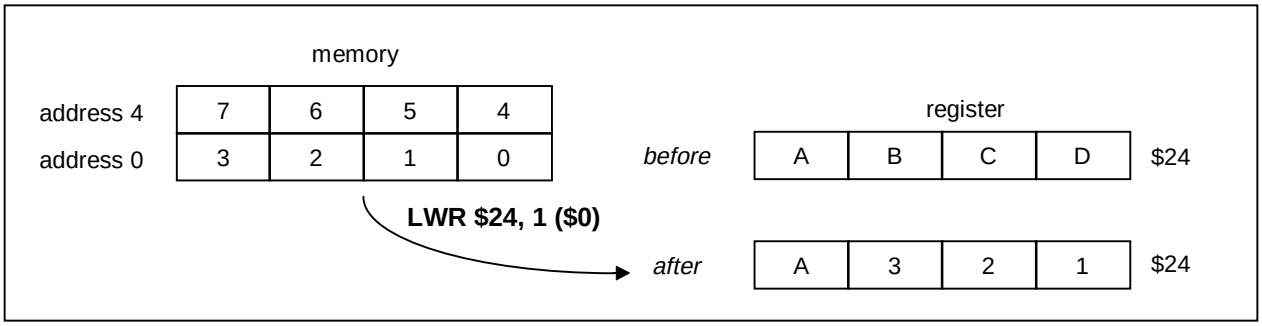
LWR rt, offset (base)

**Description:**

This instruction can be used in combination with the LWL instruction to load a register with four consecutive bytes from memory, when the bytes cross a word boundary. LWR loads the right portion of the register with the appropriate part of the low-order word; LWL loads the left portion of the register with the appropriate part of the high-order word.

The LWR instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address that can specify an arbitrary byte. It reads bytes only from the word in memory that contains the specified starting byte. From one to four bytes will be loaded, depending on the starting byte specified. In 64-bit mode, the loaded word is sign-extended.

Conceptually, it starts at the specified byte in memory and loads that byte into the low-order (right-most) byte of the register; then it loads bytes from memory into the register until it reaches the high-order byte of the word in memory. The most significant (left-most) byte(s) of the register will not be changed.



# LWR Load Word Right (2/3) LWR

The contents of general register *rt* are internally bypassed within the processor so that no NOP is needed between an immediately preceding load instruction which specifies register *rt* and a following LWR (or LWL) instruction which also specifies register *rt.*

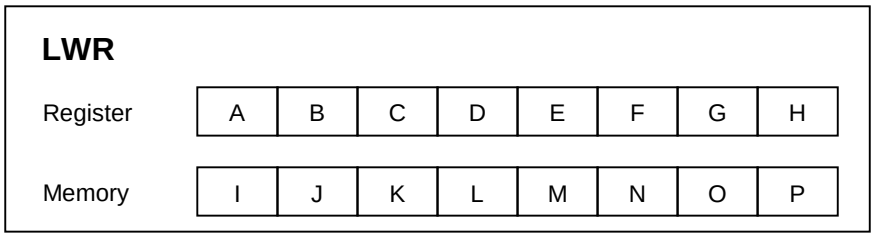
No address error exceptions due to alignment are possible.

**Operation:**

```
32   T:   vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
          (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
          pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor ReverseEndian^3)
          if BigEndianMem = 1 then
               pAddr ← pAddrPSIZE - 1...3 || 0^3
          endif
          byte ← vAddr1...0 xor BigEndianCPU^2
          word ← vAddr2 xor BigEndianCPU
          mem ← LoadMemory (uncached, 0 || byte, pAddr, vAddr, DATA)
          temp ← GPR [rt]31...32 – 8 * byte || mem31 + 32 * word...32 * word + 8 * byte
          GPR [rt] ← temp

64   T:   vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
          (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
          pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor ReverseEndian^3)
          if BigEndianMem = 1 then
               pAddr ← pAddrPSIZE - 1...3 || 0^3
          endif
          byte ← vAddr1...0 xor BigEndianCPU^2
          word ← vAddr2 xor BigEndianCPU
          mem ← LoadMemory (uncached, WORD-byte, pAddr, vAddr, DATA)
          temp ← GPR [rt]31...32 – 8 * byte || mem31 + 32 * word...32 * word + 8 * byte
          GPR [rt] ← (temp31)^32 || temp
```

# LWR Load Word Right (3/3) LWR

Given a word in a register and a word in memory, the operation of LWR is as follows:



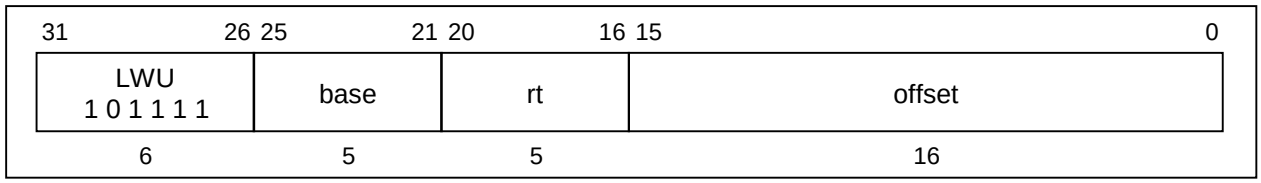

| vAddr2..0 | Destination | Type | Offset (LEM) |
|---|---|---|---|
| 0 | S S S S M N O P | 3 | 0 |
| 1 | S S S S E M N O | 2 | 1 |
| 2 | S S S S E F M N | 1 | 2 |
| 3 | S S S S E F G M | 0 | 3 |
| 4 | S S S S I J K L | 3 | 4 |
| 5 | S S S S E I J K | 2 | 5 |
| 6 | S S S S E F I J | 1 | 6 |
| 7 | S S S S E F G I | 0 | 7 |

**Remark** *LEM* Little-endian memory (BigEndianMem = 0)
*Type* AccessType (see **Table 2-3. Byte Specification Related to Load and Store Instructions**) sent to memory
*Offset* pAddr2..0 sent to memory
*S* sign-extend of destination31

**Exceptions:**

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

# LWU Load Word Unsigned LWU




**Format:**

LWU rt, offset (base)

**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The contents of the word at the memory location specified by the effective address are loaded into general register *rt*. The loaded word is zero-extended.

If either of the two least-significant bits of the effective address is non-zero, an address error exception occurs.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

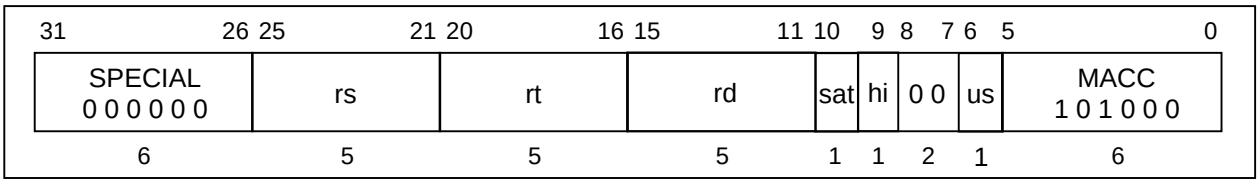
**Operation:**

```
32   T:   vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
          (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
          pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor (ReverseEndian || 0^2))
          mem ← LoadMemory (uncached, WORD, pAddr, vAddr, DATA)
          byte ← vAddr2...0 xor (BigEndianCPU || 0^2)
          GPR [rt] ← 0^32 || mem31 + 8 * byte...8 * byte

64   T:   vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
          (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
          pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor (ReverseEndian || 0^2))
          mem ← LoadMemory (uncached, WORD, pAddr, vAddr, DATA)
          byte ← vAddr2...0 xor (BigEndianCPU || 0^2)
          GPR [rt] ← 0^32 || mem31 + 8 * byte...8 * byte
```

**Exceptions:**

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception
Reserved instruction exception (32-bit user mode/supervisor mode)

# MACC Multiply and Accumulate (1/5) MACC




**Format:**

MACC rd, rs, rt
MACCU rd, rs, rt
MACCHI rd, rs, rt
MACCHIU rd, rs, rt
MACCS rd, rs, rt
MACCUS rd, rs, rt
MACCHIS rd, rs, rt
MACCHIUS rd, rs, rt

**Description:**

MACC instruction differs mnemonics by each setting of op codes sat, hi and us as follows.

| Mnemonic | sat | hi | us |
|---|---|---|---|
| MACC | 0 | 0 | 0 |
| MACCU | 0 | 0 | 1 |
| MACCHI | 0 | 1 | 0 |
| MACCHIU | 0 | 1 | 1 |
| MACCS | 1 | 0 | 0 |
| MACCUS | 1 | 0 | 1 |
| MACCHIS | 1 | 1 | 0 |
| MACCHIUS | 1 | 1 | 1 |

The number of significant bits in the operands of the MACC instruction differ depending on whether saturation processing is executed (sat = 1) or not executed (sat = 0).

- When saturation processing is executed (sat = 1): MACCS, MACCUS, MACCHIS, MACCHIUS instructions
  The contents of general register *rs* is multiplied by the contents of general register *rt*. If both operands are set as "us = 1" (MACCUS, MACCHIUS instructions), the contents are handled as 16 bit unsigned data. If they are set as "us = 0" (MACCS, MACCHIS instructions), the contents are handled as 16 bit signed integers. Sign/zero expansion by software is required for any bits exceeding 16 bits in the operands. The product of this multiply operation is added to a 64-bit value (of which only the low-order 32 bits are valid) that is linked to the HI and LO special registers. If us = 1, this add operation handles the values being added as 32 bit unsigned data. If us = 0, the values are handled as 32 bit signed integers. Sign/zero expansion by software is required for any bits exceeding 32 bits in the linked HI and LO special registers. After saturation processing to 32 bits has been performed (see the table below), the sum from this add operation is loaded to the HI and LO special register. When hi = 1 (MACCHIS, MACCHIUS instructions), data that is the same as the data loaded to the HI special register is also loaded to the rd general register. When hi = 0 (MACCS, MACCUS instructions), data that is the same as the data loaded to the LO special register is also loaded to the rd general register. Overflow exceptions do not occur.

# MACC Multiply and Accumulate (2/5) MACC

- When saturation processing is not executed (sat = 0): MACC, MACCU, MACCHI, MACCHIU instructions
  The contents of general register *rs* is multiplied to the contents of general register *rt*. If both operands are set as "us = 1" (MACCU, MACCHIU instructions), the contents are handled as 32 bit unsigned data. If they are set as "us = 0" (MACC, MACCHI instructions), the contents are handled as 32 bit signed integers. Sign/zero expansion by software is required for any bits exceeding 32 bits in the operands. The product of this multiply operation is added to a 64-bit value that is linked to the HI and LO special registers. If us = 1, this add operation handles the values being added as 64 bit unsigned data. If us = 0, the values are handled as 64 bit signed integers.
  The low-order word from the 64-bit sum from this add operation is loaded to the LO special register and the high-order word is loaded to the HI special register. When hi = 1 (MACCHI, MACCHIU instructions), data that is the same as the data loaded to the HI special register is also loaded to the rd general register. When hi = 0 (MACC, MACCU instructions), data that is the same as the data loaded to the LO special register is also loaded to the rd general register. Overflow exceptions do not occur.

The correspondence of us and sat settings and values stored during saturation processing is shown below, along with the hazard cycles required between execution of the instruction for manipulating the HI and LO registers and execution of the MACC instruction.

**Values Stored during Saturation Processing**

| us | sat | Overflow | Underflow |
|---|---|---|---|
| 0 | 0 | Store calculation result as is | Store calculation result as is |
| 1 | 0 | Store calculation result as is | Store calculation result as is |
| 0 | 1 | 0x0000 0000 7FFF FFFF | 0xFFFF FFFF 8000 0000 |
| 1 | 1 | 0xFFFF FFFF FFFF FFFF | None |

**Hazard Cycle Counts**

| Instruction | Cycle Count |
|---|---|
| MULT, MULTU | 1 |
| DMULT, DMULTU | 3 |
| DIV, DIVU | 36 |
| DDIV, DDIVU | 68 |
| MFHI, MFLO | 2 |
| MTHI, MTLO | 0 |
| MACC | 0 |
| DMACC | 0 |

# MACC — Multiply and Accumulate (3/5) — MACC

**Operation:**

```
32, sat=0, hi=0, us=0 (MACC instruction)
    T:  temp1 ← GPR[rs] * GPR[rt]
        temp2 ← temp1 + (HI || LO)
        LO ← temp2[63..32]
        HI ← temp2[31..0]
        GPR[rd] ← LO
32, sat=0, hi=0, us=1 (MACCU instruction)
    T:  temp1 ← (0 || GPR[rs]) * (0 || GPR[rt])
        temp2 ← temp1 + ((0 || HI) || (0 || LO))
        LO ← temp2[63..32]
        HI ← temp2[31..0]
        GPR[rd] ← LO
32, sat=0, hi=1, us=0 (MACCHI instruction)
    T:  temp1 ← GPR[rs] * GPR[rt]
        temp2 ← temp1 + (HI || LO)
        LO ← temp2[63..32]
        HI ← temp2[31..0]
        GPR[rd] ← HI
32, sat=0, hi=1, us=1 (MACCHIU instruction)
    T:  temp1 ← (0 || GPR[rs]) * (0 || GPR[rt])
        temp2 ← temp1 + ((0 || HI) || (0 || LO))
        LO ← temp2[63..32]
        HI ← temp2[31..0]
        GPR[rd] ← HI
32, sat=1, hi=0, us=0 (MACCS instruction)
    T:  temp1 ← GPR[rs] * GPR[rt]
        temp2 ← saturation(temp1 + (HI || LO))
        LO ← temp2[63..32]
        HI ← temp2[31..0]
        GPR[rd] ← LO
32, sat=1, hi=0, us=1 (MACCUS instruction)
    T:  temp1 ← (0 || GPR[rs]) * (0 || GPR[rt])
        temp2 ← saturation(temp1 + ((0 || HI) || (0 || LO)))
        LO ← temp2[63..32]
        HI ← temp2[31..0]
        GPR[rd] ← LO
```

# MACC Multiply and Accumulate (4/5) MACC

32, sat=1, hi=1, us=0 (MACCHIS instruction)

```
T:  temp1 ← GPR[rs] * GPR[rt]
    temp2 ← saturation(temp1 + (HI || LO))
    LO ← temp2[63..32]
    HI ← temp2[31..0]
    GPR[rd] ← HI
```

32, sat=1, hi=1, us=1 (MACCHIUS instruction)

```
T:  temp1 ← (0 || GPR[rs]) * (0 || GPR[rt])
    temp2 ← saturation(temp1 + ((0 || HI) || (0 || LO)))
    LO ← temp2[63..32]
    HI ← temp2[31..0]
    GPR[rd] ← HI
```

64, sat=0, hi=0, us=0 (MACC instruction)

```
T:  temp1 ← ((GPR[rs][31])^32 || GPR[rs]) * ((GPR[rt][31])^32 || GPR[rt])
    temp2 ← temp1 + (HI[31..0] || LO[31..0])
    LO ← ((temp2[63])^32 || temp2[63..32])
    HI ← ((temp2[31])^32 || temp2[31..0])
    GPR[rd] ← LO
```

64, sat=0, hi=0, us=1 (MACCU instruction)

```
T:  temp1 ← (0^32 || GPR[rs]) * (0^32 || GPR[rt])
    temp2 ← temp1 + (HI[31..0] || LO[31..0])
    LO ← ((temp2[63])^32 || temp2[63..32])
    HI ← ((temp2[31])^32 || temp2[31..0])
    GPR[rd] ← LO
```

64, sat=0, hi=1, us=0 (MACCHI instruction)

```
T:  temp1 ← ((GPR[rs][31])^32 || GPR[rs]) * ((GPR[rt][31])^32 || GPR[rt])
    temp2 ← temp1 + (HI[31..0] || LO[31..0])
    LO ← ((temp2[63])^32 || temp2[63..32])
    HI ← ((temp2[31])^32 || temp2[31..0])
    GPR[rd] ← HI
```

64, sat=0, hi=1, us=1 (MACCHIU instruction)

```
T:  temp1 ← (0^32 || GPR[rs]) * (0^32 || GPR[rt])
    temp2 ← temp1 + (HI[31..0] || LO[31..0])
    LO ← ((temp2[63])^32 || temp2[63..32])
    HI ← ((temp2[31])^32 || temp2[31..0])
    GPR[rd] ← HI
```

# MACC — Multiply and Accumulate (5/5) — MACC

64, sat=1, hi=0, us=0 (MACCS instruction)

```
T:  temp1 ← ((GPR[rs]31)^32 || GPR[rs]) * ((GPR[rt]31)^32 || GPR[rt])
    temp2 ← saturation(temp1 + (HI31..0 || LO31..0))
    LO ← ((temp263)^32 || temp263..32)
    HI ← ((temp231)^32 || temp231..0)
    GPR[rd] ← LO
```

64, sat=1, hi=0, us=1 (MACCUS instruction)

```
T:  temp1 ← (0^32 || GPR[rs]) * (0^32 || GPR[rt])
    temp2 ← saturation(temp1 + (HI31..0 || LO31..0))
    LO ← ((temp263)^32 || temp263..32)
    HI ← ((temp231)^32 || temp231..0)
    GPR[rd] ← LO
```

64, sat=1, hi=1, us=0 (MACCHIS instruction)

```
T:  temp1 ← ((GPR[rs]31)^32 || GPR[rs]) * ((GPR[rt]31)^32 || GPR[rt])
    temp2 ← saturation(temp1 + (HI31..0 || LO31..0))
    LO ← ((temp263)^32 || temp263..32)
    HI ← ((temp231)^32 || temp231..0)
    GPR[rd] ← HI
```

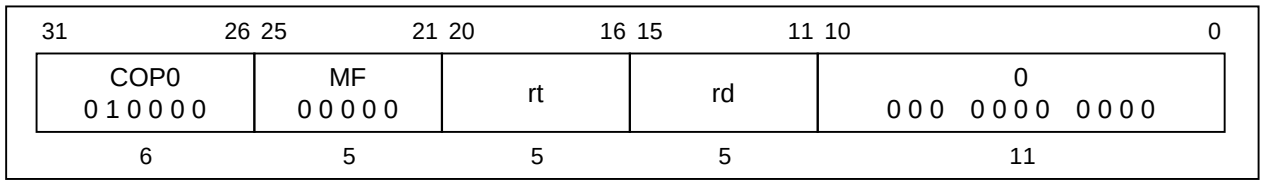
64, sat=1, hi=1, us=1 (MACCHIUS instruction)

```
T:  temp1 ← (0^32 || GPR[rs]) * (0^32 || GPR[rt])
    temp2 ← saturation(temp1 + (HI31..0 || LO31..0)
    LO ← ((temp263)^32 || temp263..32)
    HI ← ((temp231)^32 || temp231..0)
    GPR[rd] ← HI
```

**Exceptions:**

None

# MFC0 Move From System Control Coprocessor MFC0




**Format:**

MFC0 rt, rd

**Description:**

The contents of coprocessor register *rd* of the CP0 are loaded into general register *rt.*

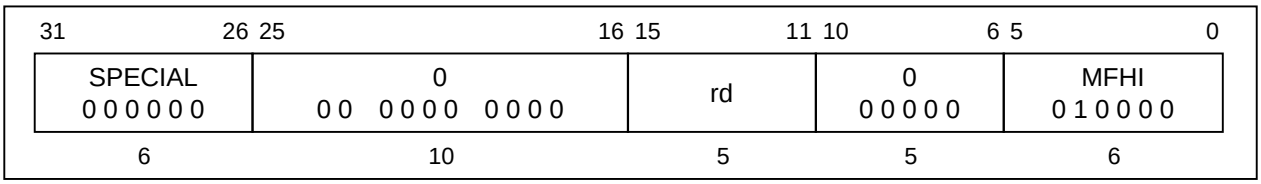
**Operation:**

```
32   T:     data  ← CPR [0, rd]
       T+1: GPR [rt] ← data

64   T:     data  ← CPR [0, rd]
       T+1: GPR [rt] ← (data31)^32 || data31...0
```

**Exceptions:**

Coprocessor unusable exception (in 64-bit/32-bit user and supervisor mode if CP0 not enabled)

# MFHI — Move From HI — MFHI




**Format:**

MFHI rd

**Description:**

The contents of special register *HI* are loaded into general register *rd*.

To ensure proper operation in the event of interruptions, the two instructions which follow a MFHI instruction may not be any of the instructions which modify the *HI* register: MULT, MULTU, DIV, DIVU, MTHI, DMULT, DMULTU, DDIV, DDIVU.

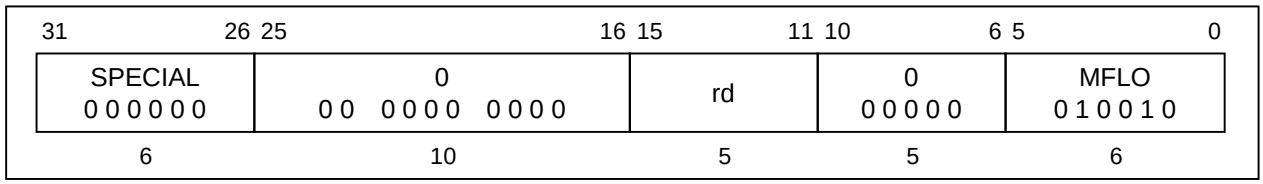
**Operation:**

```
32, 64  T:    GPR [rd] ← HI
```

**Exceptions:**

None

# MFLO Move From LO MFLO




**Format:**

MFLO rd

**Description:**

The contents of special register *LO* are loaded into general register *rd*.

To ensure proper operation in the event of interruptions, the two instructions which follow a MFLO instruction may not be any of the instructions which modify the *LO* register: MULT, MULTU, DIV, DIVU, MTLO, DMULT, DMULTU, DDIV, DDIVU.

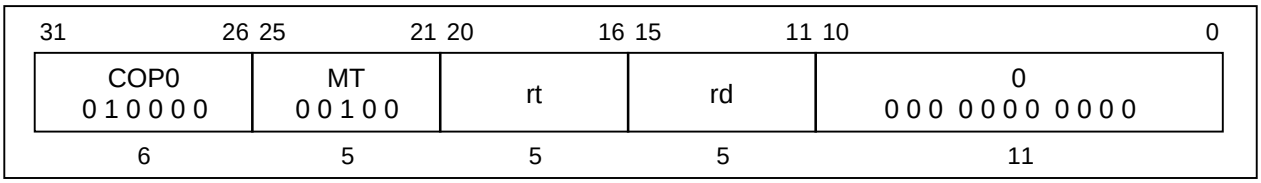
**Operation:**

```
32, 64 T:    GPR [rd] ← LO
```

**Exceptions:**

None

# MTC0 Move To Coprocessor0 MTC0




**Format:**

MTC0 rt, rd

**Description:**

The contents of general register *rt* are loaded into coprocessor register *rd* of coprocessor 0.

Because the state of the virtual address translation system may be altered by this instruction, the operation of load instructions, store instructions, and TLB operations immediately prior to and after this instruction are undefined.

When using a register used by the MTC0 by means of instructions before and after it, refer to **APPENDIX C VR4120A COPROCESSOR 0 HAZARDS** and place the instructions in the appropriate location.

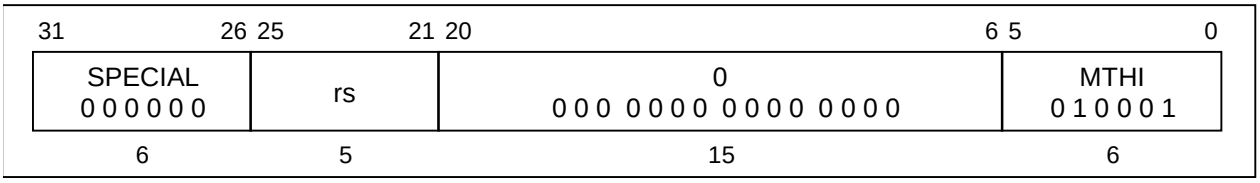
**Operation:**

```
32, 64  T:    data ← GPR [rt]
        T+1:  CPR [0, rd] ← data
```

**Exceptions:**

Coprocessor unusable exception (in 64-bit/32-bit user and supervisor mode if CP0 not enabled)

# MTHI — Move To HI — MTHI




**Format:**

MTHI rs

**Description:**

The contents of general register *rs* are loaded into special register *HI*.

If a MTHI operation is executed following a MULT, MULTU, DIV, or DIVU instruction, but before any MFLO, MFHI, MTLO, or MTHI instructions, the contents of special register *HI* are undefined.

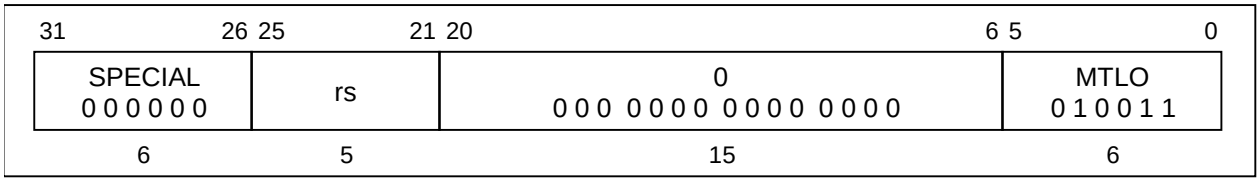
**Operation:**

```
32, 64 T-2:  HI ← undefined
       T-1:  HI ← undefined
       T:    HI ← GPR [rs]
```

**Exceptions:**

None

# MTLO Move To LO MTLO




**Format:**

MTLO rs

**Description:**

The contents of general register *rs* are loaded into special register *LO.*

If an MTLO operation is executed following a MULT, MULTU, DIV, or DIVU instruction, but before any MFLO, MFHI, MTLO, or MTHI instructions, the contents of special register *LO* are undefined.

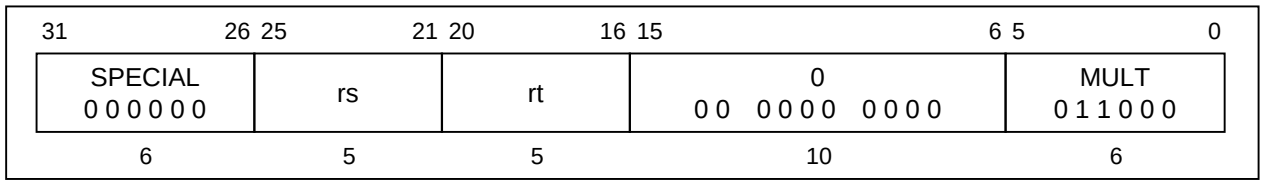
**Operation:**

```
32, 64 T-2: LO ← undefined
       T-1: LO ← undefined
       T:   LO ← GPR [rs]
```

**Exceptions:**

None

# MULT — Multiply — MULT




**Format:**

MULT rs, rt

**Description:**

The contents of general registers *rs* and *rt* are multiplied, treating both operands as signed 32-bit integer. No integer overflow exception occurs under any circumstances.

In 64-bit mode, the operands must be valid 32-bit, sign-extended values.

When the operation completes, the low-order word of the double result is loaded into special register *LO*, and the high-order word of the double result is loaded into special register *HI*.

If either of the two preceding instructions is MFHI or MFLO, the results of these instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by a minimum of two other instructions.

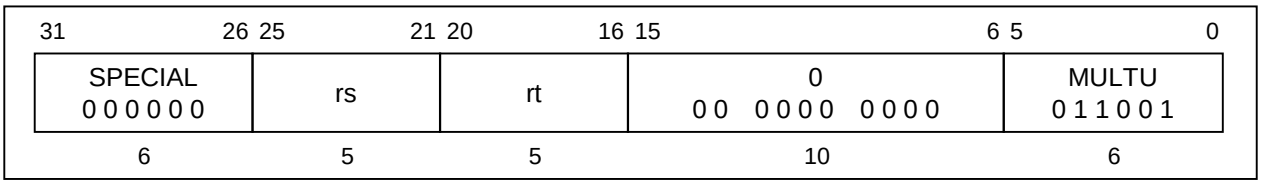
**Operation:**

```
32    T-2:  LO  ← undefined
            HI  ← undefined
      T-1:  LO  ← undefined
            HI  ← undefined
      T:    t   ← GPR [rs] * GPR [rt]
            LO  ← t31...0
            HI  ← t63...32

64    T-2:  LO  ← undefined
            HI  ← undefined
      T-1:  LO  ← undefined
            HI  ← undefined
      T:    t   ← GPR [rs]31...0 * GPR [rt]31...0
            LO  ← (t31)^32 || t31...0
            HI  ← (t63)^32 || t63...32
```

**Exceptions:**

None

# MULTU — Multiply Unsigned — MULTU




**Format:**

MULTU rs, rt

**Description:**

The contents of general register *rs* and the contents of general register *rt* are multiplied, treating both operands as unsigned values. No overflow exception occurs under any circumstances.

In 64-bit mode, the operands must be valid 32-bit, sign-extended values.

When the operation completes, the low-order word of the double result is loaded into special register *LO*, and the high-order word of the double result is loaded into special register *HI*.

If either of the two preceding instructions is MFHI or MFLO, the results of these instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by a minimum of two instructions.

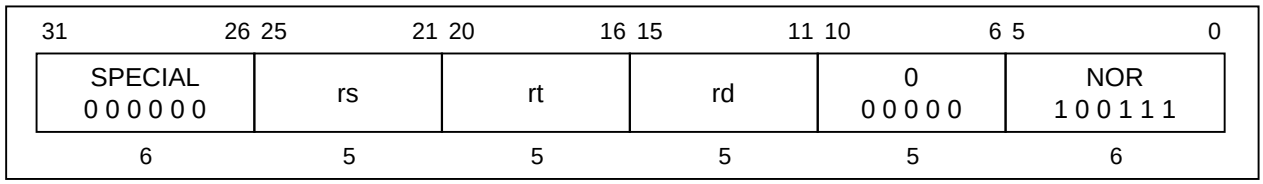
**Operation:**

```
32    T-2:  LO   ← undefined
            HI   ← undefined
      T-1:  LO   ← undefined
            HI   ← undefined
      T:    t    ← (0 || GPR [rs]) * (0 || GPR [rt])
            LO   ← t31...0
            HI   ← t63...32

64    T-2:  LO   ← undefined
            HI   ← undefined
      T-1:  LO   ← undefined
            HI   ← undefined
      T:    t    ← (0 || GPR [rs]31...0) * (0 || GPR [rt]31...0)
            LO   ← (t31)32 || t31...0
            HI   ← (t63)32 || t63...32
```

**Exceptions:**

None

# NOR Nor NOR




**Format:**

NOR rd, rs, rt

**Description:**

The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical NOR operation. The result is placed into general register *rd*.

**Operation:**

```
32, 64 T:   GPR [rd] ← GPR [rs] nor GPR [rt]
```

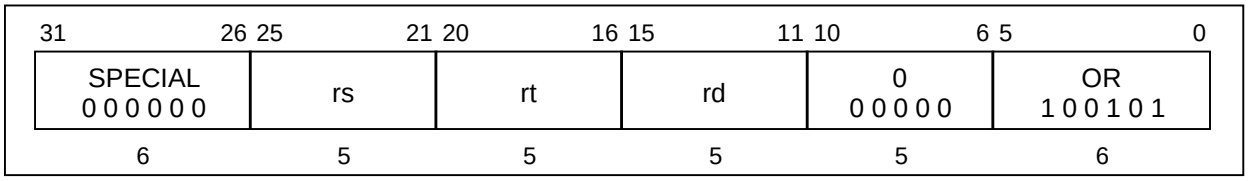
**Exceptions:**

None

# OR

## Or

## OR




**Format:**

OR rd, rs, rt

**Description:**

The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical OR operation. The result is placed into general register *rd*.

**Operation:**

```
32, 64 T:    GPR [rd] ← GPR [rs] or GPR [rt]
```

**Exceptions:**

None

# ORI — Or Immediate — ORI

**Format:**

ORI rt, rs, immediate

**Description:**

The 16-bit *immediate* is zero-extended and combined with the contents of general register *rs* in a bit-wise logical OR operation. The result is placed into general register *rt*.

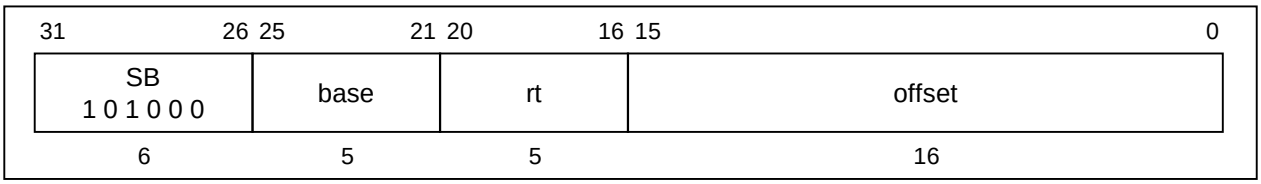
**Operation:**

```
32   T:   GPR [rt] ← GPR [rs] 31...16 || (immediate or GPR [rs] 15...0)

64   T:   GPR [rt] ← GPR [rs] 63...16 || (immediate or GPR [rs] 15...0)
```

**Exceptions:**

None

# SB Store Byte SB




**Format:**

SB rt, offset (base)

**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The least-significant byte of register *rt* is stored at the effective address.

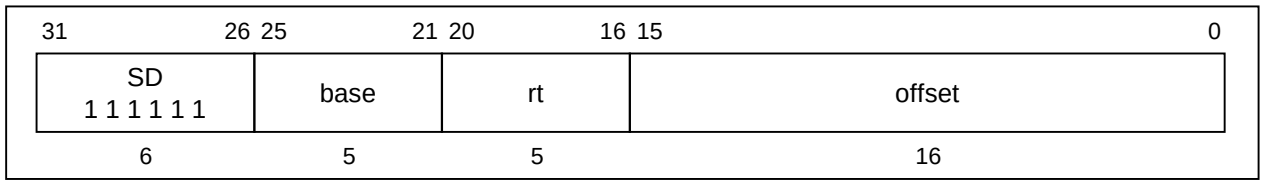
**Operation:**

```
32    T:    vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor (ReverseEndian^3))
            byte ← vAddr2...0 xor BigEndianCPU^3
            data ← GPR [rt]63 – 8 * byte...0 || 0^8 * byte
            StoreMemory (uncached, BYTE, data, pAddr, vAddr, DATA)

64    T:    vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor (ReverseEndian^3))
            byte ← vAddr2...0 xor BigEndianCPU^3
            data ← GPR [rt]63 – 8 * byte...0 || 0^8 * byte
            StoreMemory (uncached, BYTE, data, pAddr,  vAddr, DATA)
```

**Exceptions:**

TLB refill exception
TLB invalid exception
TLB modification exception
Bus error exception
Address error exception

# SD Store Doubleword SD




**Format:**

SD rt, offset (base)

**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The contents of general register *rt* are stored at the memory location specified by the effective address.

If either of the three least-significant bits of the effective address are non-zero, an address error exception occurs.

This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

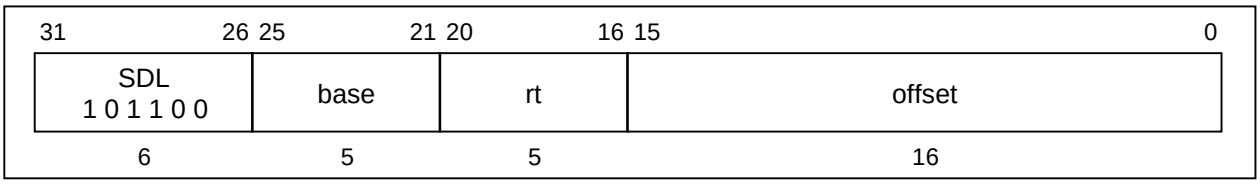
**Operation:**

```
64    T:    vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            data ← GPR [rt]
            StoreMemory (uncached, DOUBLEWORD, data, pAddr, vAddr, DATA)
```

**Exceptions:**

TLB refill exception
TLB invalid exception
TLB modification exception
Bus error exception
Address error exception
Reserved instruction exception (32-bit user mode/supervisor mode)

# SDL Store Doubleword Left (1/3) SDL




**Format:**

SDL rt, offset (base)

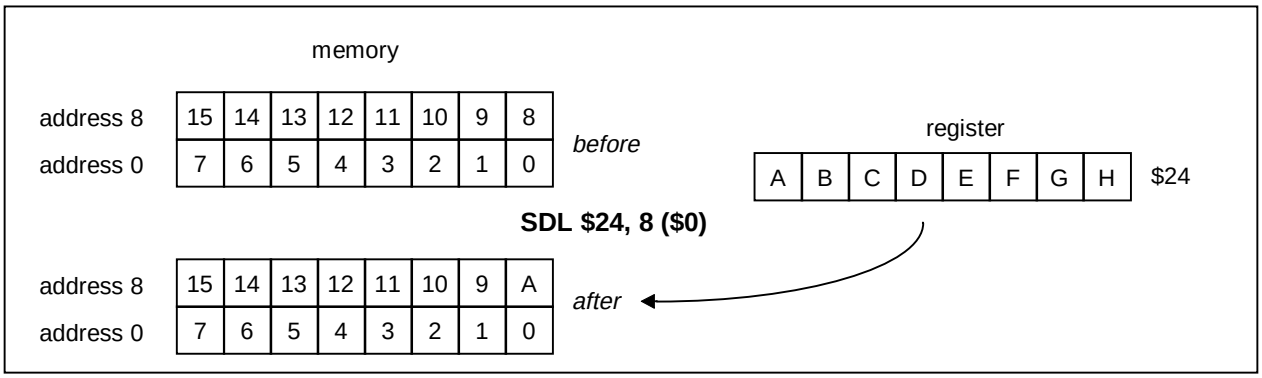
**Description:**

This instruction can be used with the SDR instruction to store the contents of a register into eight consecutive bytes of memory, when the bytes cross a doubleword boundary. SDL stores the register into the appropriate part of the high-order doubleword of memory; SDR stores the register into the appropriate part of the low-order doubleword.

The SDL instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address that may specify an arbitrary byte. It alters only the word in memory that contains that byte. From one to four bytes will be stored, depending on the starting byte specified.

Conceptually, it starts at the most-significant byte of the register and copies it to the specified byte in memory; then it copies bytes from register to memory until it reaches the low-order byte of the word in memory.

No address error exceptions due to alignment are possible.



# SDL Store Doubleword Left (2/3) SDL

An address error exception is not occurred that specify address is not located in doubleword boundary.

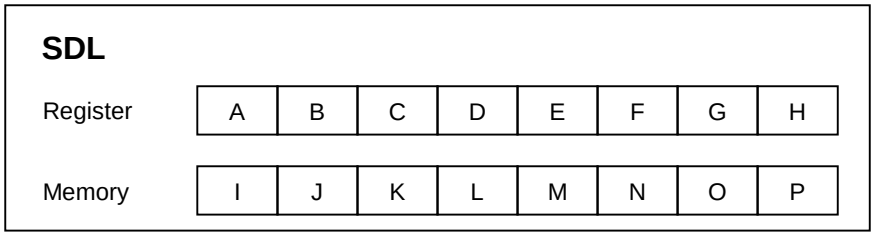
This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

**Operation:**

```
64   T:   vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
          (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
          pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor ReverseEndian^3)
          if BigEndianMem = 0 then
               pAddr ← pAddrPSIZE - 1...3 || 0^3
          endif
          byte ← vAddr2...0 xor BigEndianCPU^3
          data ← 0^56 – 8 * byte || GPR [rt]63...56 – 8 * byte
          StoreMemory (uncached, byte, data, pAddr, vAddr, DATA)
```

# SDL Store Doubleword Left (3/3) SDL

Given a doubleword in a register and a doubleword in memory, the operation of SDL instruction is as follows:



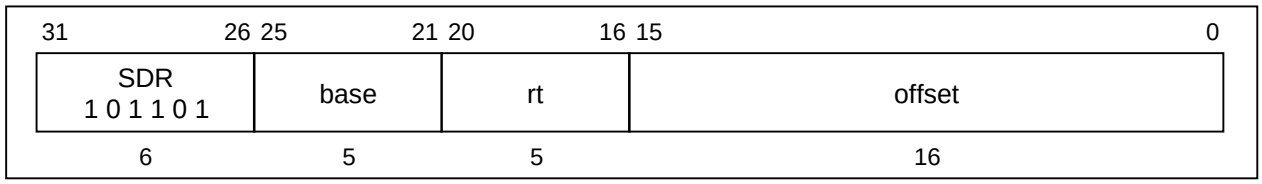

| vAddr2..0 | Destination | Type | Offset (LEM) |
|---|---|---|---|
| 0 | I J K L M N O A | 0 | 0 |
| 1 | I J K L M N A B | 1 | 0 |
| 2 | I J K L M A B C | 2 | 0 |
| 3 | I J K L A B C D | 3 | 0 |
| 4 | I J K A B C D E | 4 | 0 |
| 5 | I J A B C D E F | 5 | 0 |
| 6 | I A B C D E F G | 6 | 0 |
| 7 | A B C D E F G H | 7 | 0 |

**Remark** *LEM* Little-endian memory (BigEndianMem = 0)
*Type* AccessType (see **Table 2-3. Byte Specification Related to Load and Store Instructions**) sent to memory
*Offset* pAddr2..0 sent to memory

**Exceptions:**

TLB refill exception
TLB invalid exception
TLB modification exception
Bus error exception
Address error exception
Reserved instruction exception (32-bit user mode/supervisor mode)

# SDR Store Doubleword Right (1/3) SDR




**Format:**

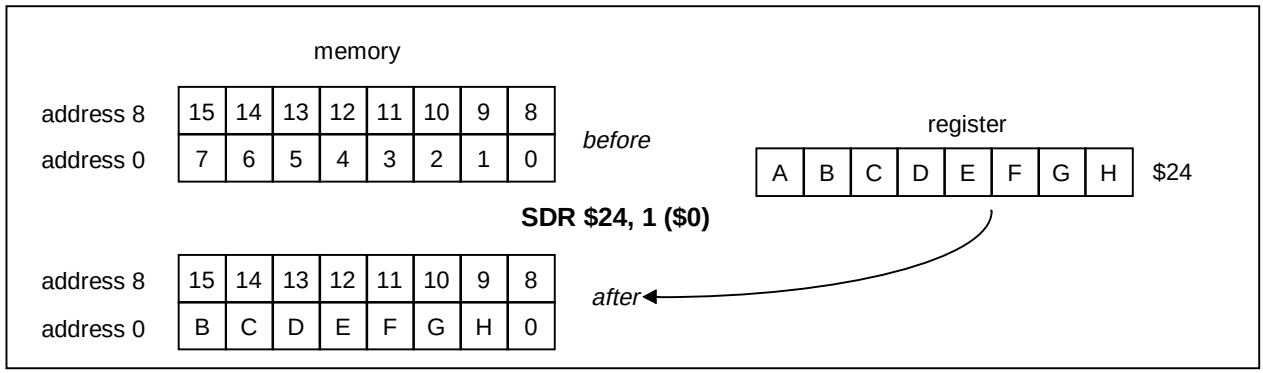
SDR rt, offset (base)

**Description:**

This instruction can be used with the SDL instruction to store the contents of a register into eight consecutive bytes of memory, when the bytes cross a boundary between two doublewords. SDR stores the register into the appropriate part of the low-order doubleword; SDL stores the register into the appropriate part of the low-order doubleword of memory.

The SDR instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address that may specify an arbitrary byte. It alters only the word in memory that contains that byte. From one to eight bytes will be stored, depending on the starting byte specified.

Conceptually, it starts at the least-significant byte of the register and copies it to the specified byte in memory; then it copies bytes from register to memory until it reaches the high-order byte of the word in memory. No address error exceptions due to alignment are possible.



# SDR Store Doubleword Right (2/3) SDR

An address error exception is not occurred that specify address is not located in doubleword boundary.

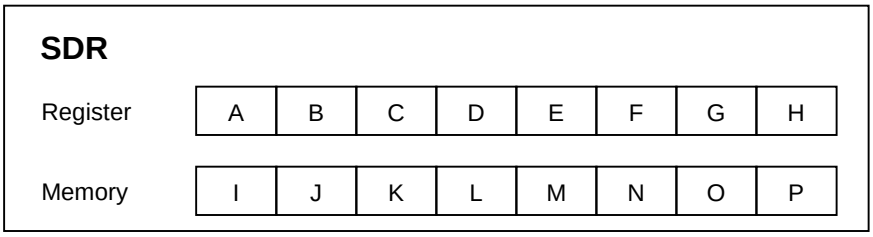
This operation is defined in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

**Operation:**

```
64   T:   vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
          (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
          pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor ReverseEndian^3)
          if BigEndianMem = 0 then
               pAddr ← pAddrPSIZE - 1...3 || 0^3
          endif
          byte ← vAddr2...0 xor BigEndianCPU^3
          data ← GPR [rt]63 – 8 * byte || 0^8 * byte
          StoreMemory (uncached, DOUBLEWORD-byte, data, pAddr, vAddr, DATA)
```

# SDR Store Doubleword Right (3/3) SDR

Given a doubleword in a register and a doubleword in memory, the operation of SDR instruction is as follows:



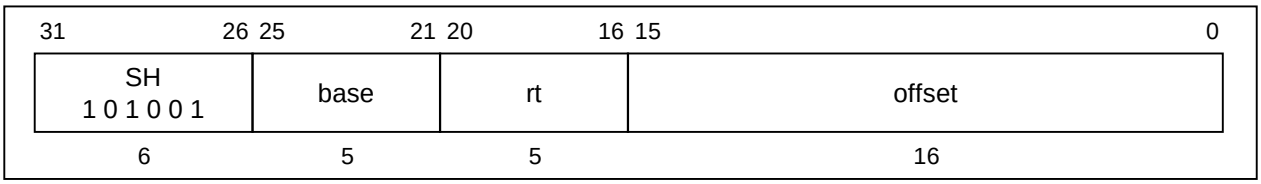

| vAddr2..0 | Destination | Type | Offset (LEM) |
|---|---|---|---|
| 0 | A B C D E F G H | 7 | 0 |
| 1 | B C D E F G H P | 6 | 1 |
| 2 | C D E F G H O P | 5 | 2 |
| 3 | D E F G H N O P | 4 | 3 |
| 4 | E F G H M N O P | 3 | 4 |
| 5 | F G H L M N O P | 2 | 5 |
| 6 | G H K L M N O P | 1 | 6 |
| 7 | H J K L M N O P | 0 | 7 |

**Remark** *LEM* Little-endian memory (BigEndianMem = 0)
*Type* AccessType (see **Table 2-3. Byte Specification Related to Load and Store Instructions**) sent to memory
*Offset* pAddr2..0 sent to memory

**Exceptions:**

TLB refill exception
TLB invalid exception
TLB modification exception
Bus error exception
Address error exception
Reserved instruction exception (32-bit user mode/supervisor mode)

# SH Store Halfword SH




**Format:**

SH rt, offset (base)

**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form an unsigned effective address. The least-significant halfword of register *rt* is stored at the effective address. If the least-significant bit of the effective address is non-zero, an address error exception occurs.

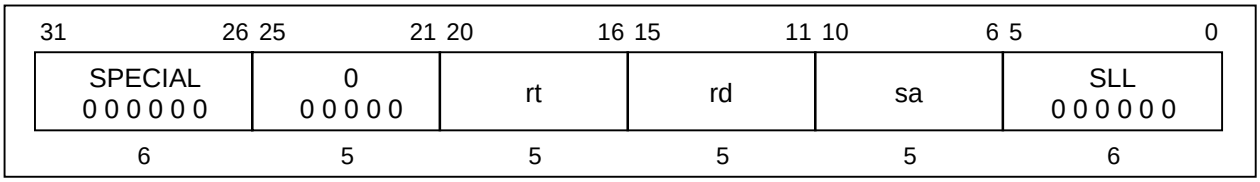
**Operation:**

```
32    T:   vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor (ReverseEndian^2 || 0))
           byte ← vAddr2...0 xor (BigEndianCPU^2 || 0)
           data ← GPR [rt]63 – 8 * byte...0 || 0^8 * byte
           StoreMemory (uncached, HALFWORD, data, pAddr, vAddr, DATA)

64    T:   vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor (ReverseEndian^2 || 0))
           byte ← vAddr2...0 xor (BigEndianCPU^2 || 0)
           data ← GPR [rt]63 – 8 * byte...0 || 0^8 * byte
           StoreMemory (uncached, HALFWORD, data, pAddr, vAddr, DATA)
```

**Exceptions:**

TLB refill exception
TLB invalid exception
TLB modification exception
Bus error exception
Address error exception

# SLL — Shift Left Logical — SLL




**Format:**

SLL rd, rt, sa

**Description:**

The contents of general register *rt* are shifted left by *sa* bits, inserting zeros into the low-order bits.

The result is placed in register *rd.*

In 64-bit mode, the 32-bit result is sign-extended when placed in the destination register. It is sign extended for all shift amounts, including zero; SLL with zero shift amount truncates a 64-bit value to 32 bits and then sign extends this 32-bit value. SLL, unlike nearly all other word operations, does not require an operand to be a properly sign-extended word value to produce a valid sign-extended word result.

**Operation:**

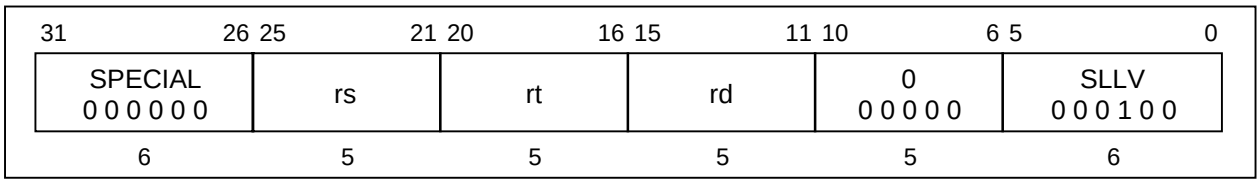
```
32   T:   GPR[rd] ← GPR[rt]31 - sa...0 || 0^sa

64   T:   s ← 0 || sa
         temp ← GPR[rt]31 - s...0 || 0^s
         GPR[rd] ← (temp31)^32 || temp
```

**Exceptions:**

None

**Caution** **SLL with a shift amount of zero may be treated as a NOP by some assemblers, at some optimization levels. If using SLL with a purpose of sign-extension, check the assembler specification.**

# SLLV — Shift Left Logical Variable — SLLV




**Format:**

SLLV rd, rt, rs

**Description:**

The contents of general register *rt* are shifted left the number of bits specified by the low-order five bits contained in general register *rs*, inserting zeros into the low-order bits.

The result is placed in register *rd*.

In 64-bit mode, the 32-bit result is sign-extended when placed in the destination register. It is sign extended for all shift amounts, including zero; SLLV with zero shift amount truncates a 64-bit value to 32 bits and then sign extends this 32-bit value. SLLV, unlike nearly all other word operations, does not require an operand to be a properly sign-extended word value to produce a valid sign-extended word result.

**Operation:**

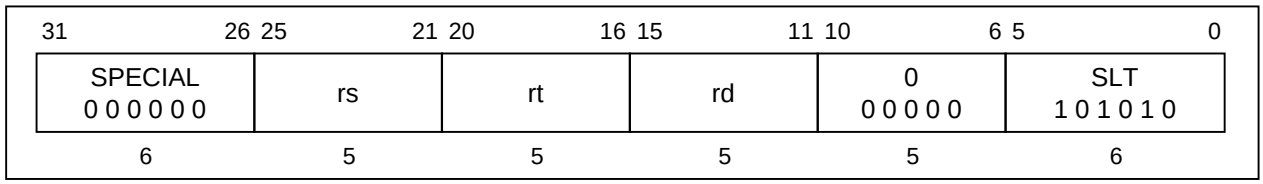
```
32    T:   s ← GPR [rs]4...0
           GPR [rd] ← GPR [rt](31 - s)...0 || 0^s

64    T:   s ← 0 || GPR [rs]4...0
           temp ← GPR [rt](31 - s)...0 || 0^s
           GPR [rd] ← (temp31)^32 || temp
```

**Exceptions:**

None

**Caution** **SLLV with a shift amount of zero may be treated as a NOP by some assemblers, at some optimization levels. If using SLLV with a purpose of sign-extension, check the assembler specification.**

# SLT — Set On Less Than — SLT




**Format:**

SLT rd, rs, rt

**Description:**

The contents of general register *rt* are subtracted from the contents of general register *rs*. Considering both quantities as signed integers, if the contents of general register *rs* are less than the contents of general register *rt*, the result is set to one; otherwise the result is set to zero.

No integer overflow exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

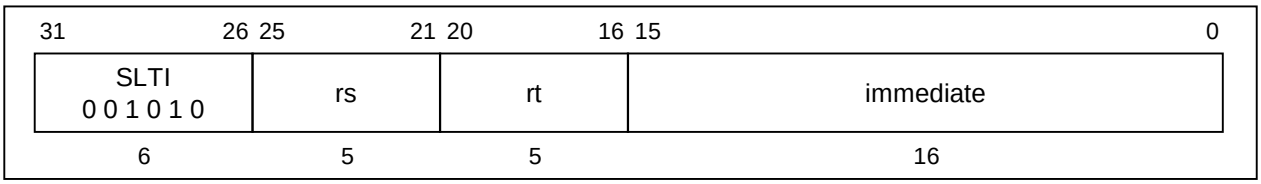
**Operation:**

```
32    T:    if GPR [rs] < GPR [rt] then
                GPR [rd] ← 0^31 || 1
            else
                GPR [rd] ← 0^32
            endif

64    T:    if GPR [rs] < GPR [rt] then
                GPR [rd] ← 0^63 || 1
            else
                GPR [rd] ← 0^64
            endif
```

**Exceptions:**

None

# SLTI Set On Less Than Immediate SLTI




**Format:**

SLTI rt, rs, immediate

**Description:**

The 16-bit *immediate* is sign-extended and subtracted from the contents of general register *rs.* Considering both quantities as signed integers, if *rs* is less than the sign-extended *immediate*, the result is set to 1; otherwise the result is set to 0.

No integer overflow exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

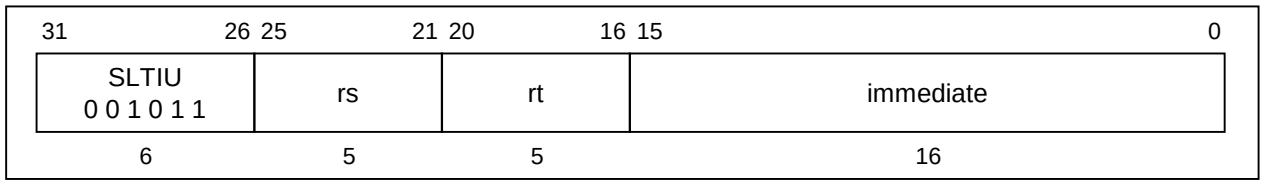
**Operation:**

```
32    T:   if GPR [rs] < (immediate15) 16 || immediate15...0 then
               GPR [rt] ← 031 || 1
           else
               GPR [rt] ← 032
           endif

64    T:   if GPR [rs] < (immediate15) 48 || immediate 15...0 then
               GPR [rt] ← 063 || 1
           else
               GPR [rt] ← 064
           endif
```

**Exceptions:**

None

# SLTIU Set On Less Than Immediate Unsigned SLTIU




**Format:**

SLTIU rt, rs, immediate

**Description:**

The 16-bit *immediate* is sign-extended and subtracted from the contents of general register *rs.* Considering both quantities as unsigned integers, if *rs* is less than the sign-extended *immediate*, the result is set to 1; otherwise the result is set to 0.

No integer overflow exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

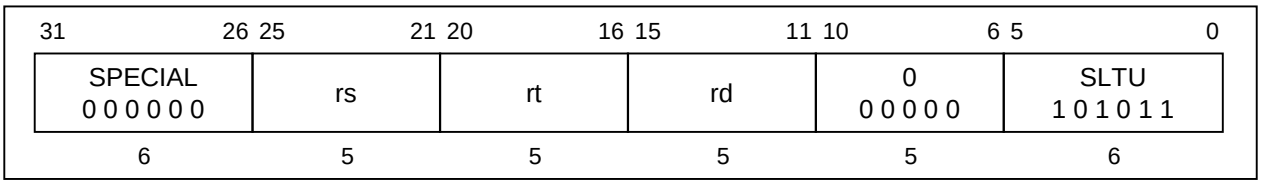
**Operation:**

```
32    T:    if (0 || GPR [rs]) < (0 || (immediate15) 16 || immediate15...0) then
                  GPR [rt] ← 0^31 || 1
            else
                  GPR [rt] ← 0^32
            endif

64    T:    if (0 || GPR [rs]) < (0 || (immediate15) 48 || immediate15...0) then
                  GPR [rt] ← 0^63 || 1
            else
                  GPR [rt] ← 0^64
            endif
```

**Exceptions:**

None

# SLTU Set On Less Than Unsigned SLTU




**Format:**

SLTU rd, rs, rt

**Description:**

The contents of general register *rt* are subtracted from the contents of general register *rs.* Considering both quantities as unsigned integers, if the contents of general register *rs* are less than the contents of general register *rt*, the result is set to 1; otherwise the result is set to 0.

No integer overflow exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

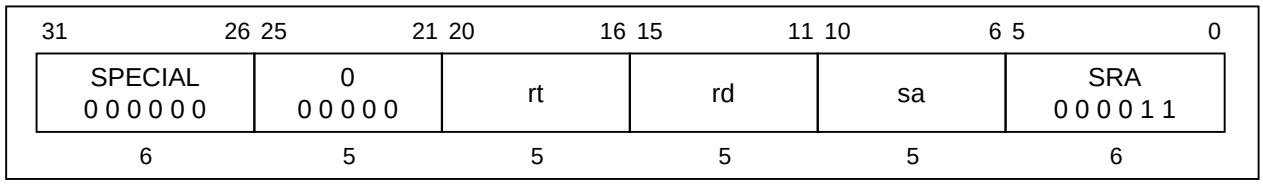
**Operation:**

```
32    T:    if (0 || GPR [rs]) < 0 || GPR [rt] then
                GPR [rd] ← 0^31 || 1
            else
                GPR [rd] ← 0^32
            endif

64    T:    if (0 || GPR [rs]) < 0 || GPR [rt] then
                GPR [rd] ← 0^63 || 1
            else
                GPR [rd] ← 0^64
            endif
```

**Exceptions:**

None

# SRA Shift Right Arithmetic SRA




**Format:**

SRA rd, rt, sa

**Description:**

The contents of general register *rt* are shifted right by *sa* bits, sign-extending the high-order bits.

The result is placed in register *rd*.

In 64-bit mode, the operand must be a valid sign-extended, 32-bit value.

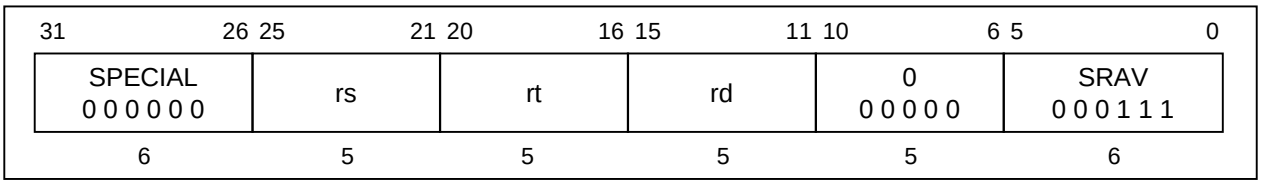
**Operation:**

```
32   T:   GPR [rd] ← (GPR [rt] 31)^sa || GPR [rt]31...sa

64   T:   s ← 0 || sa
          temp ← (GPR [rt] 31)^s || GPR [rt]31...s
          GPR [rd] ← (temp 31)^32 || temp
```

**Exceptions:**

None

# SRAV Shift Right Arithmetic Variable SRAV




**Format:**

SRAV rd, rt, rs

**Description:**

The contents of general register *rt* are shifted right by the number of bits specified by the low-order five bits of general register *rs*, sign-extending the high-order bits.

The result is placed in register *rd*.

In 64-bit mode, the operand must be a valid sign-extended, 32-bit value.

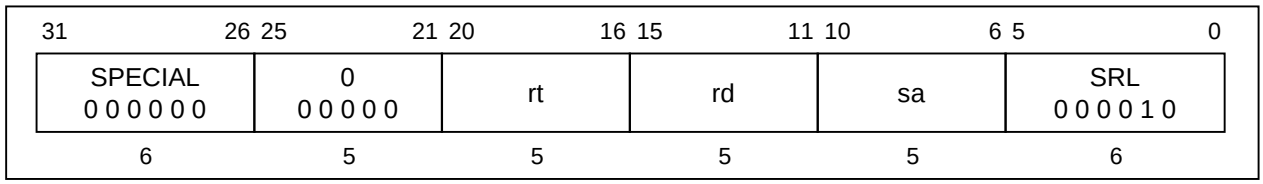
**Operation:**

```
32    T:   s ← GPR [rs]4...0
           GPR [rd] ← (GPR [rt]31)^s || GPR [rt]31...s

64    T:   s ← GPR [rs]4...0
           temp ← (GPR [rt]31)^s || GPR [rt]31...s
           GPR [rd] ← (temp31)^32 || temp
```

**Exceptions:**

None

# SRL — Shift Right Logical — SRL




**Format:**

SRL rd, rt, sa

**Description:**

The contents of general register *rt* are shifted right by *sa* bits, inserting zeros into the high-order bits.

The result is placed in register *rd*.

In 64-bit mode, the operand must be a valid sign-extended, 32-bit value.

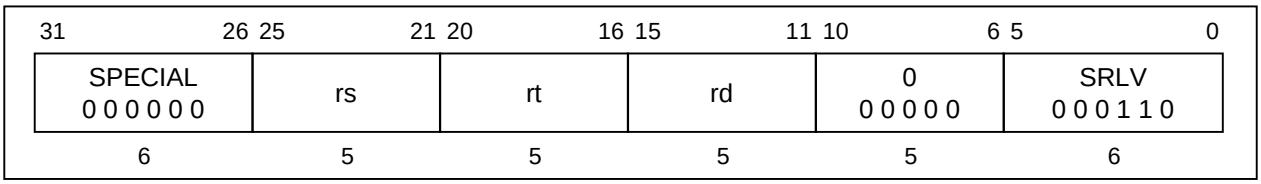
**Operation:**

```
32   T:   GPR [rd] ← 0^sa || GPR [rt]31...sa

64   T:   s ← 0 || sa
          temp ← 0^s || GPR [rt]31...s
          GPR [rd] ← (temp31)^32 || temp
```

**Exceptions:**

None

# SRLV Shift Right Logical Variable SRLV




**Format:**

SRLV rd, rt, rs

**Description:**

The contents of general register *rt* are shifted right by the number of bits specified by the low-order five bits of general register *rs,* inserting zeros into the high-order bits.

The result is placed in register *rd*.

In 64-bit mode, the operand must be a valid sign-extended, 32-bit value.

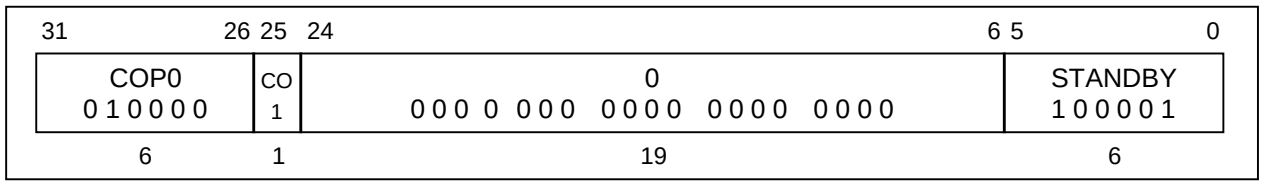
**Operation:**

```
32    T:   s ← GPR [rs]4...0
           GPR [rd] ← 0^s || GPR [rt]31...s

64    T:   s ← GPR [rs]4...0
           temp ← 0^s || GPR [rt]31...s
           GPR [rd] ← (temp31)^32 || temp
```

**Exceptions:**

None

# STANDBY Standby STANDBY




**Format:**

STANDBY

**Description:**

STANDBY instruction starts mode transition from Fullspeed mode to Standby mode.

When the STANDBY instruction finishes the WB stage, this processor wait by the SysAD bus is idle state, after then the internal clocks will shut down, thus freezing the pipeline. The PLL, Timer/Interrupt clocks and the internal bus clocks (TClock and MasterOut) will continue to run.

Once this processor is in Standby mode, any interrupt, including the internally generated timer interrupt, NMI, Soft Reset, and Cold Reset will cause this processor to exit Standby mode and to enter Fullspeed mode.

**Operation:**

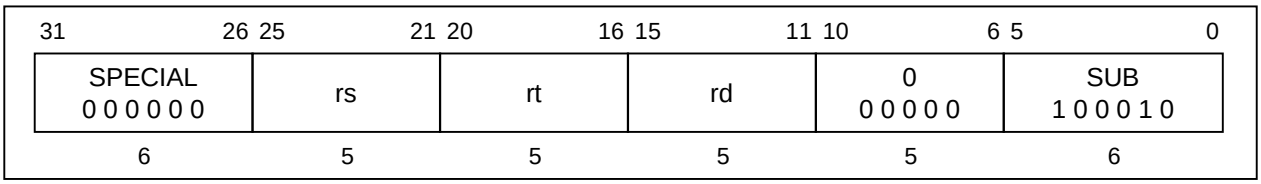
```
32, 64  T:
        T+1:  Standby operation ()
```

**Exceptions:**

Coprocessor unusable exception

**Remark** Refer to **Section 2.7.3.1 Power modes** for details about the operation of the peripheral units at mode transition.

# SUB — Subtract — SUB




**Format:**

SUB rd, rs, rt

**Description:**

The contents of general register *rt* are subtracted from the contents of general register *rs* to form a result. The result is placed into general register *rd.* In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

The only difference between this instruction and the SUBU instruction is that SUBU never traps on overflow.

An integer overflow exception takes place if the carries out of bits 30 and 31 differ (2's complement overflow). The destination register *rd* is not modified when an integer overflow exception occurs.

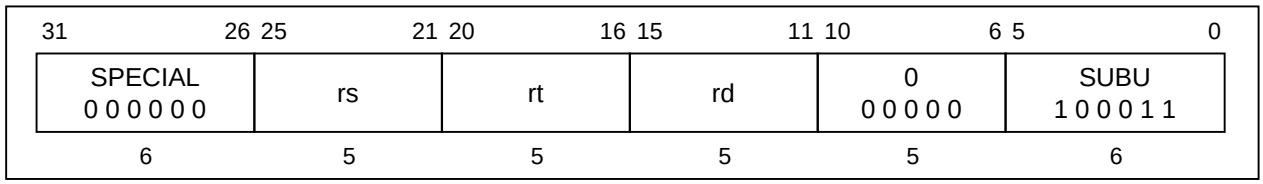
**Operation:**

```
32    T:    GPR [rd] ← GPR [rs] - GPR [rt]

64    T:    temp ← GPR [rs] - GPR [rt]
            GPR [rd] ← (temp31)^32 || temp31...0
```

**Exceptions:**

Integer overflow exception

# SUBU Subtract Unsigned SUBU




**Format:**

SUBU rd, rs, rt

**Description:**

The contents of general register *rt* are subtracted from the contents of general register *rs* to form a result.

The result is placed into general register *rd*.

In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

The only difference between this instruction and the SUB instruction is that SUBU never traps on overflow.

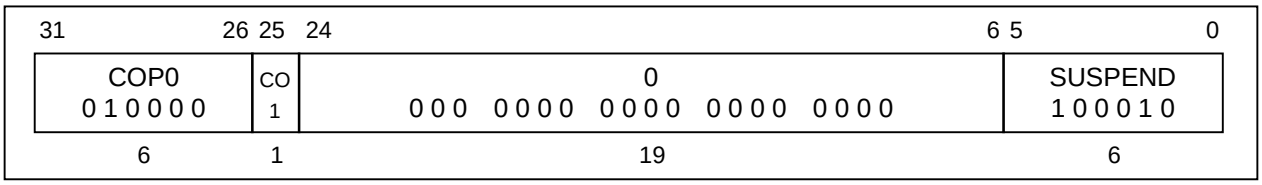
**Operation:**

```
32   T:   GPR [rd] ← GPR [rs] - GPR [rt]

64   T:   temp ← GPR [rs] - GPR [rt]
          GPR [rd] ← (temp31)^32 || temp31...0
```

**Exceptions:**

None

# SUSPEND Suspend SUSPEND




**Format:**

SUSPEND

**Description:**

SUSPEND instruction starts mode transition from Fullspeed mode to Suspend mode.

When the SUSPEND instruction finishes the WB stage, this processor wait by the SysAD bus is idle state, after then the internal clocks including the TClock will shut down, thus freezing the pipeline. The PLL, Timer/Interrupt clocks and MasterOut, will continue to run.

Once this processor is in Suspend mode, any interrupt, including the internally generated timer interrupt, NMI, Soft Reset and Cold Reset will cause this processor to exit Suspend mode and to enter Fullspeed mode.

**Operation:**

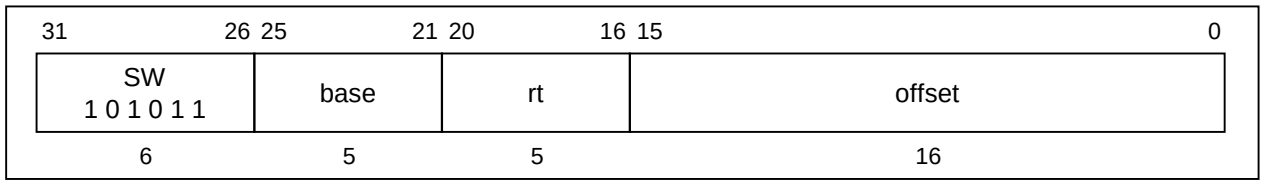
```
32, 64  T:
        T+1:  Suspend Operation ()
```

**Exceptions:**

Coprocessor unusable exception

**Remark** Refer to **Section 2.7.3.1 Power modes** for details about the operation of the peripheral units at mode transition.

# SW Store Word SW




**Format:**

SW rt, offset (base)

**Description:**

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The contents of general register *rt* are stored at the memory location specified by the effective address.

If either of the two least-significant bits of the effective address are non-zero, an address error exception occurs.

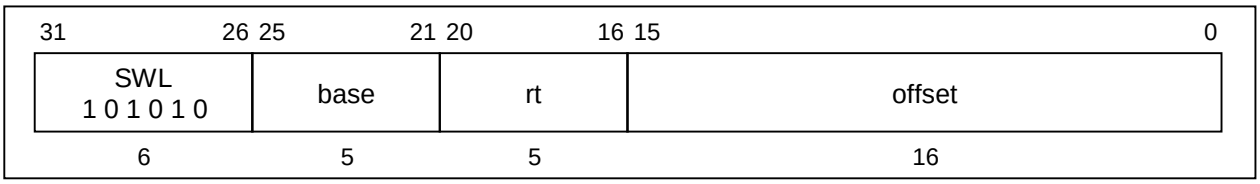
**Operation:**

```
32    T:   vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor (ReverseEndian || 0^2))
           byte ← vAddr2...0 xor (BigEndianCPU || 0^2)
           data ← GPR [rt]63 – 8 * byte || 0^8 * byte
           StoreMemory (uncached, WORD, data, pAddr, vAddr, DATA)

64    T:   vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor (ReverseEndian || 0^2))
           byte ← vAddr2...0 xor (BigEndianCPU || 0^2)
           data ← GPR [rt]63 – 8 * byte || 0^8 * byte
           StoreMemory (uncached, WORD, data, pAddr, vAddr, DATA)
```

**Exceptions:**

TLB refill exception
TLB invalid exception
TLB modification exception
Bus error exception
Address error exception

# SWL Store Word Left (1/3) SWL




**Format:**

SWL rt, offset (base)

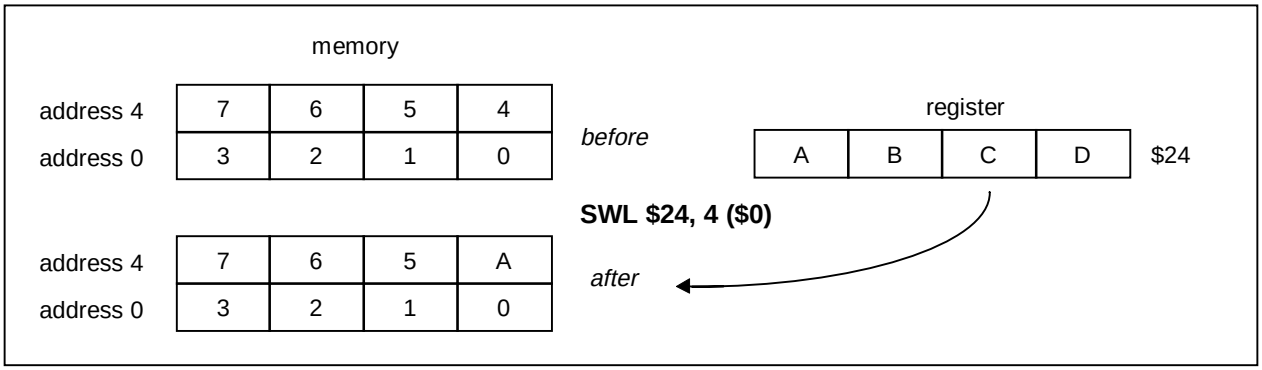
**Description:**

This instruction can be used with the SWR instruction to store the contents of a register into four consecutive bytes of memory, when the bytes cross a word boundary. SWL stores the register into the appropriate part of the high-order word of memory; SWR stores the register into the appropriate part of the low-order word.

The SWL instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address that may specify an arbitrary byte. It alters only the word in memory that contains that byte. From one to four bytes will be stored, depending on the starting byte specified.

Conceptually, it starts at the most-significant byte of the register and copies it to the specified byte in memory; then it copies bytes from register to memory until it reaches the low-order byte of the word in memory.

No address error exceptions due to alignment are possible.



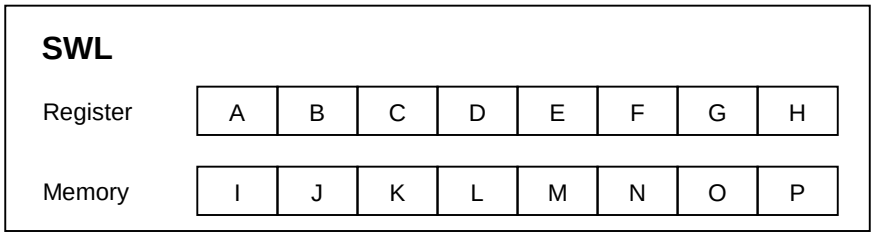
# SWL Store Word Left (2/3) SWL

**Operation:**

```
32    T:   vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor ReverseEndian^3)
           if BigEndianMem = 0 then
                pAddr ← pAddrPSIZE - 1...2 || 0^2
           endif
           byte ← vAddr1...0 xor BigEndianCPU^2
           if (vAddr2 xor BigEndianCPU) = 0 then
                data ← 0^32 || 0^24 – 8 * byte || GPR [rt]31...24 – 8 * byte
           else
                data ← 0^24 – 8 * byte || GPR [rt]31...24 – 8 * byte || 0^32
           endif
           StoreMemory (uncached, byte, data, pAddr, vAddr, DATA)

64    T:   vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor ReverseEndian^3)
           if BigEndianMem = 0 then
                pAddr ← pAddrPSIZE - 1...2 || 0^2
           endif
           byte ← vAddr1...0 xor BigEndianCPU^2
           if (vAddr2 xor BigEndianCPU) = 0 then
                data ← 0^32 || 0^24 – 8 * byte || GPR [rt]31...24 – 8 * byte
           else
                data ← 0^24 – 8 * byte || GPR [rt]31...24 – 8 * byte || 0^32
           endif
           StoreMemory (uncached, byte, data, pAddr, vAddr, DATA)
```

# SWL Store Word Left (3/3) SWL

Given a doubleword in a register and a doubleword in memory, the operation of SWL is as follows:



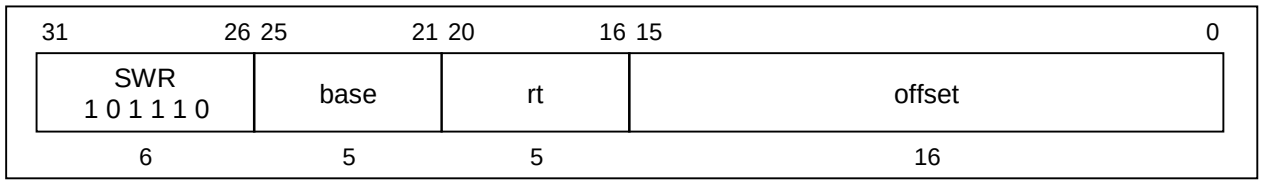

| vAddr2..0 | Destination | Type | Offset (LEM) |
|---|---|---|---|
| 0 | I J K L M N O E | 0 | 0 |
| 1 | I J K L M N E F | 1 | 0 |
| 2 | I J K L M E F G | 2 | 0 |
| 3 | I J K L E F G H | 3 | 0 |
| 4 | I J K E M N O P | 0 | 4 |
| 5 | I J E F M N O P | 1 | 4 |
| 6 | I E F G M N O P | 2 | 4 |
| 7 | E F G H M N O P | 3 | 4 |

**Remark** *LEM* Little-endian memory (BigEndianMem = 0)
*Type* AccessType (see **Table 2-3. Byte Specification Related to Load and Store Instructions**) sent to memory
*Offset* pAddr2..0 sent to memory

**Exceptions:**

TLB refill exception
TLB invalid exception
TLB modification exception
Bus error exception
Address error exception

# SWR Store Word Right (1/3) SWR




**Format:**

SWR rt, offset (base)

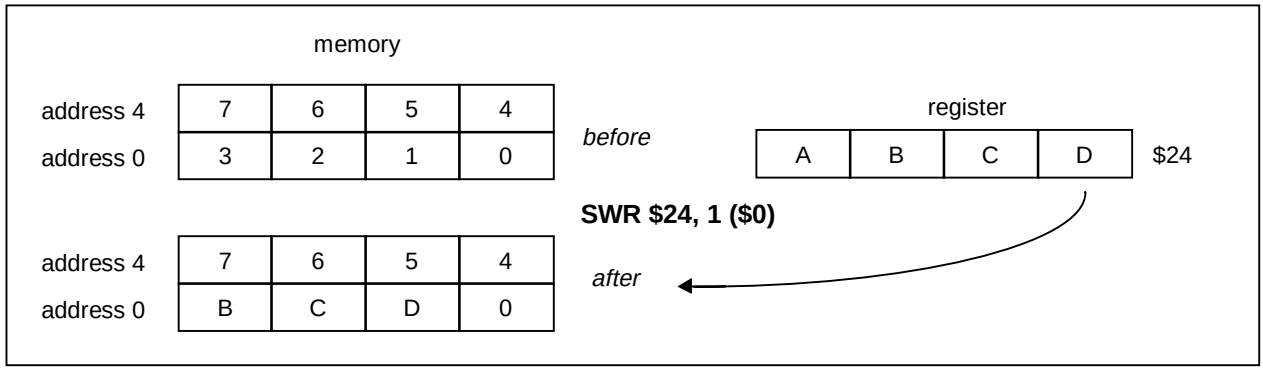
**Description:**

This instruction can be used with the SWL instruction to store the contents of a register into four consecutive bytes of memory, when the bytes cross a boundary between two words. SWR stores the register into the appropriate part of the low-order word; SWL stores the register into the appropriate part of the low-order word of memory.

The SWR instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address that may specify an arbitrary byte. It alters only the word in memory that contains that byte. From one to four bytes will be stored, depending on the starting byte specified.

Conceptually, it starts at the least-significant (rightmost) byte of the register and copies it to the specified byte in memory; then copies bytes from register to memory until it reaches the high-order byte of the word in memory.

No address error exceptions due to alignment are possible.



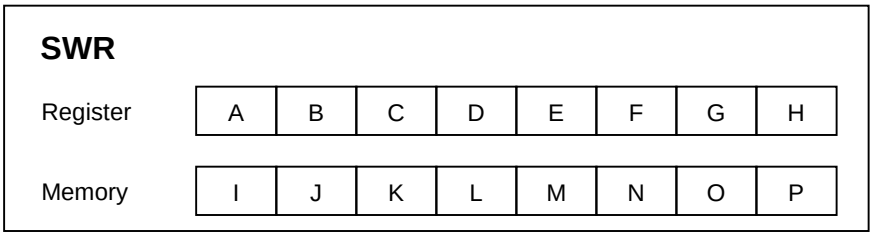
# SWR Store Word Right (2/3) SWR

**Operation:**

```
32    T:   vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor ReverseEndian^3)
           if BigEndianMem = 1 then
                pAddr ← pAddrPSIZE - 1...2 || 0^2
           endif
           byte ← vAddr1...0 xor BigEndianCPU^2
           if (vAddr2 xor BigEndianCPU) = 0 then
                data ← 0^32 || GPR [rt]31 – 8 * byte...0 || 0^8 * byte
           else
                data ← GPR [rt]31 – 8 * byte || 0^8 * byte || 0^32
           endif
           StoreMemory (uncached, WORD-byte, data, pAddr, vAddr, DATA)

64    T:   vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE - 1...3 || (pAddr2...0 xor ReverseEndian^3)
           if BigEndianMem = 1 then
                pAddr ← pAddrPSIZE - 1...2 || 0^2
           endif
           byte ← vAddr1...0 xor BigEndianCPU^2
           if (vAddr2 xor BigEndianCPU) = 0 then
                data ← 0^32 || GPR [rt]31 – 8 * byte...0 || 0^8 * byte
           else
                data ← GPR [rt]31 – 8 * byte || 0^8 * byte || 0^32
           endif
           StoreMemory (uncached, WORD-byte, data, pAddr, vAddr, DATA)
```

# SWR Store Word Right (3/3) SWR

Given a doubleword in a register and a doubleword in memory, the operation of SWR instruction is as follows:




| vAddr2..0 | Destination | Type | Offset (LEM) |
|---|---|---|---|
| 0 | I J K L E F G H | 3 | 0 |
| 1 | I J K L F G H P | 2 | 1 |
| 2 | I J K L G H O P | 1 | 2 |
| 3 | I J K L H N O P | 0 | 3 |
| 4 | E F G H M N O P | 3 | 4 |
| 5 | F G H L M N O P | 2 | 5 |
| 6 | G H K L M N O P | 1 | 6 |
| 7 | H J K L M N O P | 0 | 7 |

**Remark** *LEM* Little-endian memory (BigEndianMem = 0)

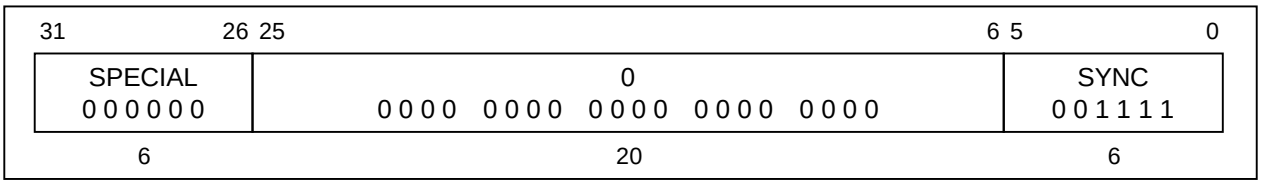
*Type* AccessType (see **Table 2-3. Byte Specification Related to Load and Store Instructions**) sent to memory

*Offset* pAddr2..0 sent to memory

**Exceptions:**

TLB refill exception
TLB invalid exception
TLB modification exception
Bus error exception
Address error exception

# SYNC Synchronize SYNC




**Format:**

SYNC

**Description:**

The SYNC instruction is executed as a NOP on the VR4121. This operation maintains compatibility with code compiled for the VR4000.

This instruction is defined to maintain the compatibility with VR4000 and VR4400.

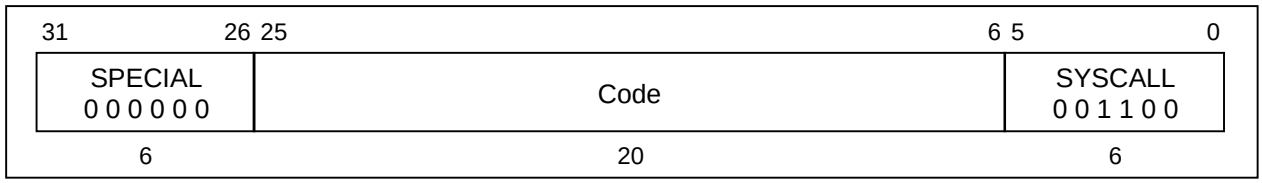
**Operation:**

```
32, 64 T:    SyncOperation ( )
```

**Exceptions:**

None

# SYSCALL System Call SYSCALL




**Format:**

SYSCALL

**Description:**

A system call exception occurs, immediately and unconditionally transferring control to the exception handler. The code field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

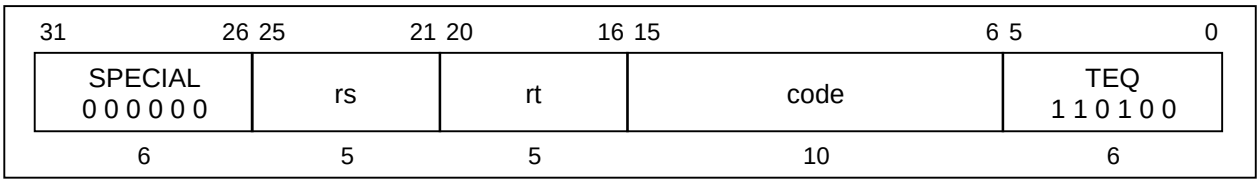
**Operation:**

```
32, 64 T:    SystemCallException
```

**Exceptions:**

System Call exception

# TEQ — Trap If Equal — TEQ




**Format:**

TEQ rs, rt

**Description:**

The contents of general register *rt* are compared to general register *rs*. If the contents of general register *rs* are equal to the contents of general register *rt*, a trap exception occurs.

The code field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

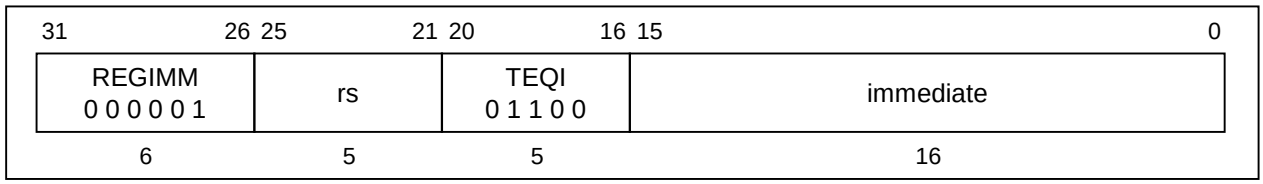
**Operation:**

```
32, 64  T:    if GPR [rs] = GPR [rt] then
                    TrapException
              endif
```

**Exceptions:**

Trap exception

# TEQI Trap If Equal Immediate TEQI




**Format:**

TEQI rs, immediate

**Description:**

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. If the contents of general register *rs* are equal to the sign-extended *immediate*, a trap exception occurs.

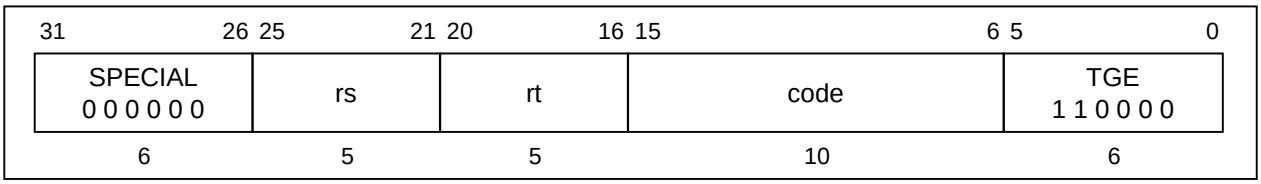
**Operation:**

```
32    T:    if GPR [rs] = (immediate15)^16 || immediate15...0 then
                  TrapException
            endif

64    T:    if GPR [rs] = (immediate15)^48 || immediate15...0 then
                  TrapException
            endif
```

**Exceptions:**

Trap exception

# TGE Trap If Greater Than Or Equal TGE




**Format:**

TGE rs, rt

**Description:**

The contents of general register *rt* are compared to the contents of general register *rs*. Considering both quantities as signed integers, if the contents of general register *rs* are greater than or equal to the contents of general register *rt*, a trap exception occurs.

The code field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

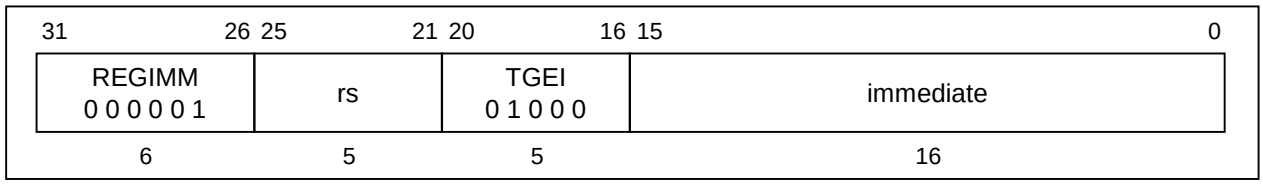
**Operation:**

```
32, 64  T:    if GPR [rs] ≥ GPR [rt] then
                  TrapException
              endif
```

**Exceptions:**

Trap exception

# TGEI Trap If Greater Than Or Equal Immediate TGEI




**Format:**

TGEI rs, immediate

**Description:**

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. Considering both quantities as signed integers, if the contents of general register *rs* are greater than or equal to the sign-extended *immediate*, a trap exception occurs.

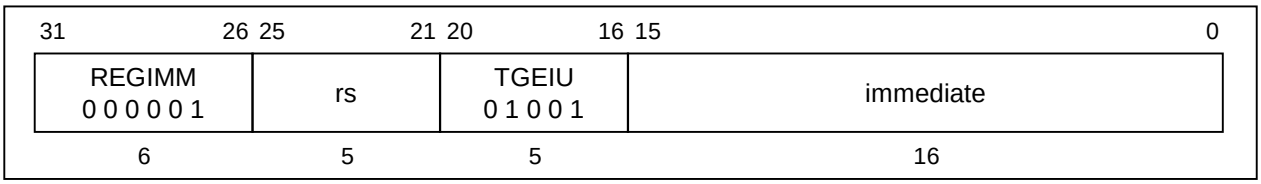
**Operation:**

```
32    T:    if GPR [rs] ≥ (immediate15)16 || immediate15...0 then
                  TrapException
            endif

64    T:    if GPR [rs] ≥ (immediate15)48 || immediate15...0 then
                  TrapException
            endif
```

**Exceptions:**

Trap exception

# TGEIU Trap If Greater Than Or Equal Immediate Unsigned TGEIU




**Format:**

TGEIU rs, immediate

**Description:**

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. Considering both quantities as unsigned integers, if the contents of general register *rs* are greater than or equal to the sign-extended *immediate*, a trap exception occurs.

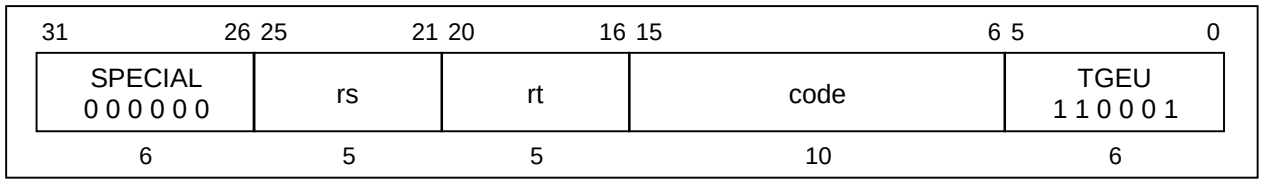
**Operation:**

```
32    T:    if (0 || GPR [rs]) ≥ (0 || (immediate15)^16 || immediate15...0) then
                  TrapException
            endif

64    T:    if (0 || GPR [rs]) ≥ (0 || (immediate15)^48 || immediate15...0) then
                  TrapException
            endif
```

**Exceptions:**

Trap exception

# TGEU Trap If Greater Than Or Equal Unsigned TGEU




**Format:**

TGEU rs, rt

**Description:**

The contents of general register *rt* are compared to the contents of general register *rs*. Considering both quantities as unsigned integers, if the contents of general register *rs* are greater than or equal to the contents of general register *rt*, a trap exception occurs.

The code field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

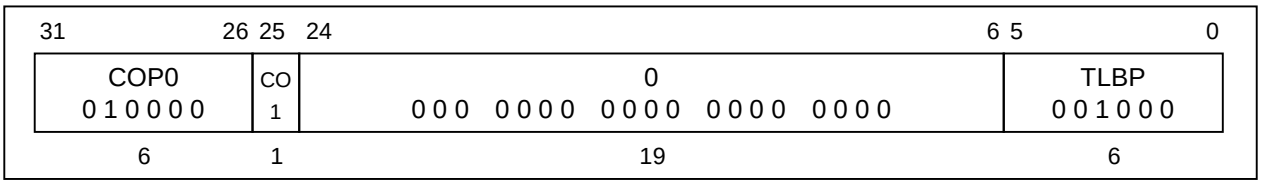
**Operation:**

```
32, 64  T:    if (0 || GPR [rs]) ≥ (0 || GPR [rt]) then
                    TrapException
              endif
```

**Exceptions:**

Trap exception

# TLBP Probe TLB For Matching Entry TLBP




**Format:**

TLBP

**Description:**

The Index register is loaded with the address of the TLB entry whose contents match the contents of the EntryHi register. If no TLB entry matches, the high-order bit of the Index register is set.

The architecture does not specify the operation of memory references associated with the instruction immediately after a TLBP instruction, nor is the operation specified if more than one TLB entry matches.

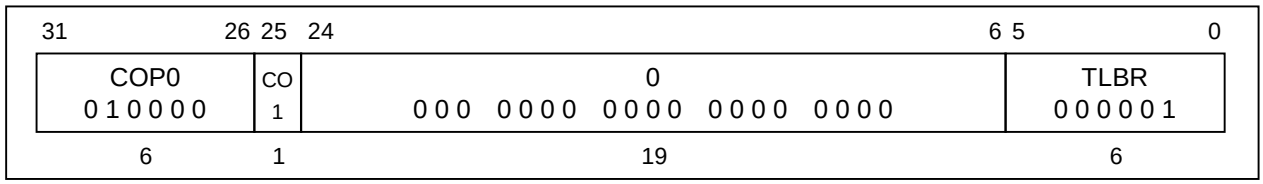
**Operation:**

```
32    T:    Index ← 1 || 0^25 || Undefined^6
            for i in 0...TLBEntries - 1
                  if (TLB [i]95...77 = EntryHi31...13) and (TLB [i]76 or
                  (TLB [i]71...64 = EntryHi7...0)) then
                        Index ← 0^26 || i5...0
                  endif
            endfor

64    T:    Index ← 1 || 0^25 || Undefined^6
            for i in 0...TLBEntries - 1
                  if (TLB [i]167...141 and not (0^15 || TLB [i]216...205))
                  = (EntryHi39...13) and not (0^15 || TLB [i]216...205)) and
                  (TLB [i]140 or (TLB [i]135...128 = EntryHi7...0)) then
                        Index ← 0^26 || i5...0
                  endif
            endfor
```

**Exceptions:**

Coprocessor unusable exception

# TLBR Read Indexed TLB Entry TLBR




**Format:**

TLBR

**Description:**

The EntryHi and EntryLo registers are loaded with the contents of the TLB entry pointed at by the contents of the TLB Index register.

The *G* bit (which controls ASID matching) read from the TLB is written into both of the EntryLo0 and EntryLo1 registers. The operation is invalid (and the results are unspecified) if the contents of the TLB Index register are greater than the number of TLB entries in the processor.

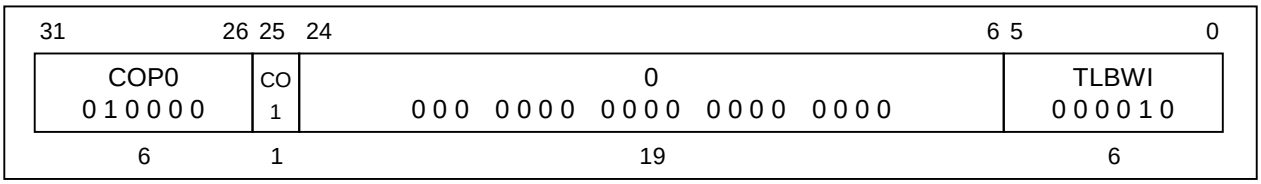
**Operation:**

```
32   T:   PageMask ← TLB [Index5...0]127...96
          EntryHi ← TLB [Index5...0]95...64 and not TLB [Index5...0]127...96
          EntryLo1 ← TLB [Index5...0]63...33 || TLB [Index5...0]76
          EntryLo0 ← TLB [Index5...0]31...1 || TLB [Index5...0]76

64   T:   PageMask ← TLB [Index5...0]255...192
          EntryHi ← TLB [Index5...0]191...128 and not TLB [Index5...0]255...192
          EntryLo1 ← TLB [Index5...0]127...65 || TLB [Index5...0]140
          EntryLo0 ← TLB [Index5...0]63...1 || TLB [Index5...0]140
```

**Exceptions:**

Coprocessor unusable exception

# TLBWI Write Indexed TLB Entry TLBWI




**Format:**

TLBWI

**Description:**

The TLB entry pointed at by the contents of the TLB Index register is loaded with the contents of the EntryHi and EntryLo registers.

The *G* bit of the TLB is written with the logical AND of the *G* bits in the EntryLo0 and EntryLo1 registers.

The operation is invalid (and the results are unspecified) if the contents of the TLB Index register are greater than the number of TLB entries in the processor.

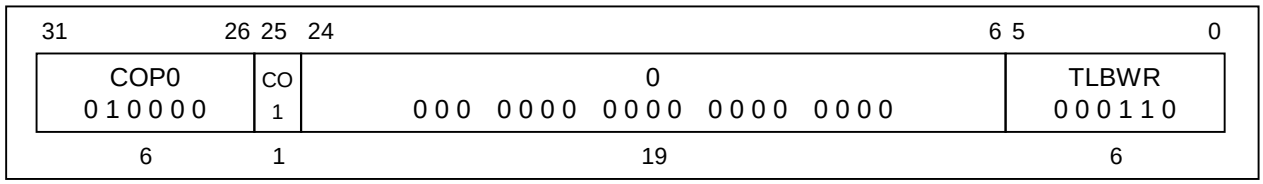
**Operation:**

```
32, 64  T:    TLB [Index5...0] ←
                    PageMask || (EntryHi and not PageMask) || EntryLo1 || EntryLo0
```

**Exceptions:**

Coprocessor unusable exception

# TLBWR Write Random TLB Entry TLBWR




**Format:**

TLBWR

**Description:**

The TLB entry pointed at by the contents of the TLB Random register is loaded with the contents of the EntryHi and EntryLo registers.

The *G* bit of the TLB is written with the logical AND of the *G* bits in the EntryLo0 and EntryLo1 registers.

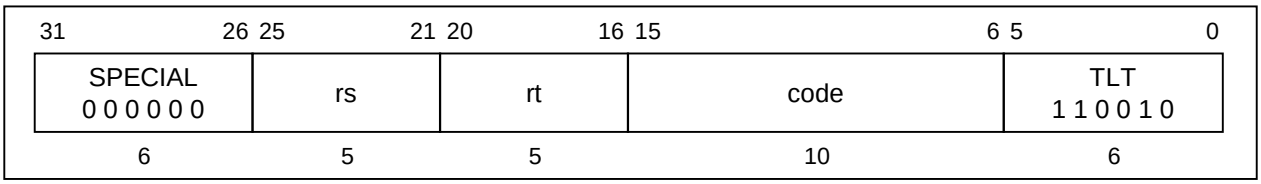
**Operation:**

```
32, 64 T:    TLB [Random5...0] ←
                    PageMask || (EntryHi and not PageMask) || EntryLo1 || EntryLo0
```

**Exceptions:**

Coprocessor unusable exception

# TLT — Trap If Less Than — TLT




**Format:**

TLT rs, rt

**Description:**

The contents of general register *rt* are compared to general register *rs*. Considering both quantities as signed integers, if the contents of general register *rs* are less than the contents of general register *rt*, a trap exception occurs.

The code field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

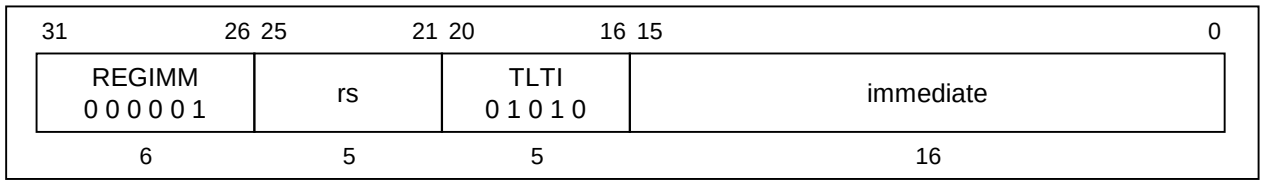
**Operation:**

```
32, 64  T:    if GPR [rs] < GPR [rt] then
                    TrapException
              endif
```

**Exceptions:**

Trap exception

# TLTI Trap If Less Than Immediate TLTI




**Format:**

TLTI rs, immediate

**Description:**

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. Considering both quantities as signed integers, if the contents of general register *rs* are less than the sign-extended *immediate*, a trap exception occurs.

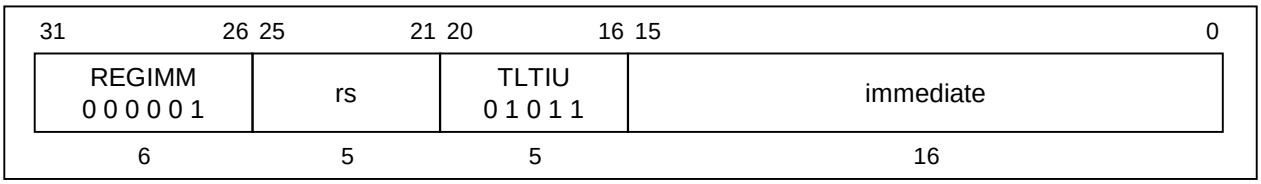
**Operation:**

```
32    T:    if GPR [rs] < (immediate15)^16 || immediate15...0 then
                  TrapException
            endif

64    T:    if GPR [rs] < (immediate15)^48 || immediate15...0 then
                  TrapException
            endif
```

**Exceptions:**

Trap exception

# TLTIU Trap If Less Than Immediate Unsigned TLTIU




**Format:**

TLTIU rs, immediate

**Description:**

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. Considering both quantities as unsigned integers, if the contents of general register *rs* are less than the sign-extended *immediate*, a trap exception occurs.

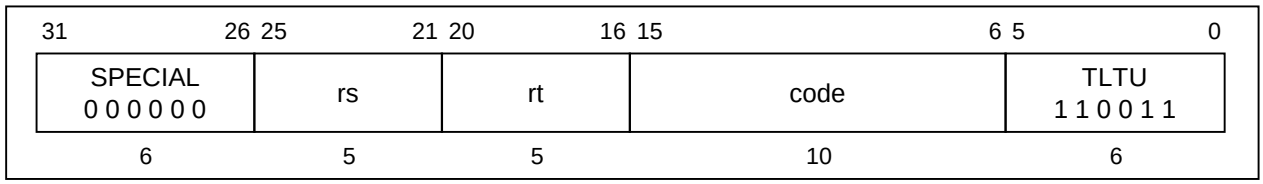
**Operation:**

```
32    T:    if (0 || GPR [rs]) < (0 || (immediate15)^16 || immediate15...0) then
                  TrapException
            endif

64    T:    if (0 || GPR [rs]) < (0 || (immediate15)^48 || immediate15...0) then
                  TrapException
            endif
```

**Exceptions:**

Trap exception

# TLTU Trap If Less Than Unsigned TLTU




**Format:**

TLTU rs, rt

**Description:**

The contents of general register *rt* are compared to general register *rs*. Considering both quantities as unsigned integers, if the contents of general register *rs* are less than the contents of general register *rt*, a trap exception occurs.

The code field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

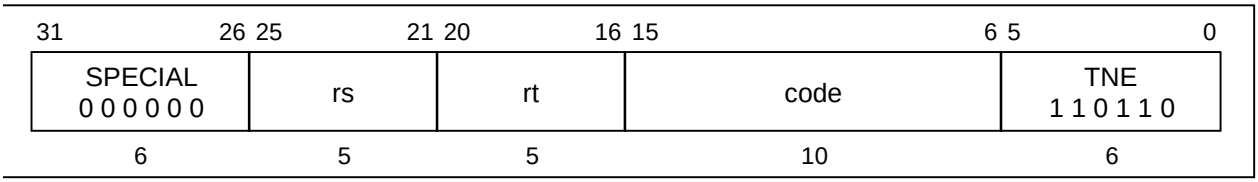
**Operation:**

```
32, 64  T:    if (0 || GPR [rs]) < (0 || GPR [rt]) then
                    TrapException
              endif
```

**Exceptions:**

Trap exception

# TNE — Trap If Not Equal — TNE




**Format:**

TNE rs, rt

**Description:**

The contents of general register *rt* are compared to general register *rs*. If the contents of general register *rs* are not equal to the contents of general register *rt*, a trap exception occurs.

The code field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

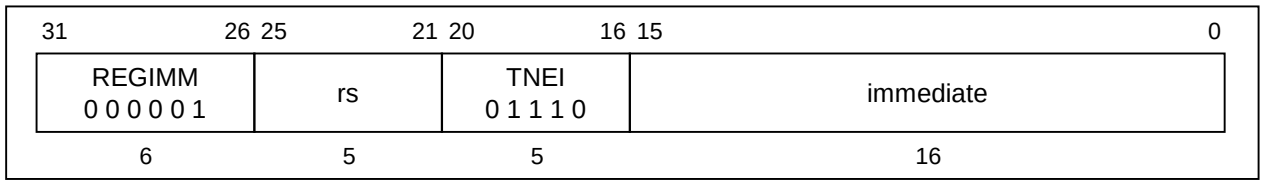
**Operation:**

```
32, 64  T:    if GPR [rs] ≠ GPR [rt] then
                    TrapException
              endif
```

**Exceptions:**

Trap exception

# TNEI Trap If Not Equal Immediate TNEI




**Format:**

TNEI rs, immediate

**Description:**

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. If the contents of general register *rs* are not equal to the sign-extended *immediate*, a trap exception occurs.

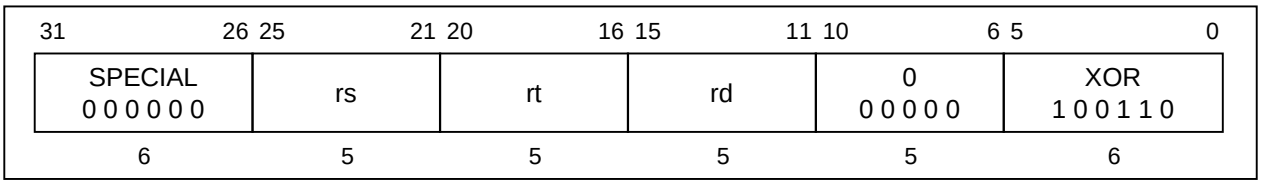
**Operation:**

```
32   T:   if GPR [rs] ≠ (immediate15)^16 || immediate15...0 then
               TrapException
          endif

64   T:   if GPR [rs] ≠ (immediate15)^48 || immediate15...0 then
               TrapException
          endif
```

**Exceptions:**

Trap exception

# XOR Exclusive Or XOR




**Format:**

XOR rd, rs, rt

**Description:**

The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical exclusive OR operation.

The result is placed into general register *rd.*

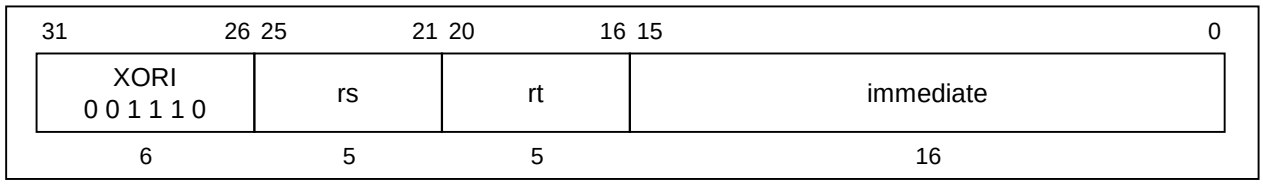
**Operation:**

```
32, 64 T:   GPR [rd] ← GPR [rs] xor GPR [rt]
```

**Exceptions:**

None

# XORI — Exclusive OR Immediate — XORI




**Format:**

XORI rt, rs, immediate

**Description:**

The 16-bit *immediate* is zero-extended and combined with the contents of general register *rs* in a bit-wise logical exclusive OR operation.

The result is placed into general register *rt.*

**Operation:**

```
32    T:    GPR [rt] ← GPR [rs] xor (0^16 || immediate)

64    T:    GPR [rt] ← GPR [rs] xor (0^48 || immediate)
```

**Exceptions:**

None

## A.6 CPU Instruction Opcode Bit Encoding

Figure A-1 lists the VR4120A Opcode Bit Encoding.

**Figure A-1. VR4120A Opcode Bit Encoding (1/2)**

**Opcode**

| 31...29 \ 28...26 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | SPECIAL | REGIMM | J | JAL | BEQ | BNE | BLEZ | BGTZ |
| 1 | ADDI | ADDIU | SLTI | SLTIU | ANDI | ORI | XORI | LUI |
| 2 | COP0 | π | π | * | BEQL | BNEL | BLEZL | BGTZL |
| 3 | DADDIε | DADDIUε | LDLε | LDRε | * | JALXθ | * | * |
| 4 | LB | LH | LWL | LW | LBU | LHU | LWR | LWUε |
| 5 | SB | SH | SWL | SW | SDLε | SDRε | SWR | CACHEδ |
| 6 | * | π | π | * | * | π | π | LDε |
| 7 | * | π | π | * | * | π | π | SDε |

**SPECIAL function**

| 5...3 \ 2...0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | SLL | * | SRL | SRA | SLLV | * | SRLV | SRAV |
| 1 | JR | JALR | * | * | SYSCALL | BREAK | * | SYNC |
| 2 | MFHI | MTHI | MFLO | MTLO | DSLLVε | * | DSRLVε | DSRAVε |
| 3 | MULT | MULTU | DIV | DIVU | DMULTε | DMULTUε | DDIVε | DDIVUε |
| 4 | ADD | ADDU | SUB | SUBU | AND | OR | XOR | NOR |
| 5 | MACC | DMACC | SLT | SLTU | DADDε | DADDUε | DSUBε | DSUBUε |
| 6 | TGE | TGEU | TLT | TLTU | TEQ | * | TNE | * |
| 7 | DSLLε | * | DSRLε | DSRAε | DSLL32ε | * | DSRL32ε | DSRA32ε |

**REGIMM rt**

| 20...19 \ 18...16 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | BLTZ | BGEZ | BLTZL | BGEZL | * | * | * | * |
| 1 | TGEI | TGEIU | TLTI | TLTIU | TEQI | * | TNEI | * |
| 2 | BLTZAL | BGEZAL | BLTZALL | BGEZALL | * | * | * | * |
| 3 | * | * | * | * | * | * | * | * |

**Figure A-1. VR4120AOpcode Bit Encoding (2/2)**

**COP0 rs**

| 25, 24 \ 23...21 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | MF | DMFε | γ | γ | MT | DMTε | γ | γ |
| 1 | BC | γ | γ | γ | γ | γ | γ | γ |
| 2 | CO | | | | | | | |
| 3 | | | | | | | | |

**COP0 rt**

| 20...19 \ 18...16 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | BCF | BCT | BCFL | BCTL | γ | γ | γ | γ |
| 1 | γ | γ | γ | γ | γ | γ | γ | γ |
| 2 | γ | γ | γ | γ | γ | γ | γ | γ |
| 3 | γ | γ | γ | γ | γ | γ | γ | γ |

**CP0 Function**

| 5...3 \ 2...0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | φ | TLBR | TLBWI | φ | φ | φ | TLBWR | φ |
| 1 | TLBP | φ | φ | φ | φ | φ | φ | φ |
| 2 | ξ | φ | φ | φ | φ | φ | φ | φ |
| 3 | ERET χ | φ | φ | φ | φ | φ | φ | φ |
| 4 | φ | STANDBY | SUSPEND | HIBERNAT | φ | φ | φ | φ |
| 5 | φ | φ | φ | φ | φ | φ | φ | φ |
| 6 | φ | φ | φ | φ | φ | φ | φ | φ |
| 7 | φ | φ | φ | φ | φ | φ | φ | φ |

**Key:**

* Operation codes marked with an asterisk cause reserved instruction exceptions in all current implementations and are reserved for future versions of the architecture.

γ Operation codes marked with a gamma cause a reserved instruction exception. They are reserved for future versions of the architecture.

δ Operation codes marked with a delta are valid only for VR4400 Series processors with CP0 enabled, and cause a reserved instruction exception on other processors.

φ Operation codes marked with a phi are invalid but do not cause reserved instruction exceptions in VR4121 implementations.

ξ Operation codes marked with a xi cause a reserved instruction exception on VR4121 processor.

χ Operation codes marked with a chi are valid on VR4000 Series only.

ε Operation codes marked with epsilon are valid when the processor operating as a 64-bit processor. These instructions will cause a reserved instruction exception if 64-bit operation is not enabled.

π Operation codes marked with a pi are invalid and cause coprocessor unusable exception.

θ Operation codes marked with a theta are valid when MIPS16 instruction execution is enabled, and cause a reserved instruction exception when MIPS16 instruction execution is disabled.

**[MEMO]**

# APPENDIX B MIPS16 INSTRUCTION SET FORMAT

This chapter describes the format of each MIPS16 instruction, and the format of the MIPS instructions that are made by converting MIPS16 instructions in alphabetical order. For details of MIPS16 instruction conversion and opcode, refer to **Section 2.3 MIPS16 Instruction Set**.

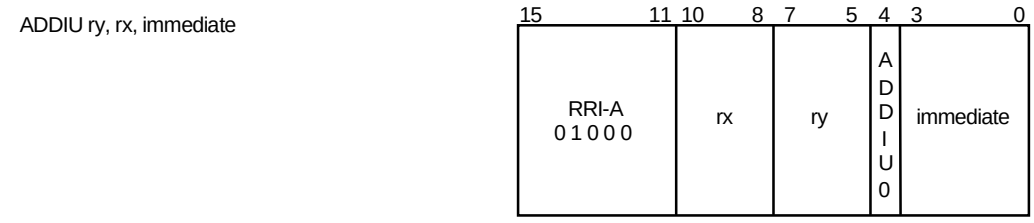
**Caution For some instructions, their format or syntax may become ineffective after they are converted to a 32-bit instruction. For details of formats and syntax of 32-bit instructions, refer to Section 2.2 MIPS III Instruction Set Summary and APPENDIX A MIPS III INSTRUCTION SET DETAILS.**

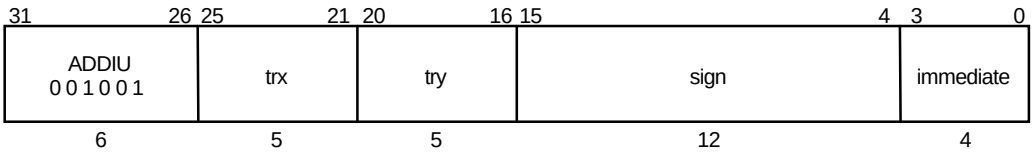
# ADDIU

**Add Immediate Unsigned**

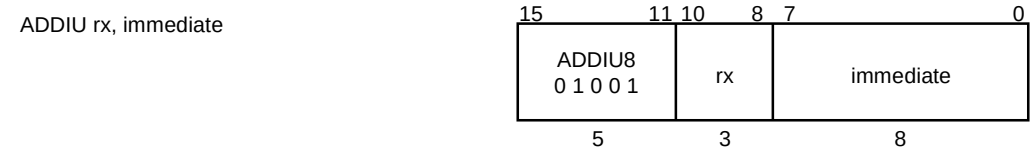
(1/2)

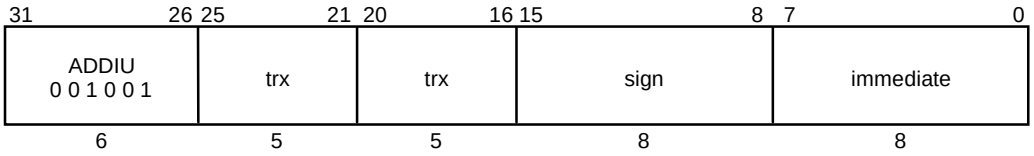
ADDIU ry, rx, immediate



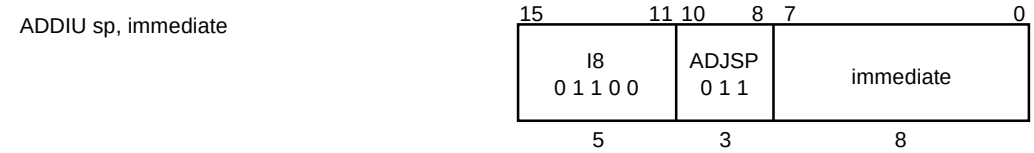



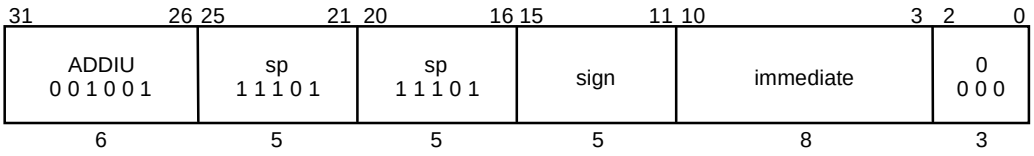

ADDIU rx, immediate




ADDIU sp, immediate



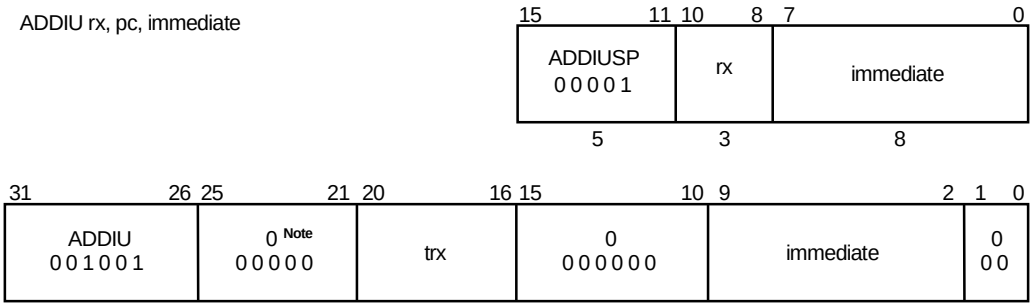



# ADDIU

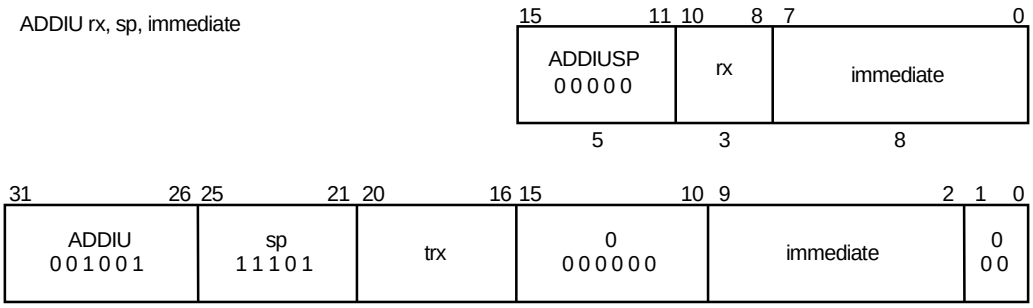
**Add Immediate Unsigned**

(2/2)

ADDIU rx, pc, immediate




ADDIU rx, sp, immediate

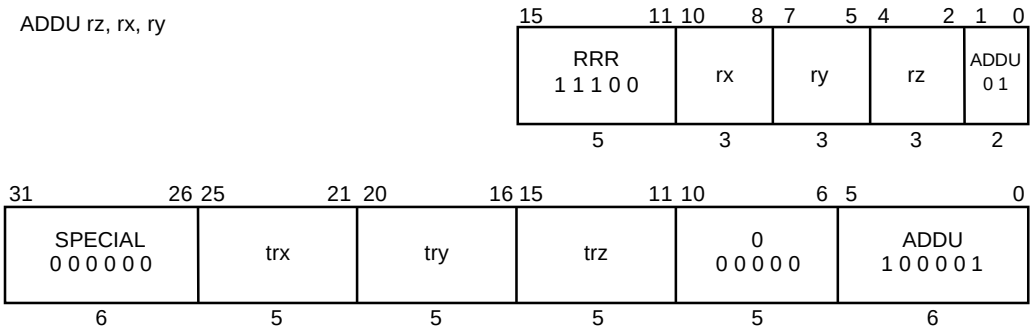



**Note** Zeros are shown in the field of bits 21 to 25 as placeholders. The 32-bit PC-relative instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. Please see **Section 2.3 MIPS16 Instruction Set** for a complete definition of the semantics of the MIPS16 PC-relative instructions.

# ADDU

**Add Unsigned**

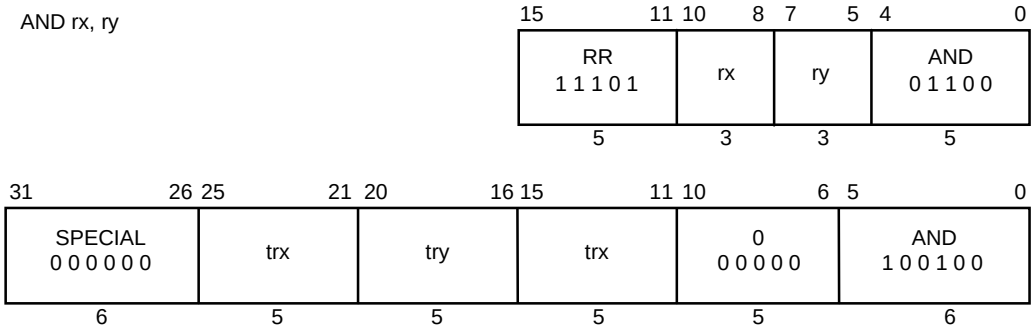
ADDU rz, rx, ry




# AND

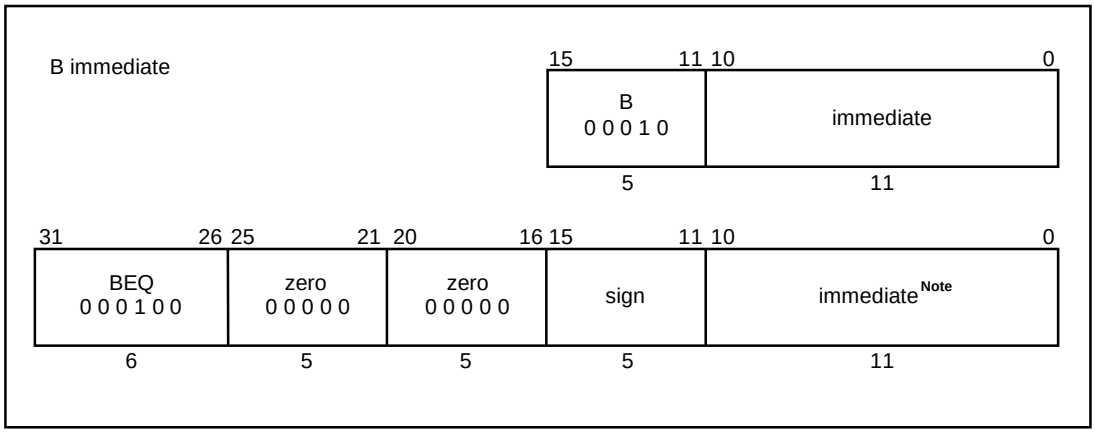
**AND**

AND rx, ry



# B

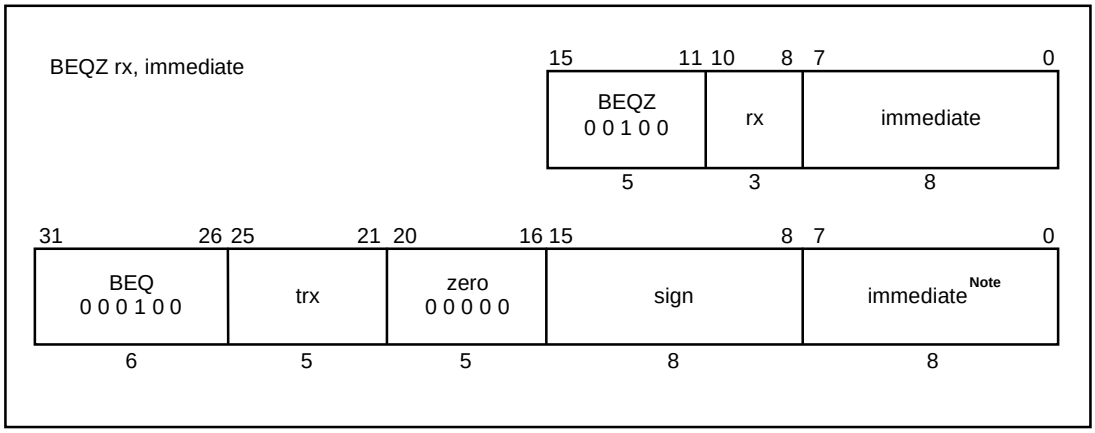
**Branch Unconditional**




**Note** In MIPS16 mode, the branch offset is interpreted as halfword aligned. This is unlike 32-bit MIPS mode which interprets the offset value as word aligned. The 32-bit branch instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. Please see **Section 2.2 MIPS III Instruction Set Summary** and **APPENDIX A MIPS III INSTRUCTION SET DETAILS** for a complete definition of the semantics of the MIPS16 branch instructions.

# BEQZ

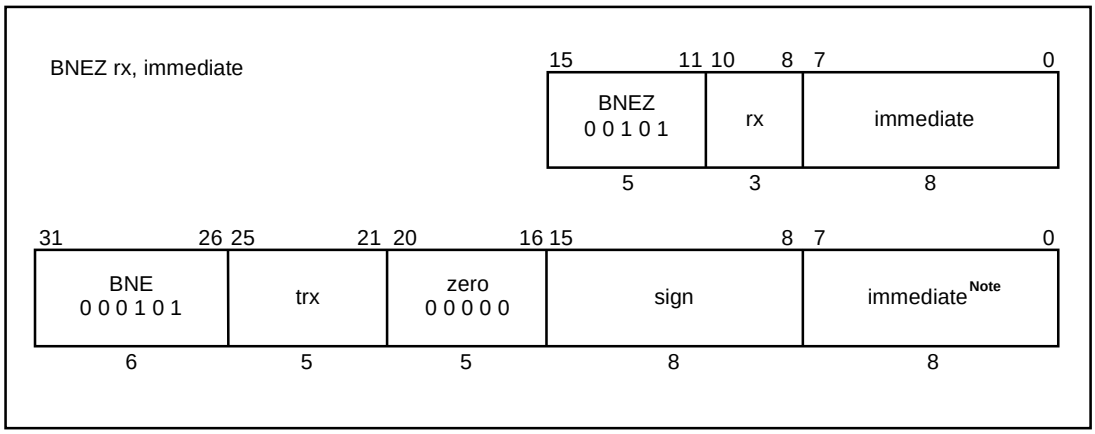
**Branch On Equal to Zero**




**Note** In MIPS16 mode, the branch offset is interpreted as halfword aligned. This is unlike 32-bit MIPS mode which interprets the offset value as word aligned. The 32-bit branch instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. Please see **Section 2.2 MIPS III Instruction Set Summary** and **APPENDIX A MIPS III INSTRUCTION SET DETAILS** for a complete definition of the semantics of the MIPS16 branch instructions.

# BNEZ

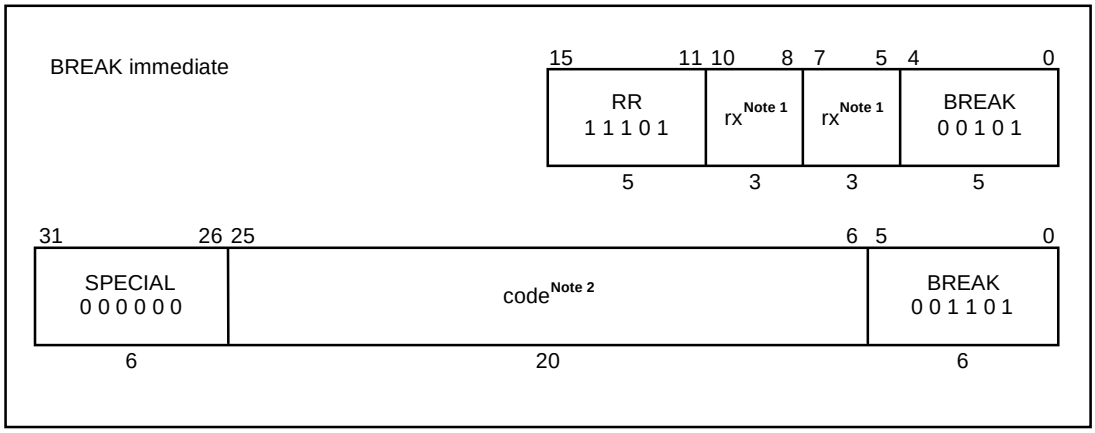
**Branch on Not Equal to Zero**




**Note** In MIPS16 mode, the branch offset is interpreted as halfword aligned. This is unlike 32-bit MIPS mode which interprets the offset value as word aligned. The 32-bit branch instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. Please see **Section 2.2 MIPS III Instruction Set Summary** and **APPENDIX A MIPS III INSTRUCTION SET DETAILS** for a complete definition of the semantics of the MIPS16 branch instructions.

# BREAK

**Breakpoint**

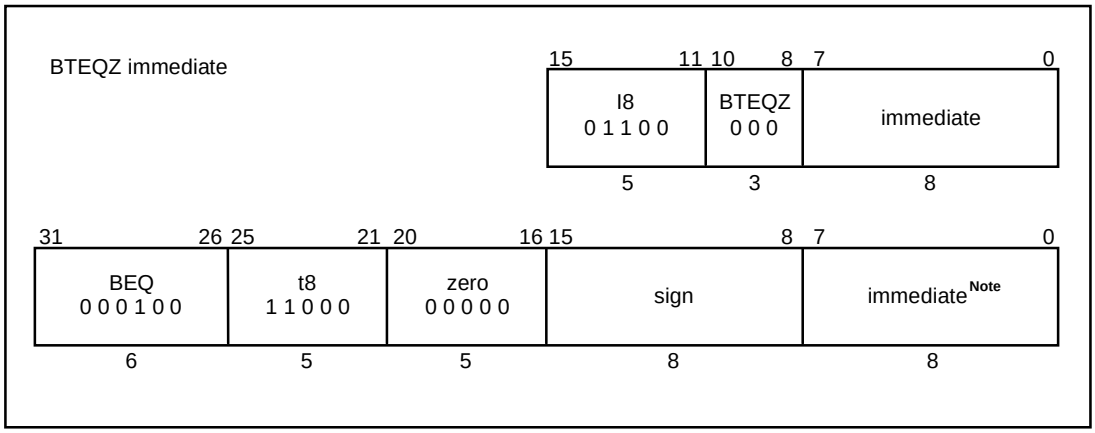



**Notes 1.** The two register fields in the MIPS16 break instruction may be used as a 6-bit code (immediate) field for software parameters. The 6-bit code can be retrieved by the exception handler.

**2.** The 32-bit break instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. The code field is entirely ignored by the pipeline, and it is not visible in any way to the software executing on the processor.

# BTEQZ

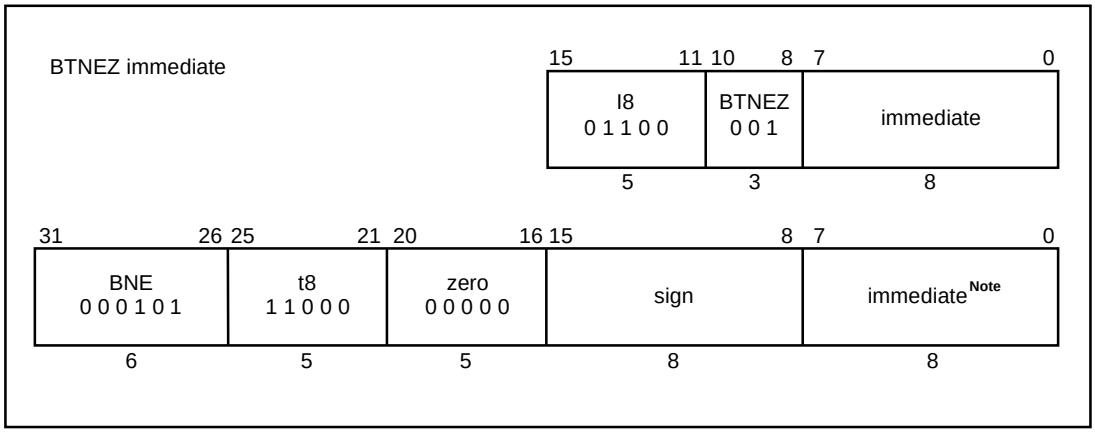
**Branch on T Equal to Zero**




**Note** In MIPS16 mode, the branch offset is interpreted as halfword aligned. This is unlike 32-bit MIPS mode which interprets the offset value as word aligned. The 32-bit branch instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. Please see **Section 2.2 MIPS III Instruction Set Summary** and **APPENDIX A MIPS III INSTRUCTION SET DETAILS** for a complete definition of the semantics of the MIPS16 branch instructions.

# BTNEZ

**Branch On T Not Equal to Zero**

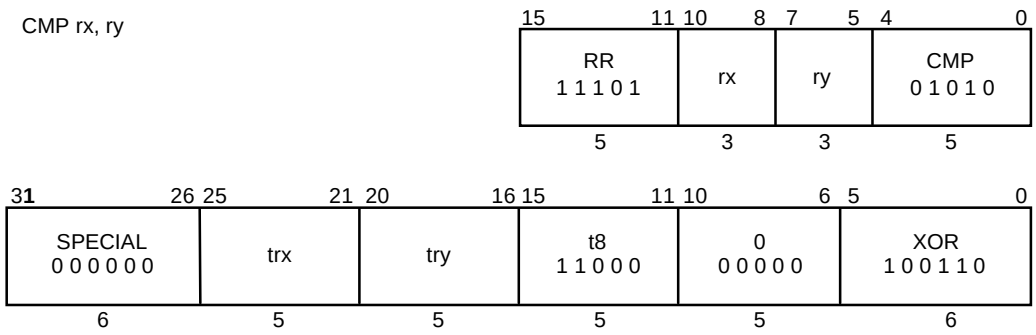



**Note** In MIPS16 mode, the branch offset is interpreted as halfword aligned. This is unlike 32-bit MIPS mode which interprets the offset value as word aligned. The 32-bit branch instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. Please see **Section 2.2 MIPS III Instruction Set Summary** and **APPENDIX A MIPS III INSTRUCTION SET DETAILS** for a complete definition of the semantics of the MIPS16 branch instructions.

# CMP

**Compare**

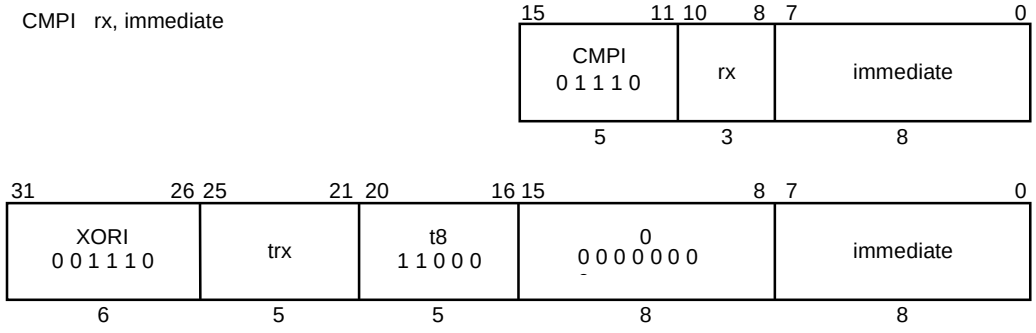
CMP rx, ry




# CMPI

**Compare Immediate**

CMPI rx, immediate

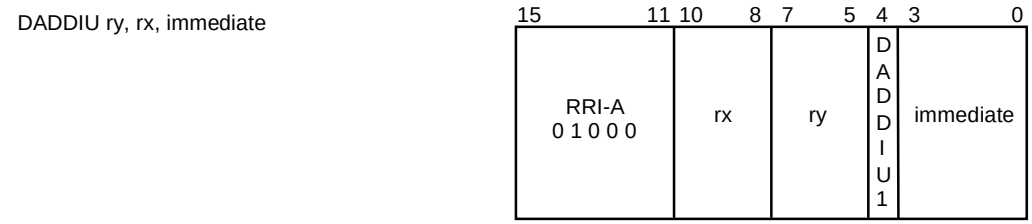


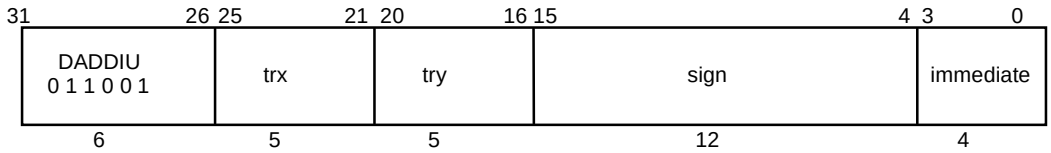
# DADDIU

**Doubleword Add Immediate Unsigned**

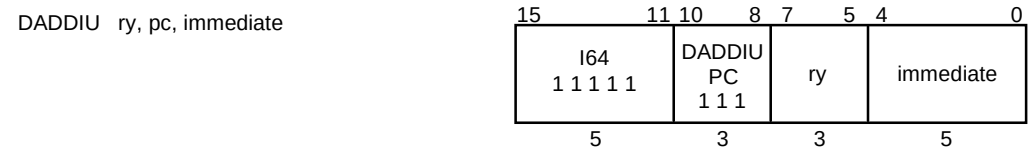
(1/2)

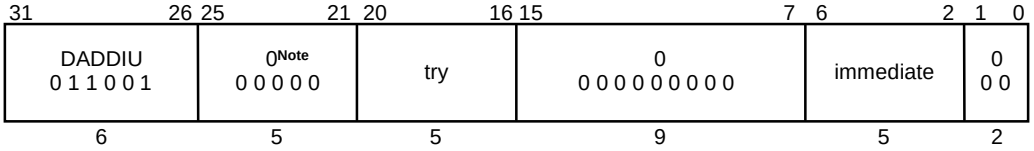
DADDIU ry, rx, immediate



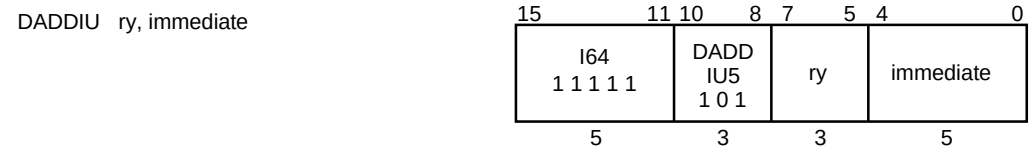



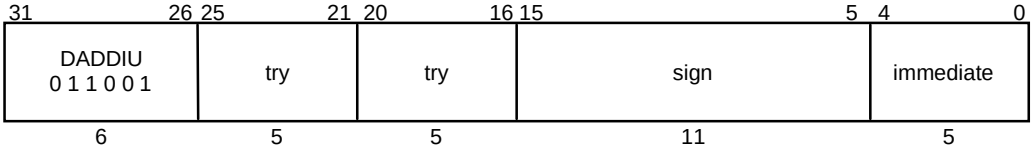

DADDIU ry, pc, immediate




DADDIU ry, immediate



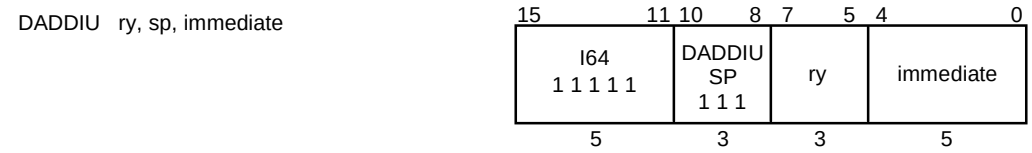

**Note** Zeros are shown in the field of bits 21 to 25 as placeholders. The 32-bit PC-relative instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. Please see **Section 2.3 MIPS16 Instruction Set** for a complete definition of the semantics of the MIPS16 PC-relative instructions.

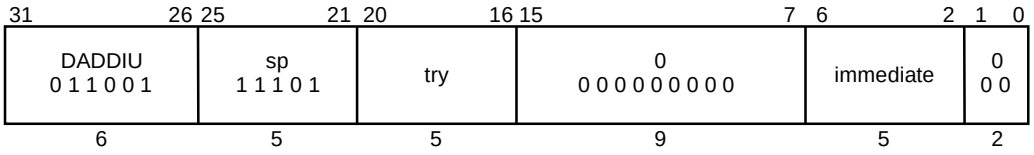
# DADDIU

**Doubleword Add Immediate Unsigned**

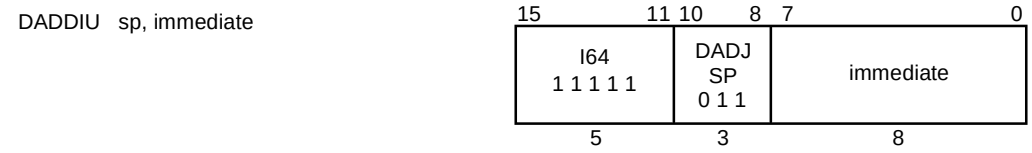
(2/2)

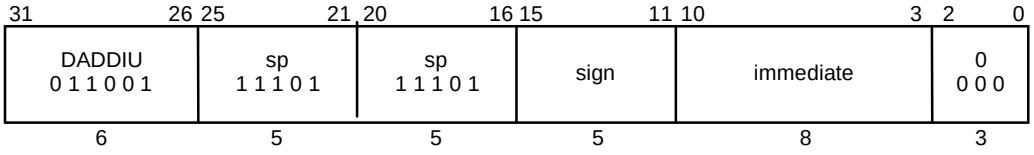
DADDIU ry, sp, immediate




DADDIU sp, immediate

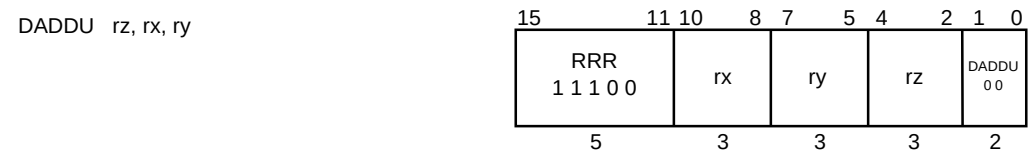


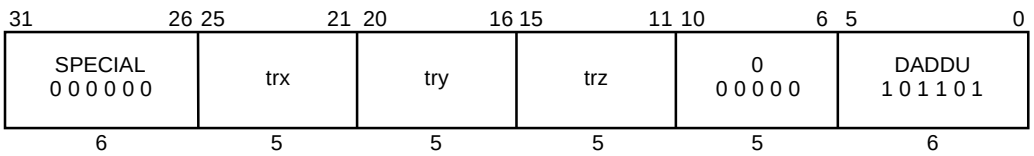




# DADDU

**Doubleword Add Unsigned**

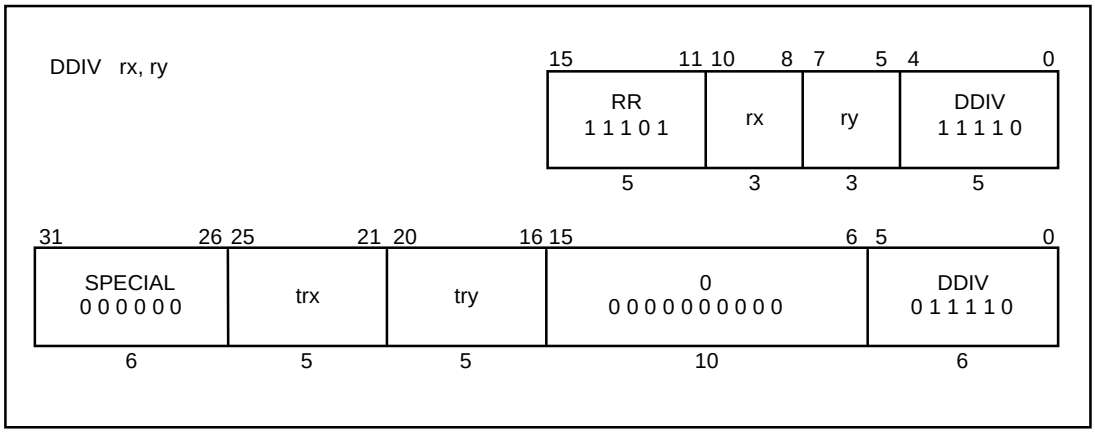
DADDU rz, rx, ry



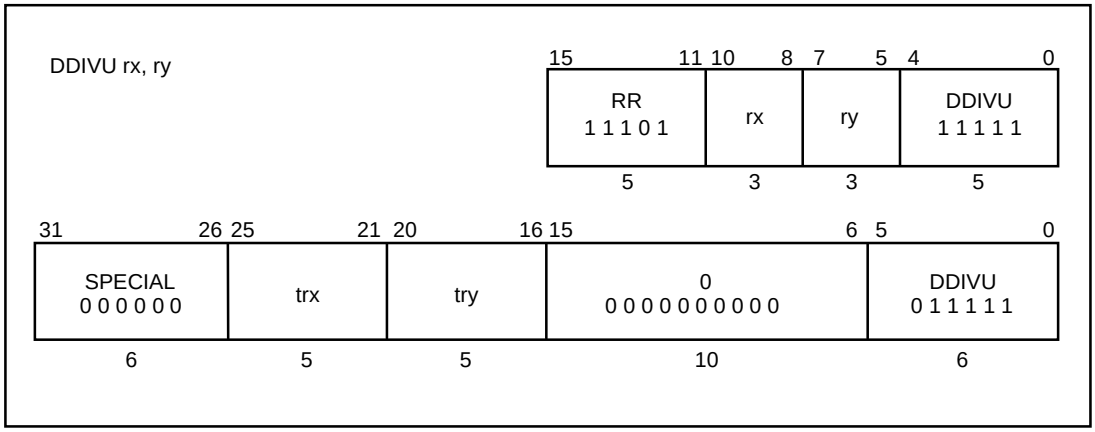
# DDIV

**Doubleword Divide**



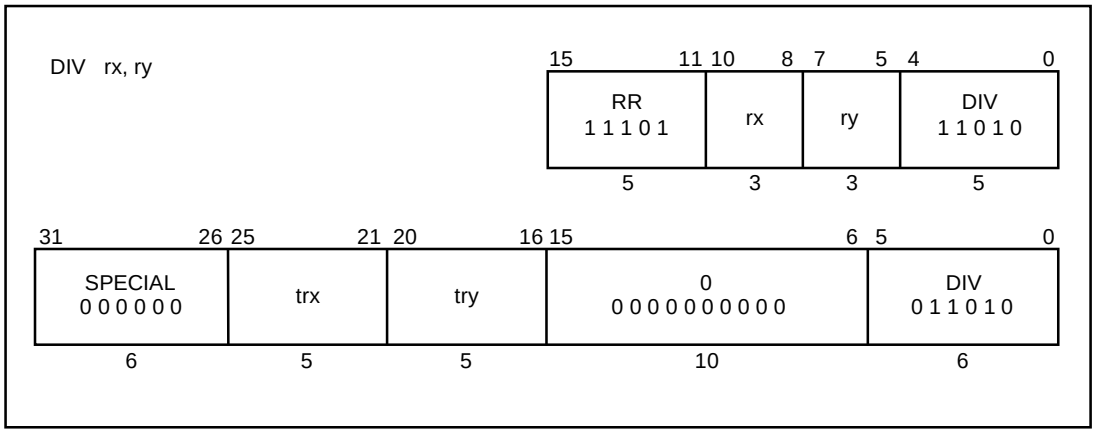

# DDIVU

**Doubleword Divide Unsigned**



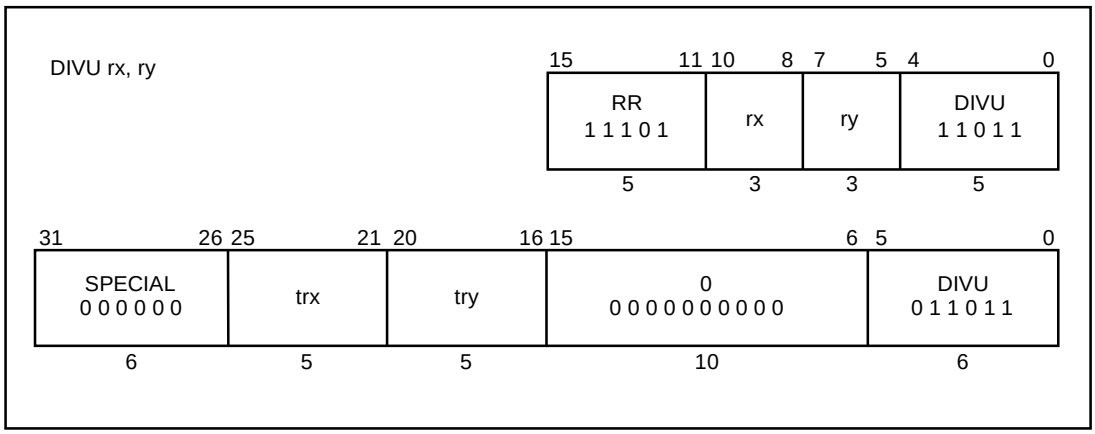
# DIV

**Divide**




# DIVU

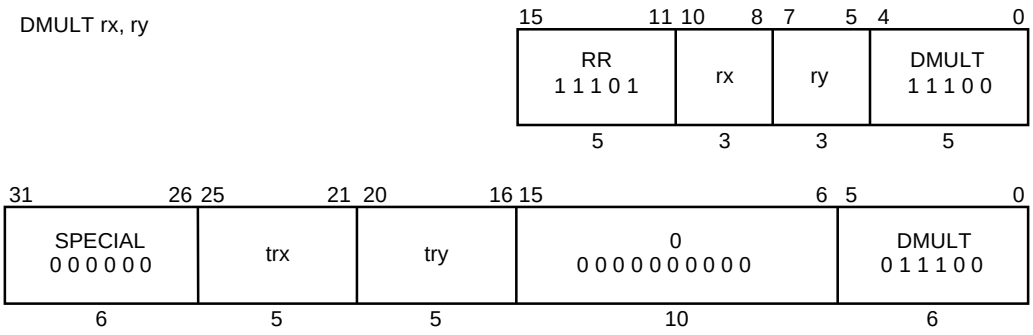
**Divide Unsigned**



# DMULT

**Doubleword Multiply**

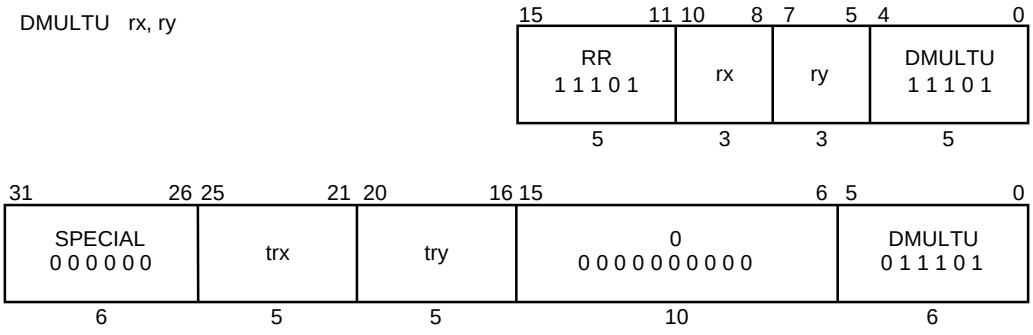
DMULT rx, ry




# DMULTU

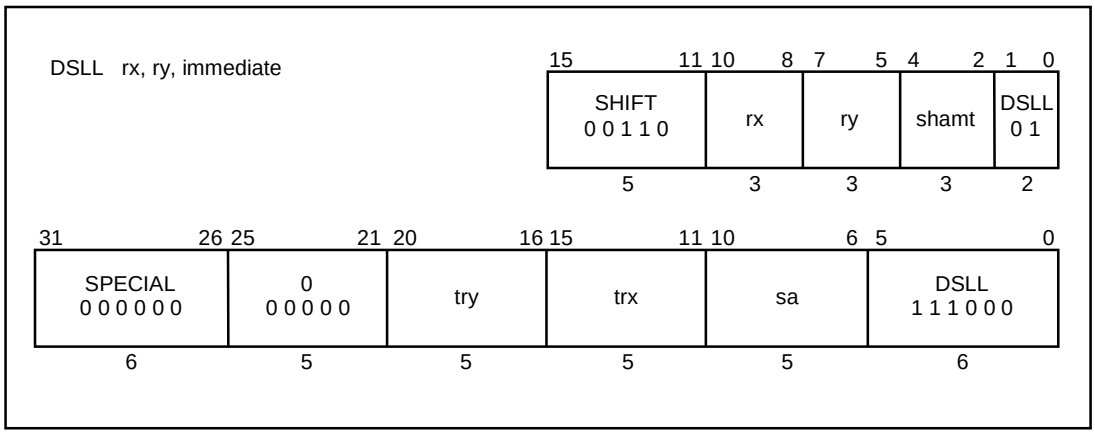
**Doubleword Multiply Unsigned**

DMULTU rx, ry



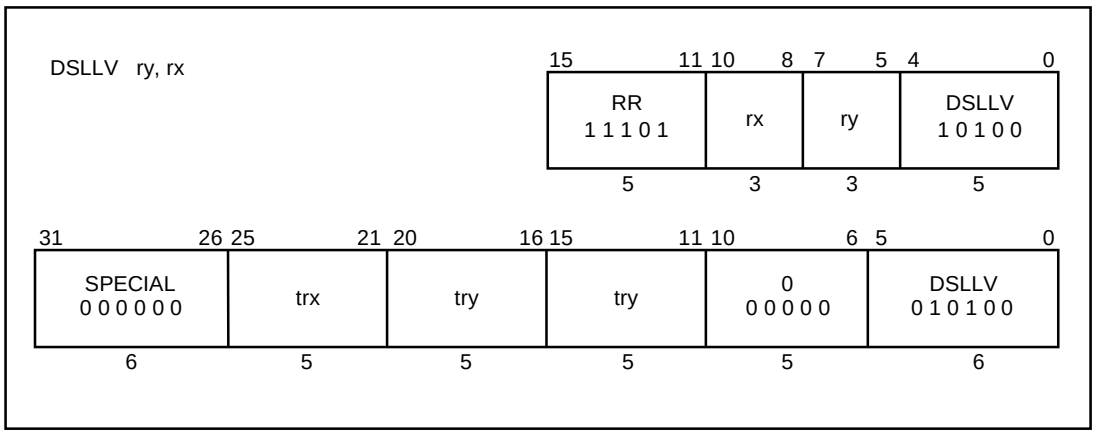
# DSLL

**Doubleword Shift Left Logical**




# DSLLV

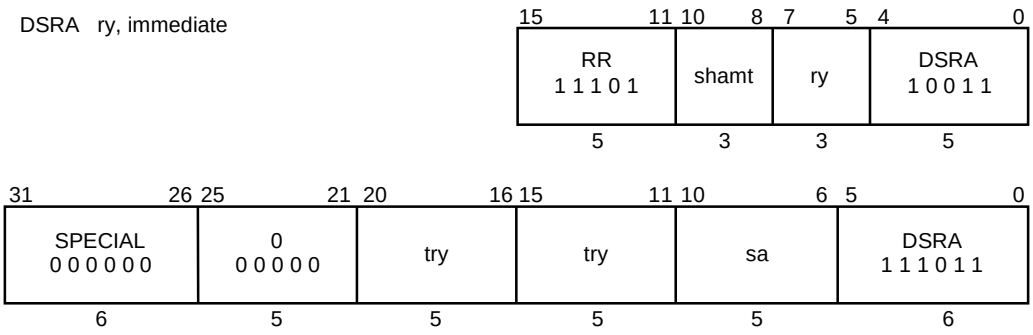
**Doubleword Shift Left Logical Variable**



# DSRA

**Doubleword Shift Right Arithmetic**

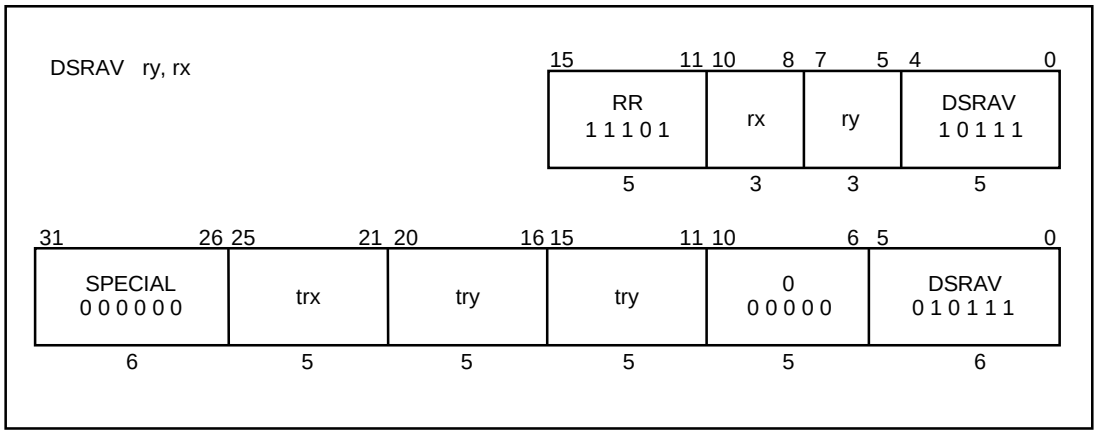
DSRA ry, immediate




# DSRAV

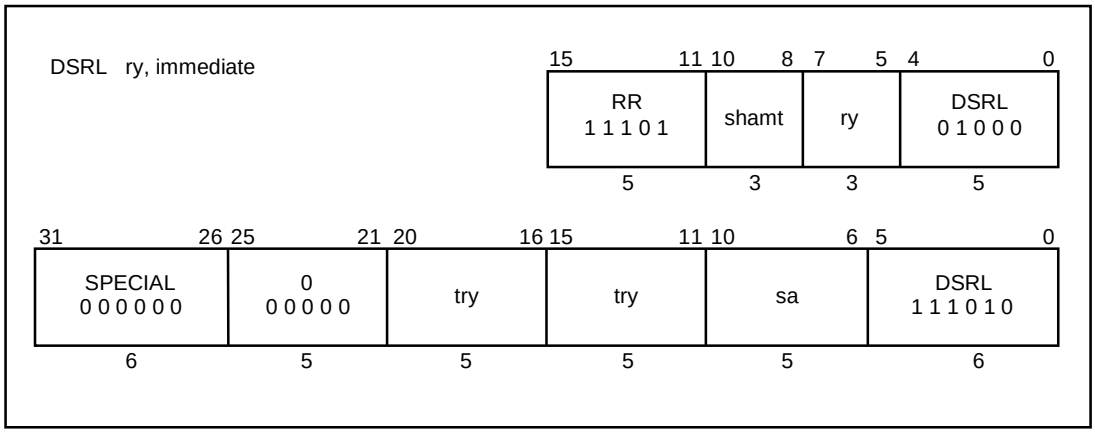
**Doubleword Shift Right Arithmetic Variable**

DSRAV ry, rx



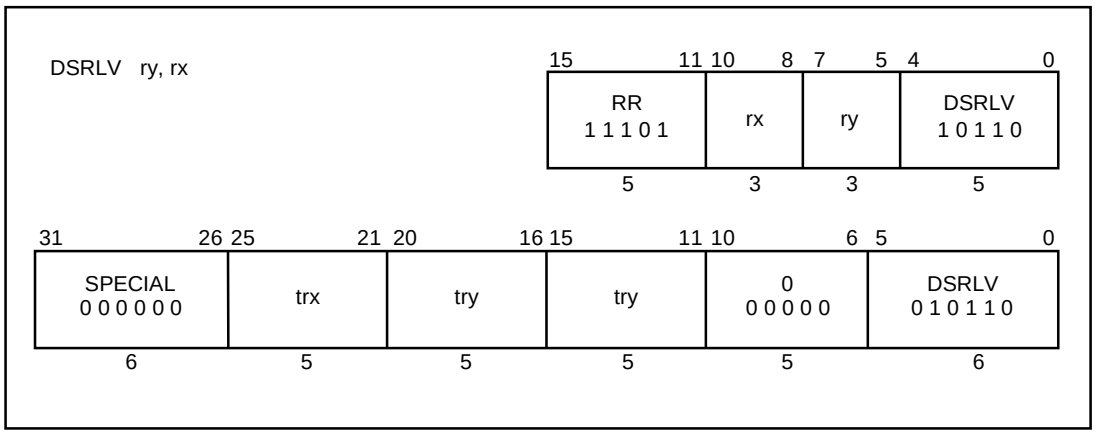
# DSRL

**Doubleword Shift Right Logical**




# DSRLV

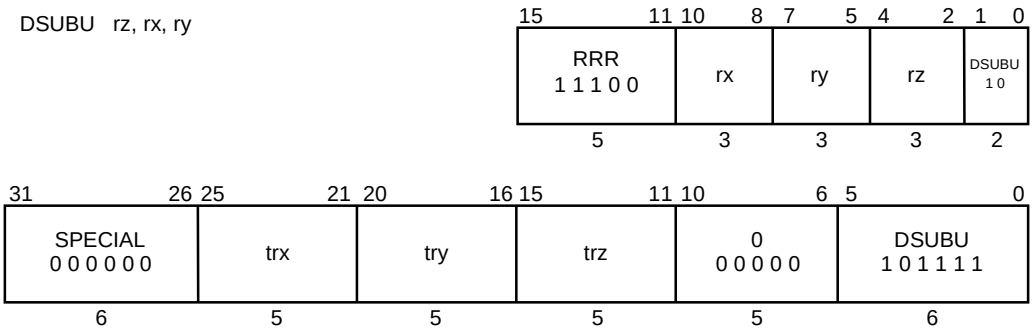
**Doubleword Shift Right Logical Variable**



# DSUBU

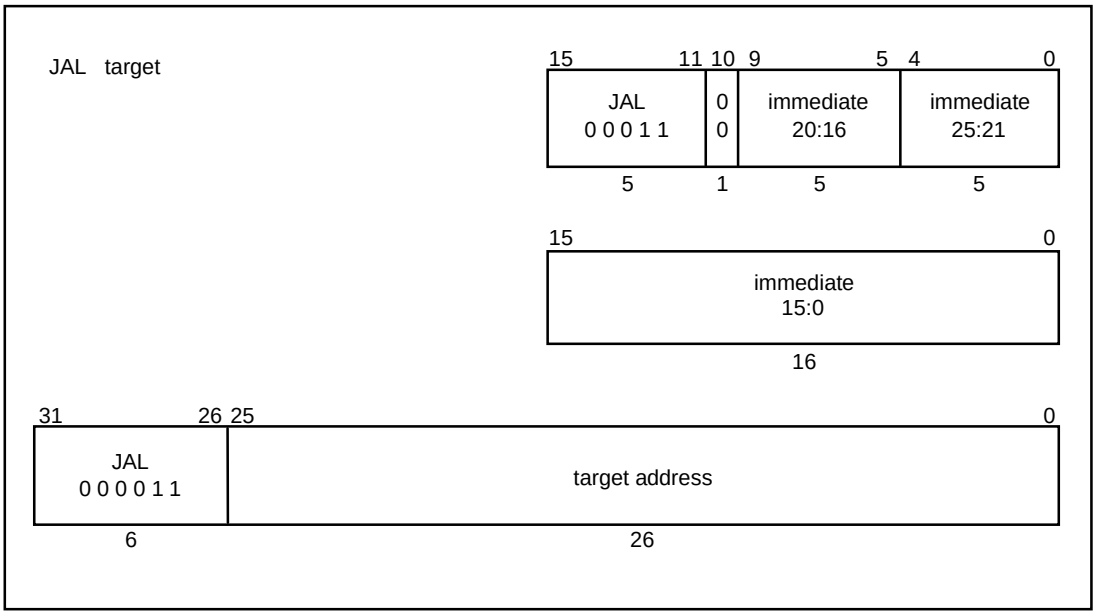
**Doubleword Subtract Unsigned**

DSUBU rz, rx, ry



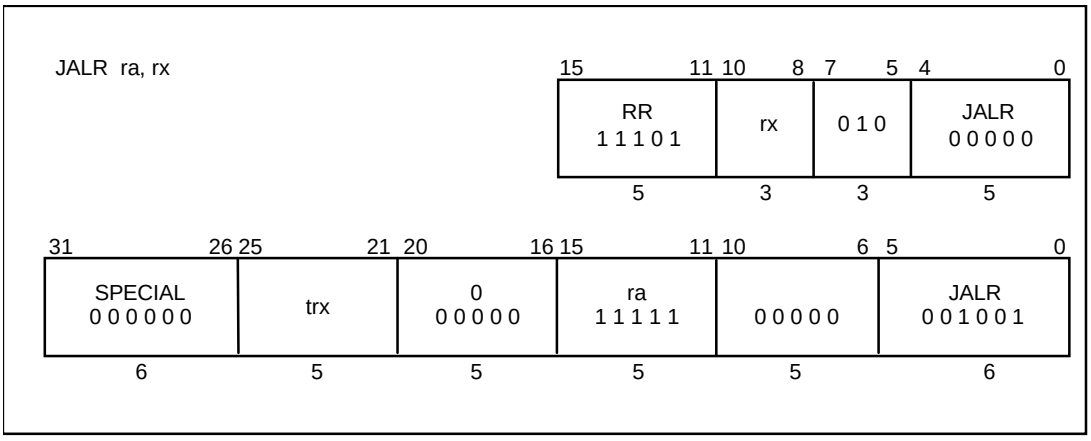
# JAL

**Jump and Link**



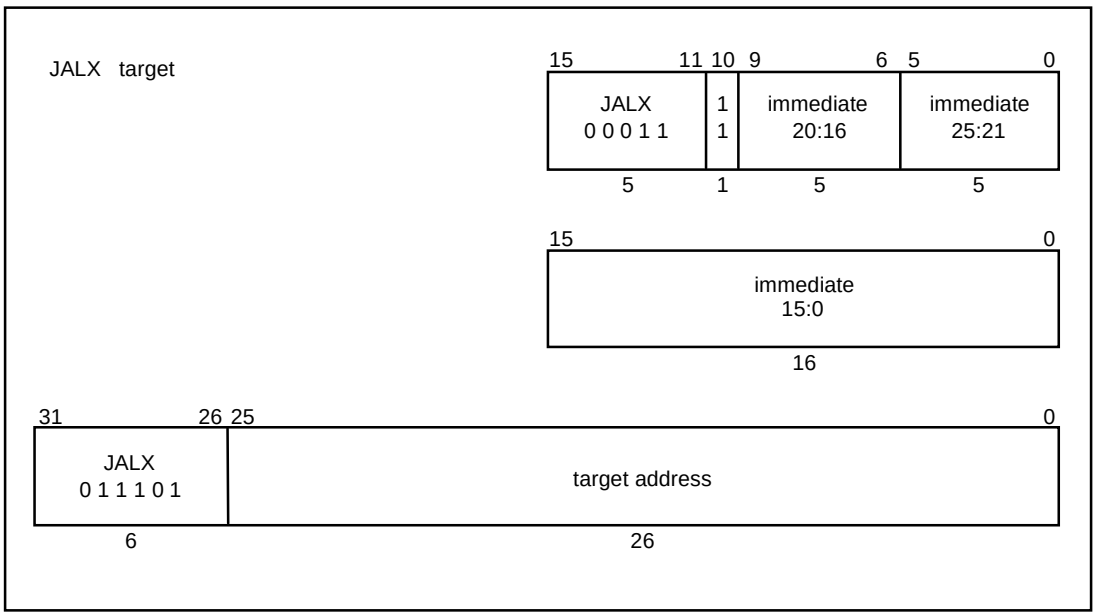

# JALR

**Jump and Link Register**



# JALX

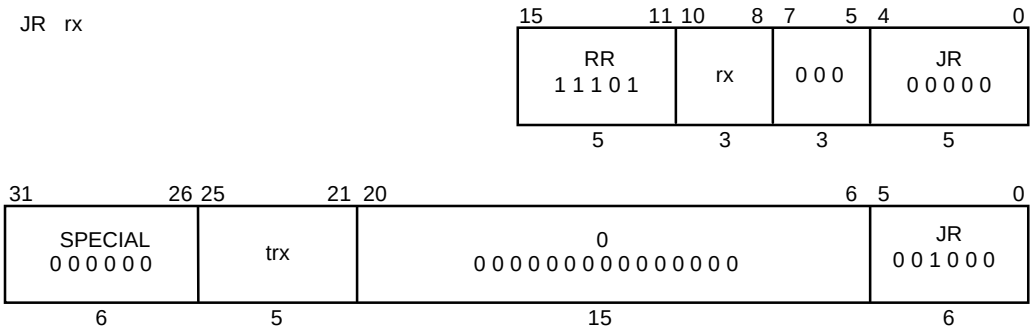
**Jump and Link Exchange**



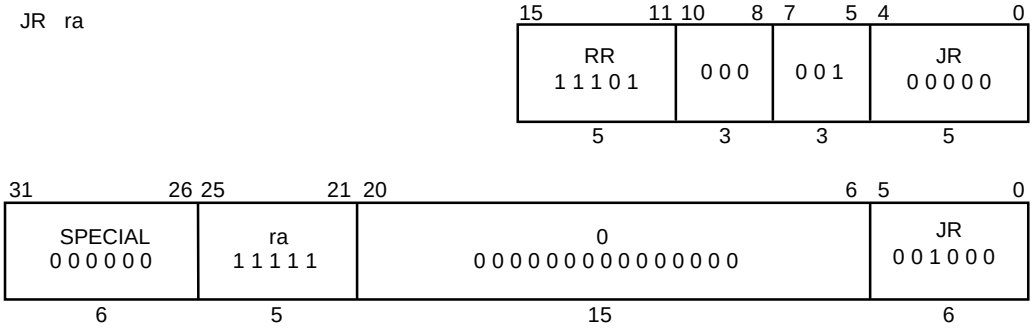
# JR

**Jump Register**

JR rx



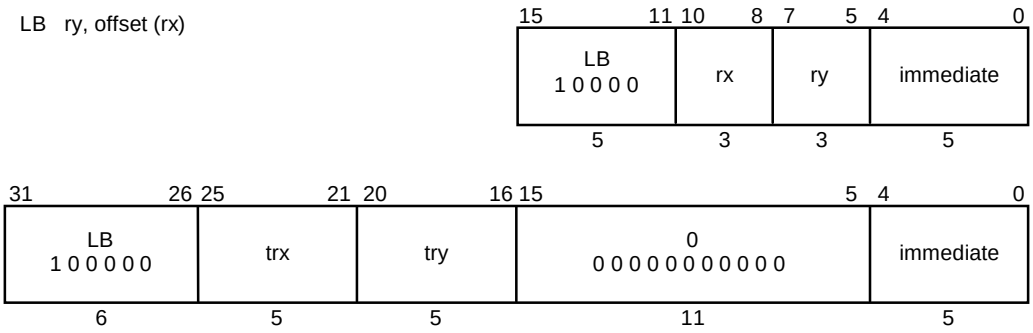

JR ra



# LB

**Load Byte**

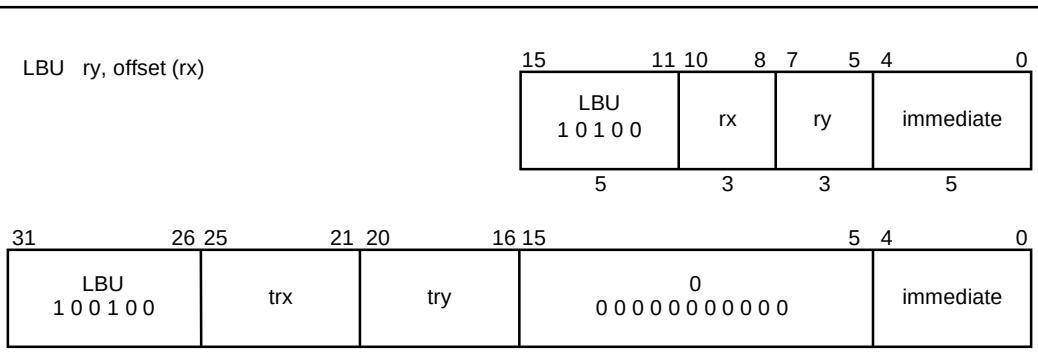
LB ry, offset (rx)




# LBU

**Load Byte Unsigned**

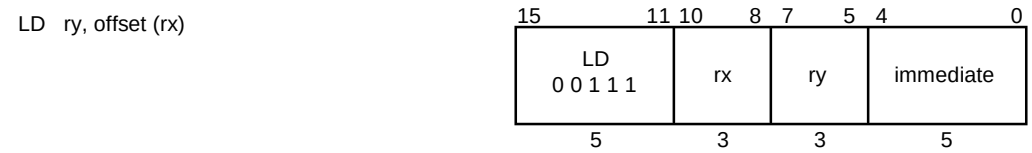
LBU ry, offset (rx)

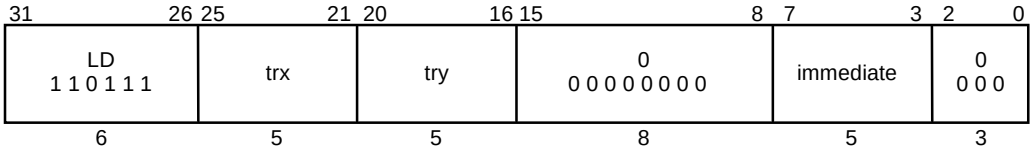


# LD

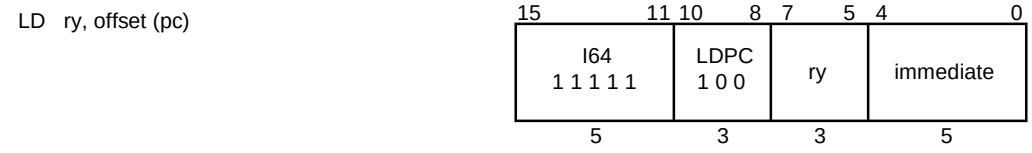
**Load Doubleword**

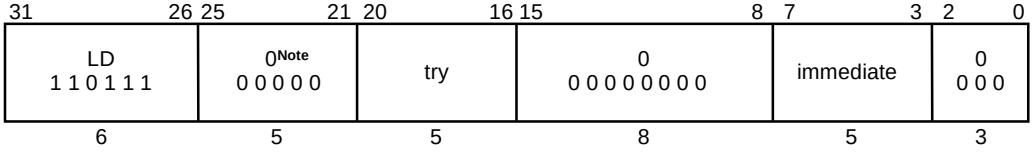
LD ry, offset (rx)



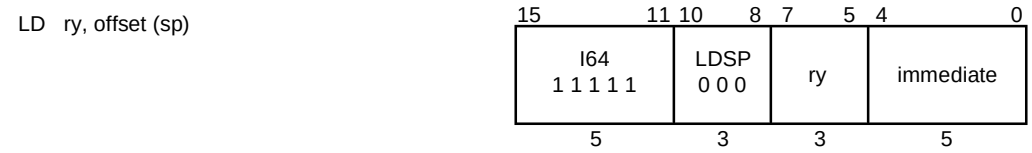

LD ry, offset (pc)

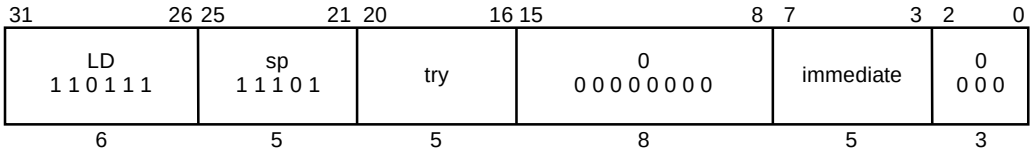



**Note** Zeros are shown in the field of bits 21 to 25 as placeholders. The 32-bit PC-relative instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. Please see **Section 2.3 MIPS16 Instruction Set** for a complete definition of the semantics of the MIPS16 PC-relative instructions.

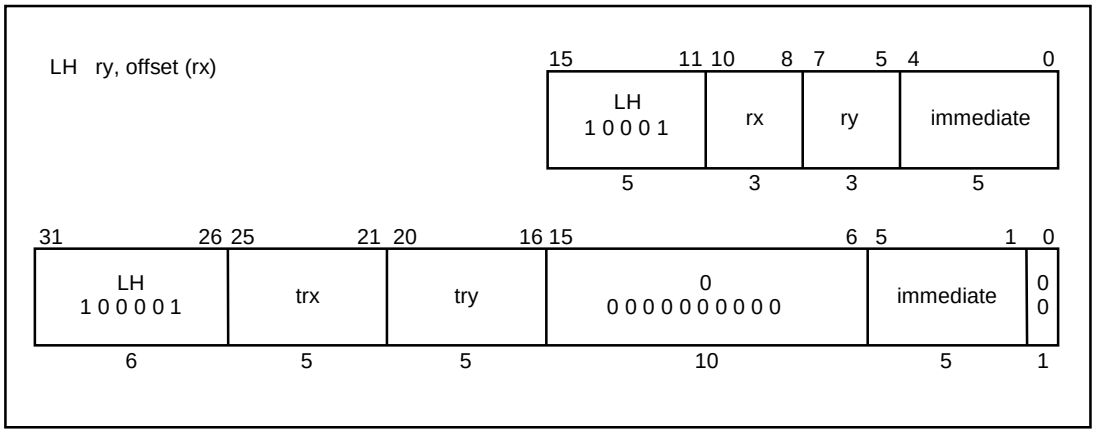
LD ry, offset (sp)



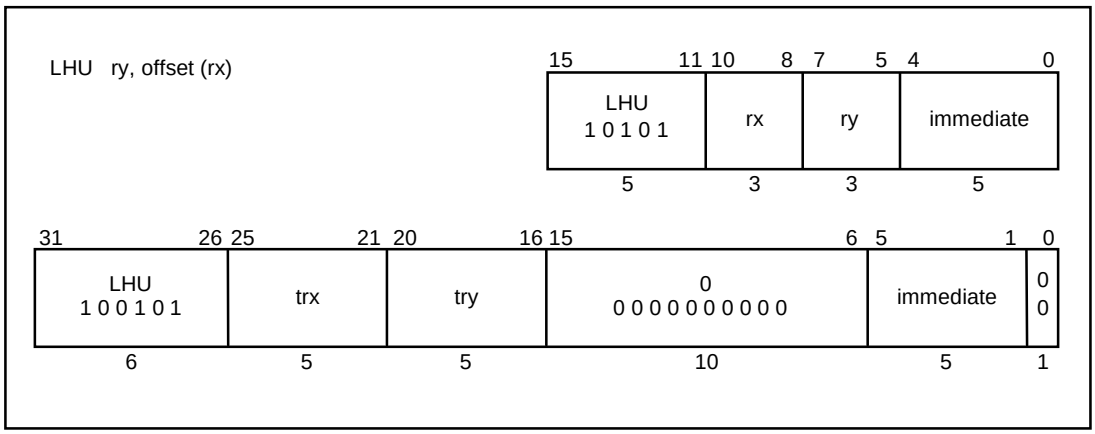
# LH

**Load Halfword**



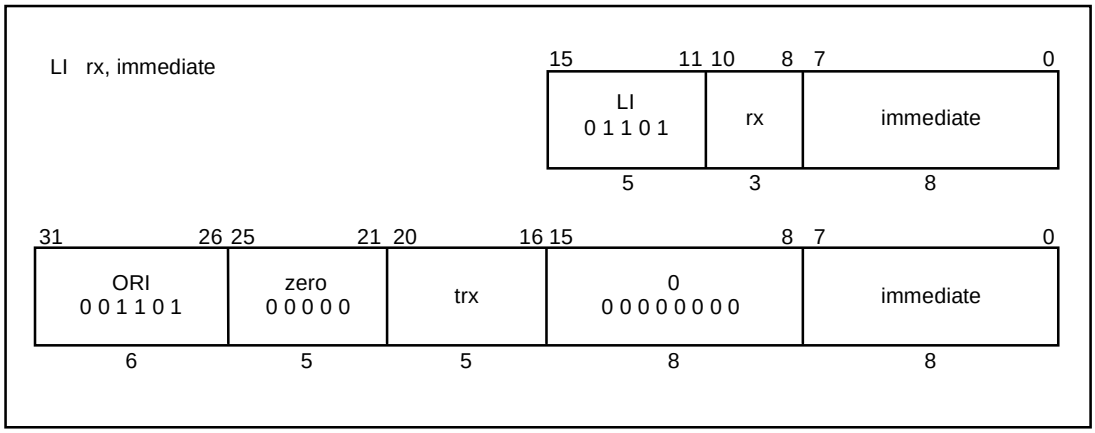

# LHU

**Load Halfword Unsigned**



# LI

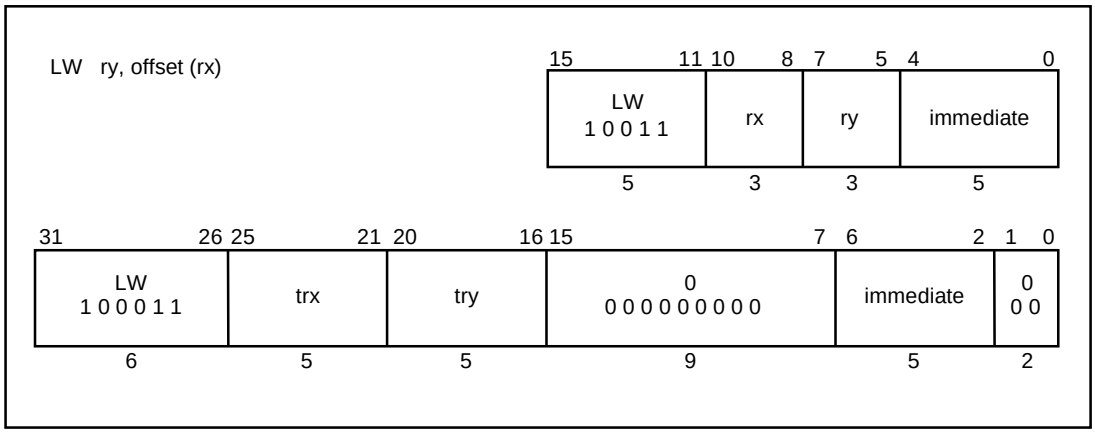
**Load Immediate**




# LW

**Load Word**

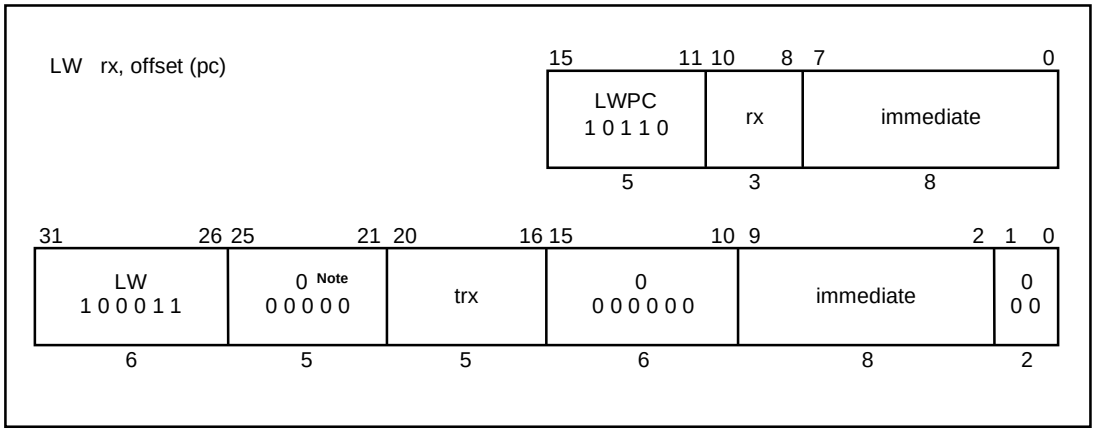
(1/2)



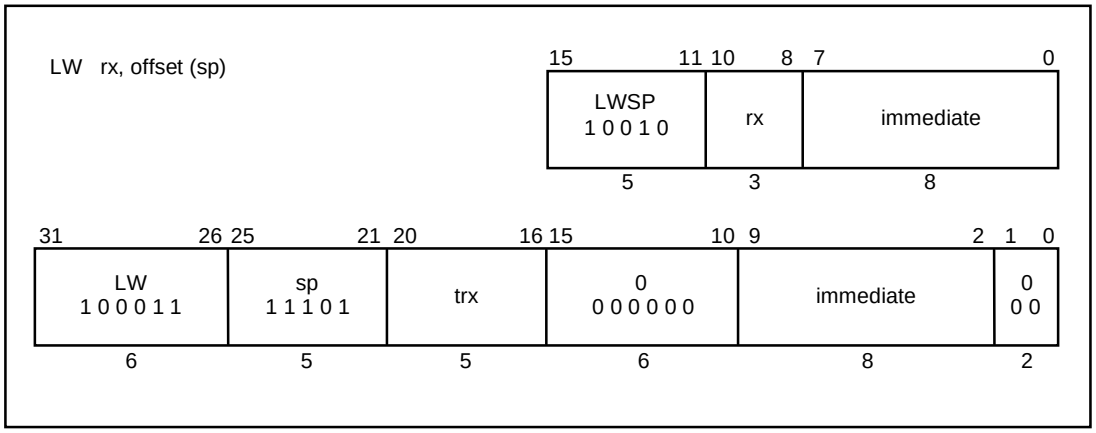
# LW

**Load Word**

(2/2)

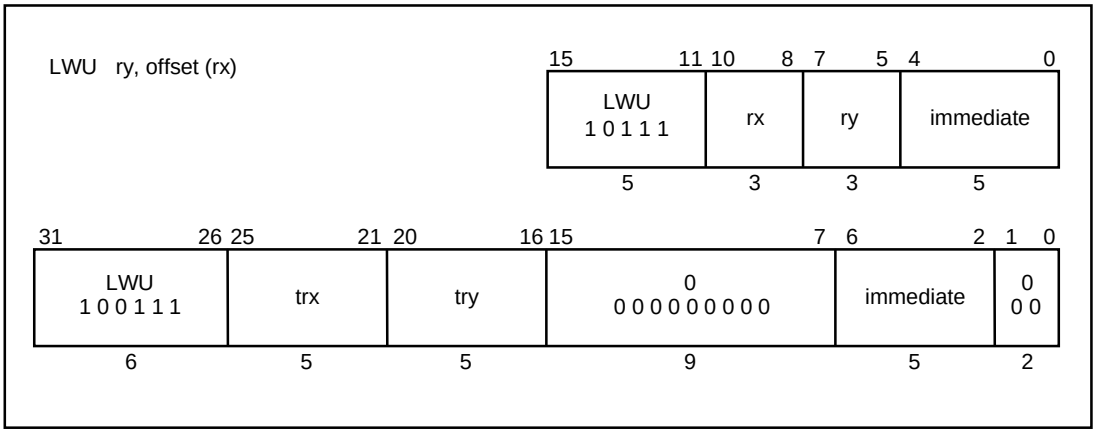



**Note** Zeros are shown in the field of bits 21 to 25 as placeholders. The 32-bit PC-relative instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. Please see **Section 2.3 MIPS16 Instruction Set** for a complete definition of the semantics of the MIPS16 PC-relative instructions.



# LWU

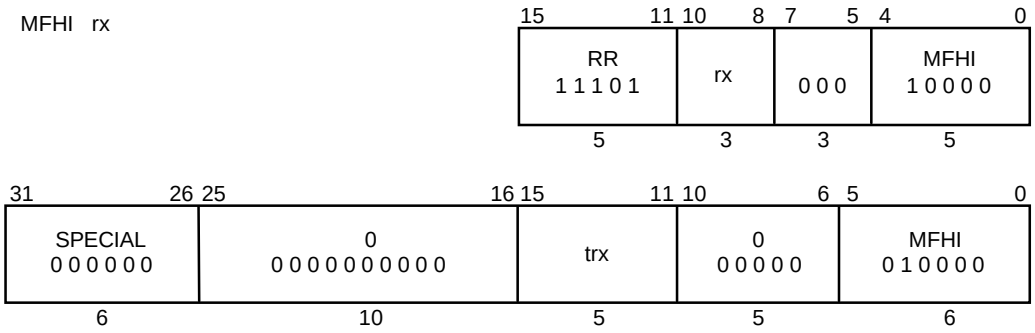
**Load Word Unsigned**



# MFHI

**Move From HI Register**

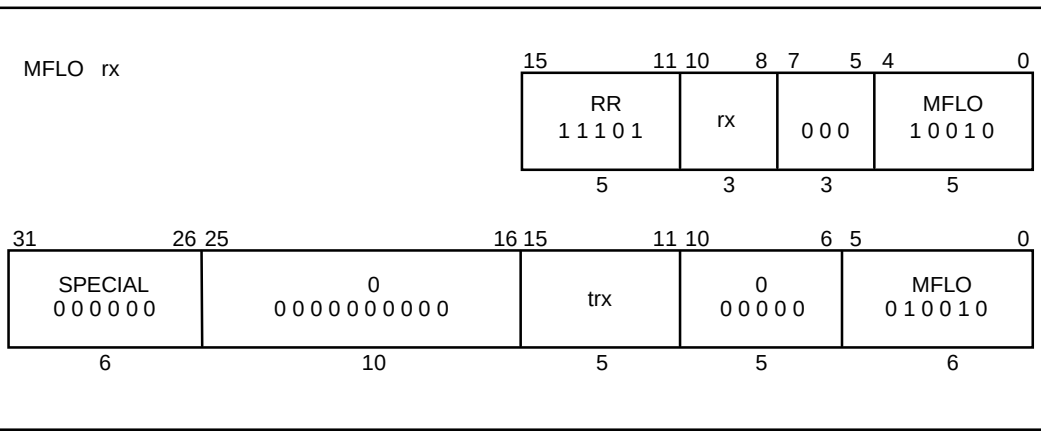
MFHI rx




# MFLO

**Move From LO Register**

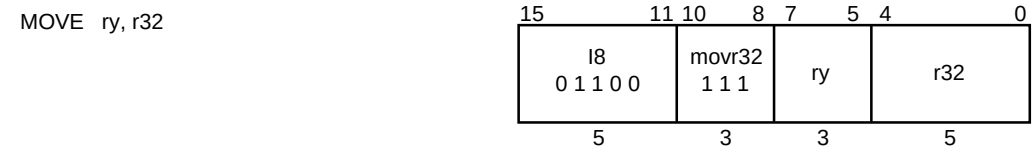
MFLO rx

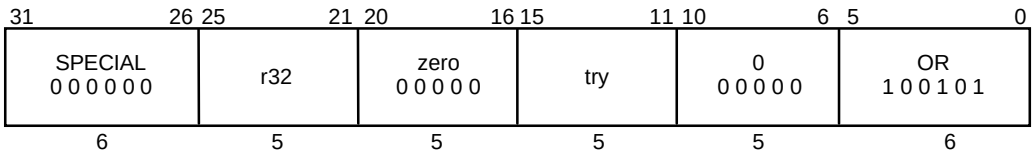


# MOVE

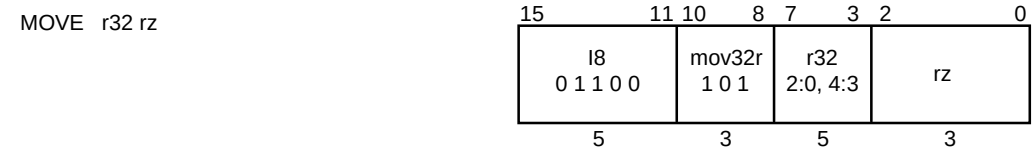
**Move**

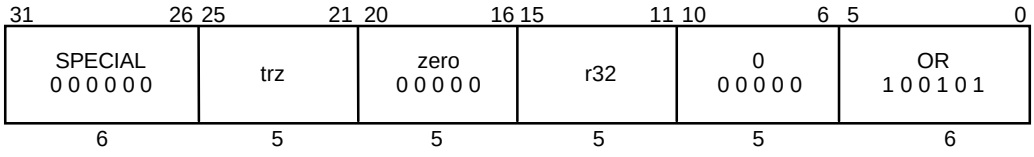
MOVE ry, r32




MOVE r32 rz

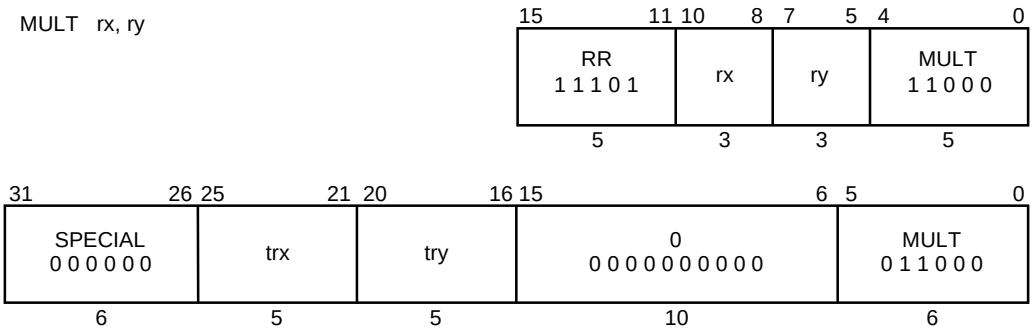


# MULT

**Multiply**

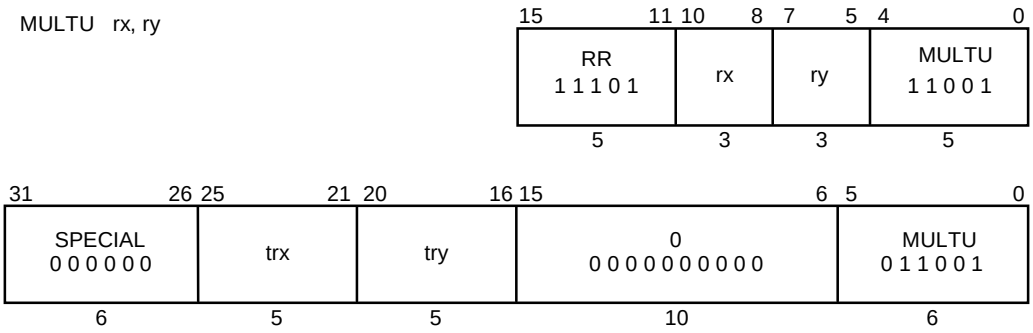
MULT rx, ry




# MULTU

**Multiply Unsigned**

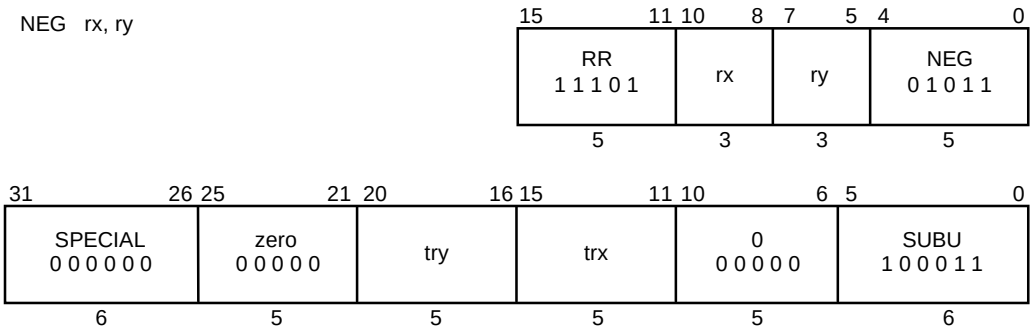
MULTU rx, ry



# NEG

**Negate**

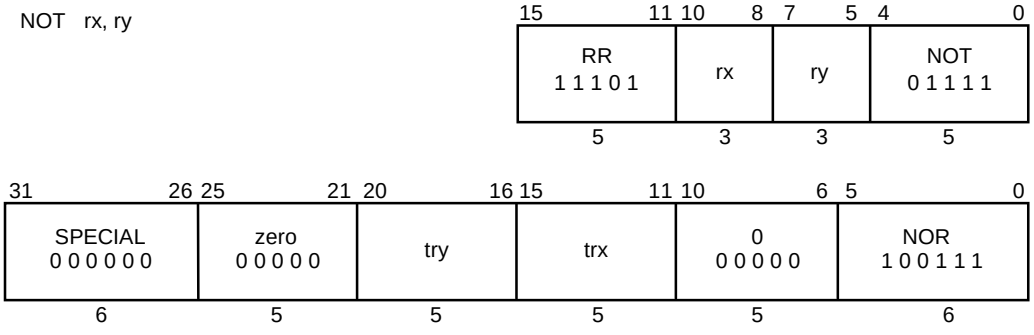
NEG rx, ry




# NOT

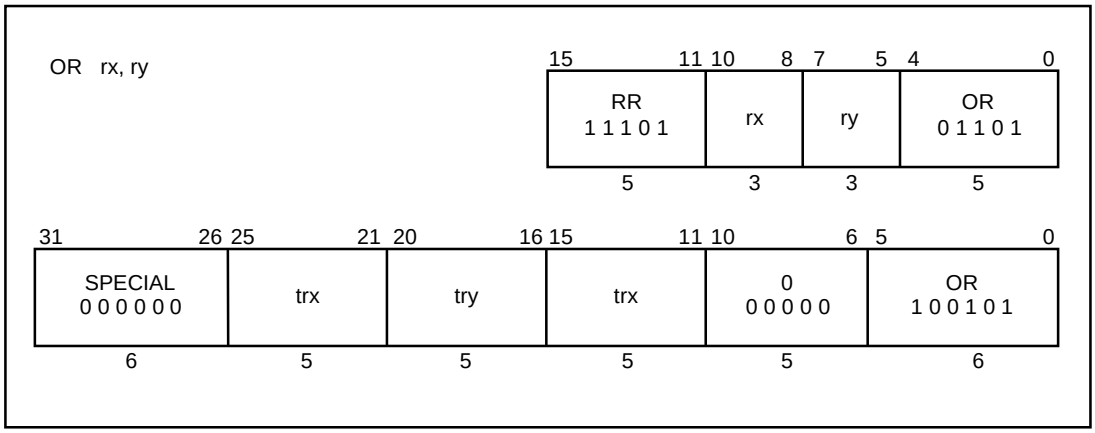
**NOT**

NOT rx, ry



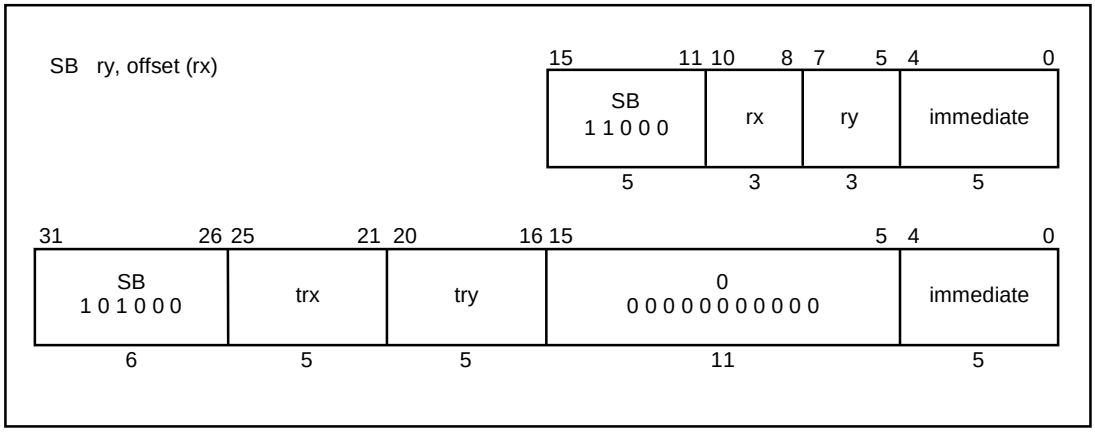
# OR

**OR**




# SB

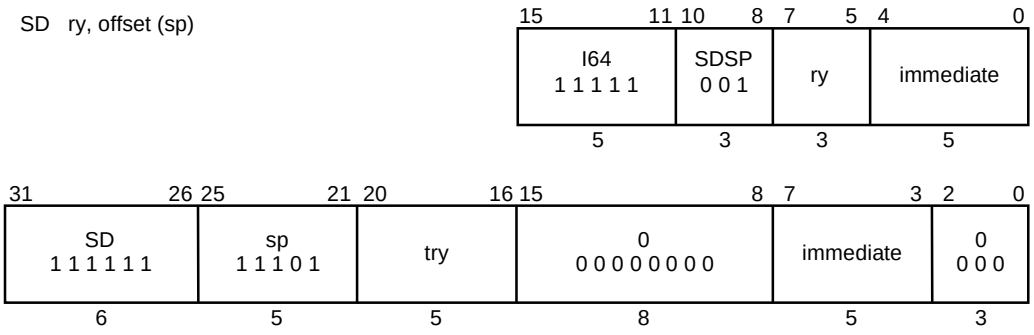
**Store Byte**



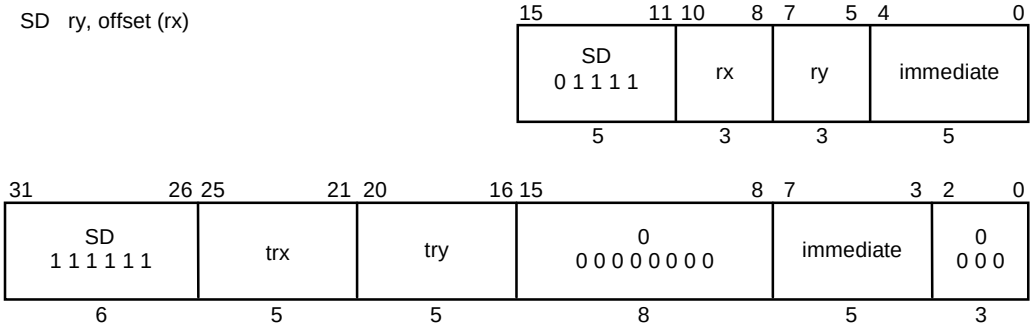
# SD

**Store Doubleword**

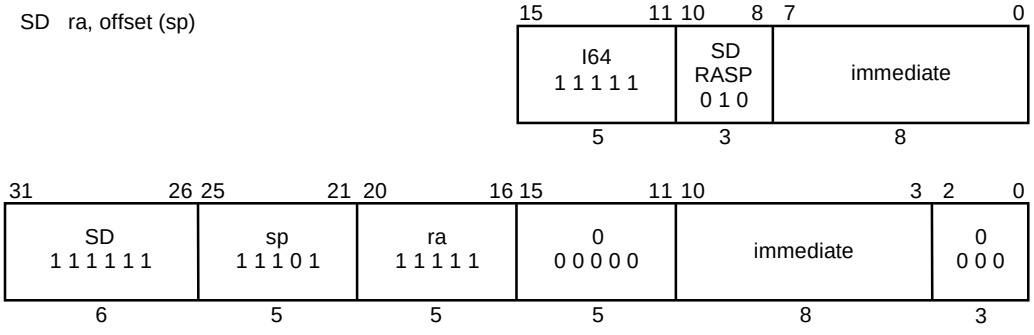
SD ry, offset (sp)




SD ry, offset (rx)

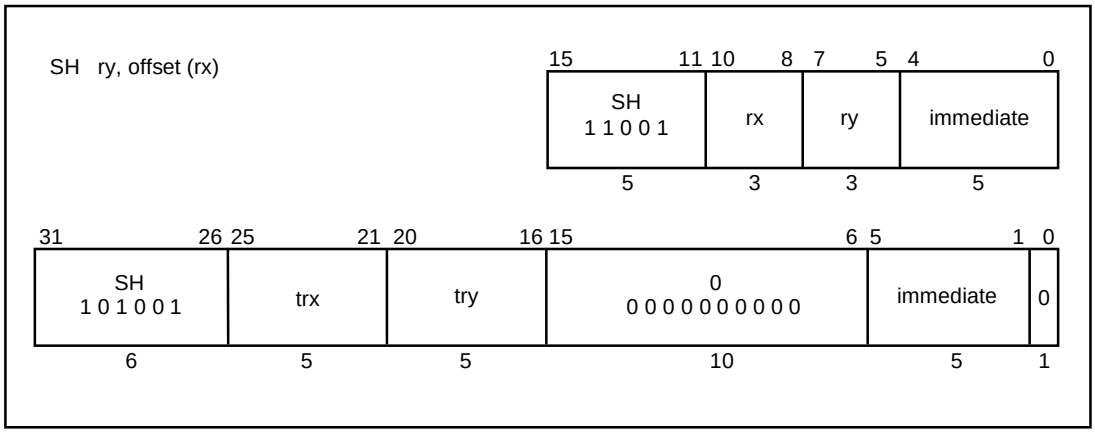



SD ra, offset (sp)



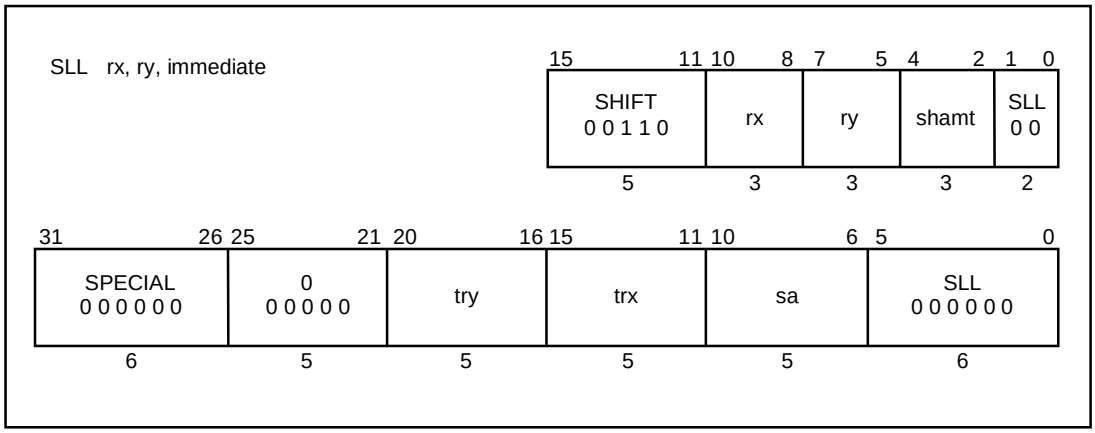
# SH

**Store Halfword**



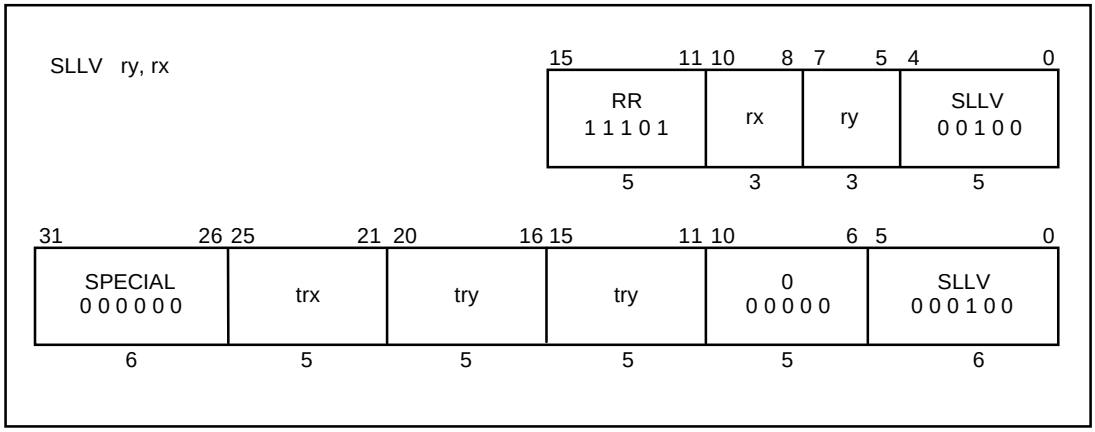
# SLL

**Shift Left Logical**



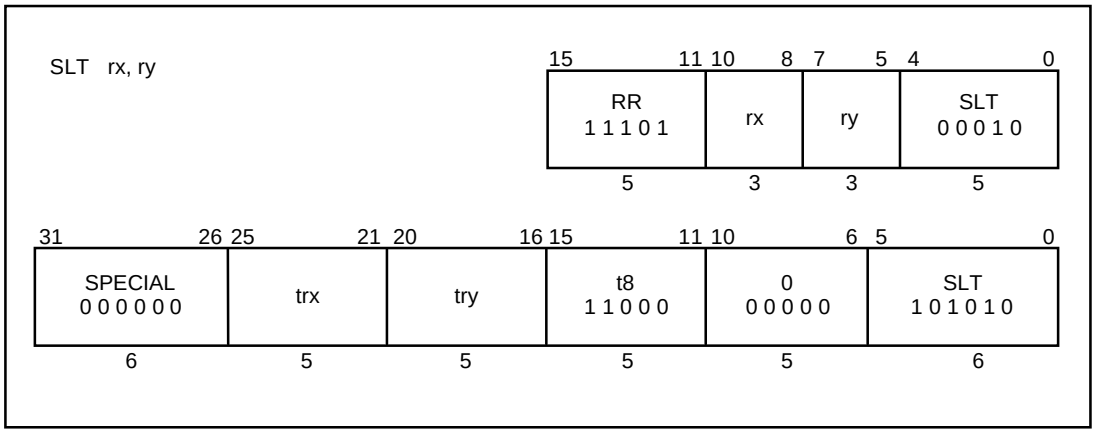

# SLLV

**Shift Left Logical Variable**



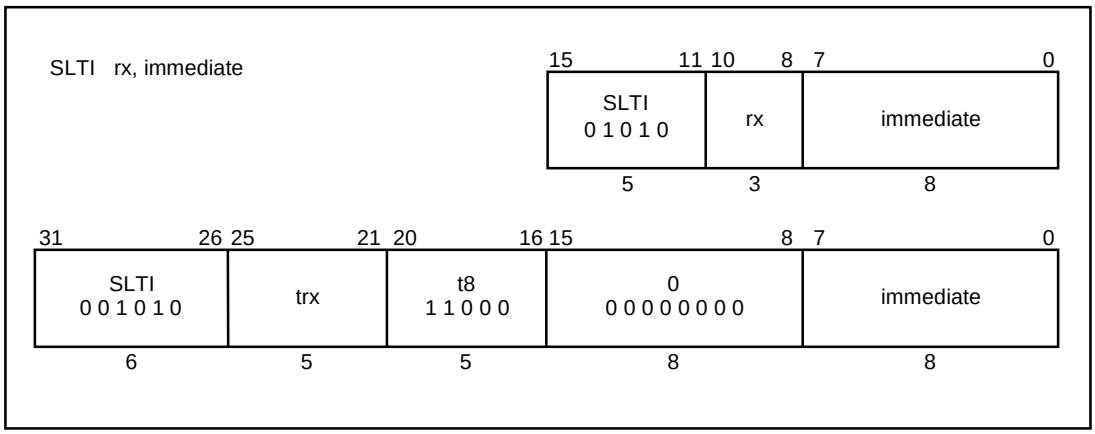
# SLT

**Set on Less Than**



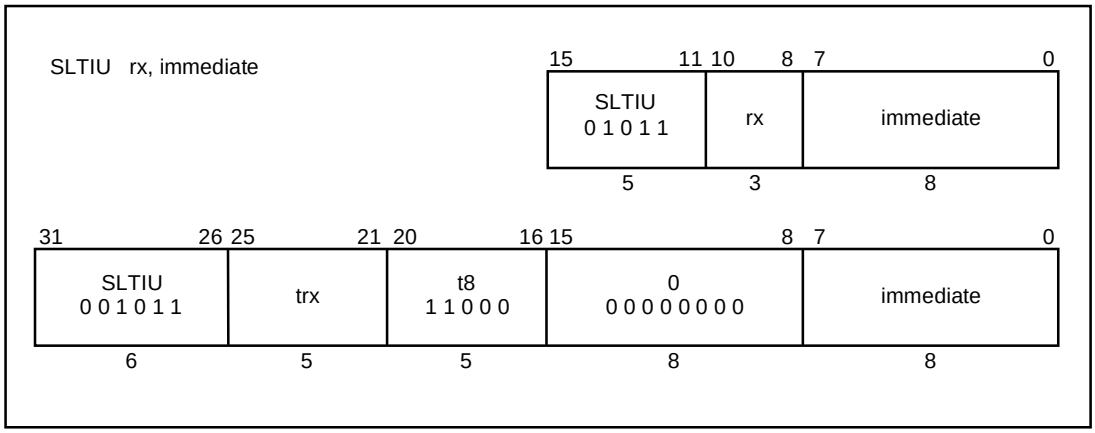

# SLTI

**Set on Less Than Immediate**



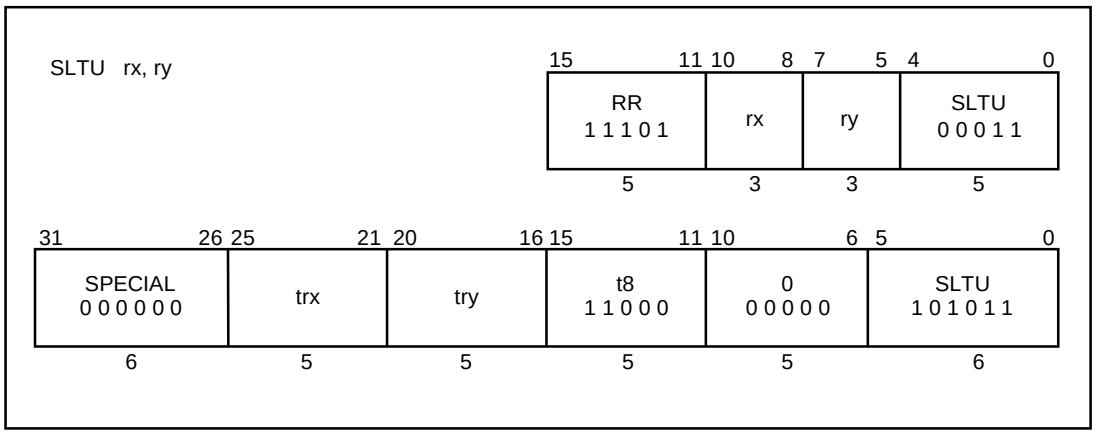
# SLTIU

**Set on Less Than Immediate Unsigned**




# SLTU

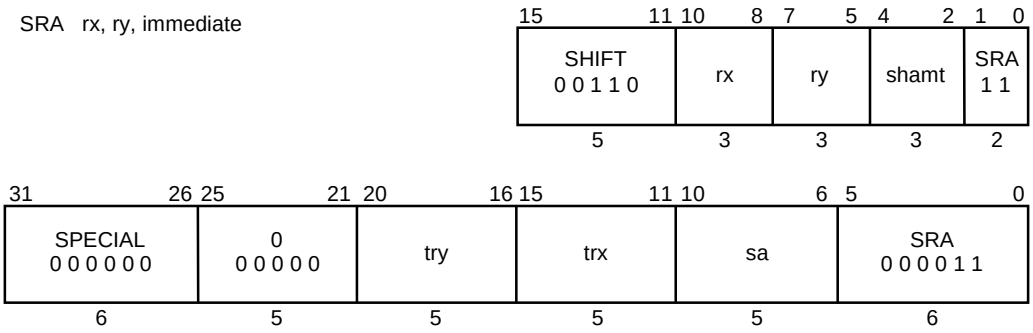
**Set on Less Than Unsigned**



# SRA

**Shift Right Arithmetic**

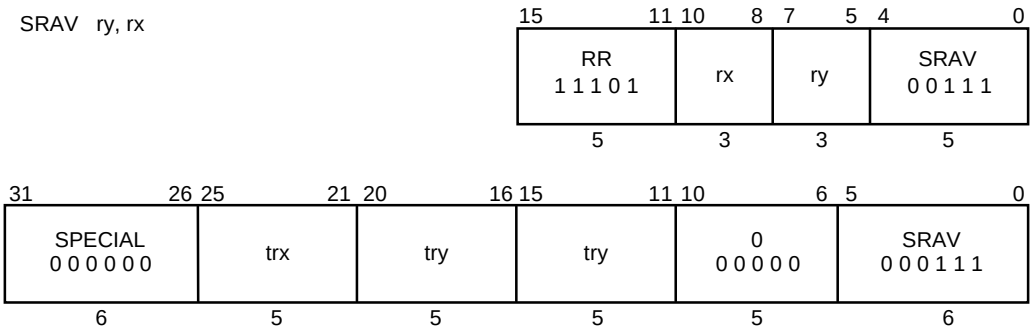
SRA rx, ry, immediate




# SRAV

**Shift Right Arithmetic Variable**

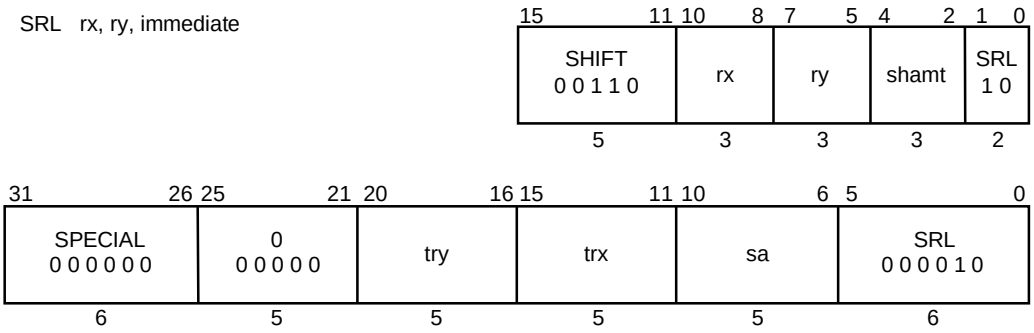
SRAV ry, rx



# SRL

**Shift Right Logical**

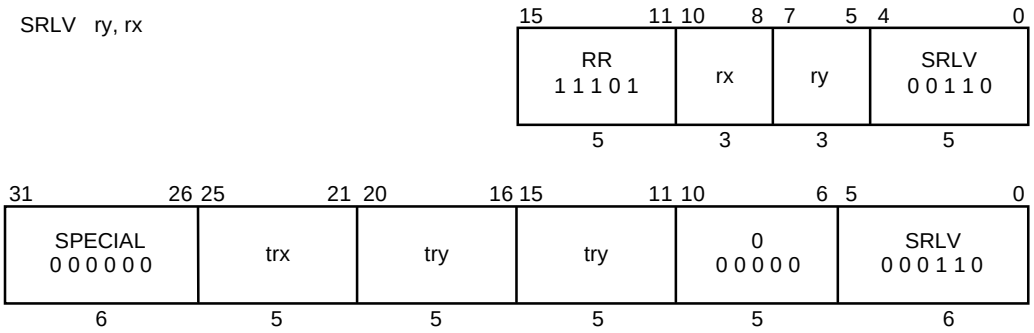
SRL rx, ry, immediate




# SRLV

**Shift Right Logical Variable**

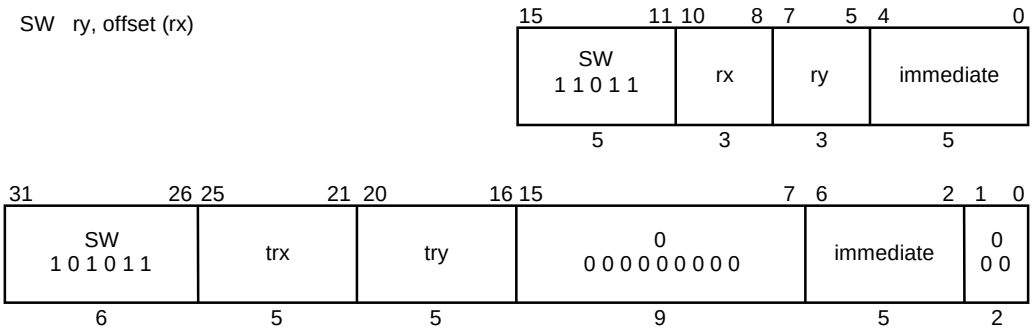
SRLV ry, rx



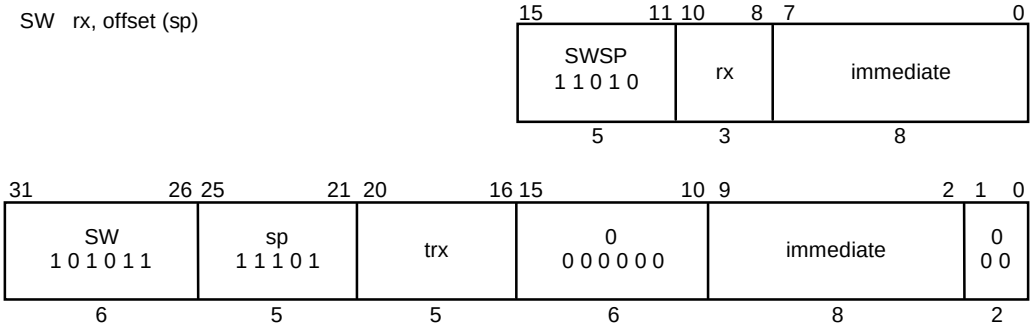
# SW

**Store Word**

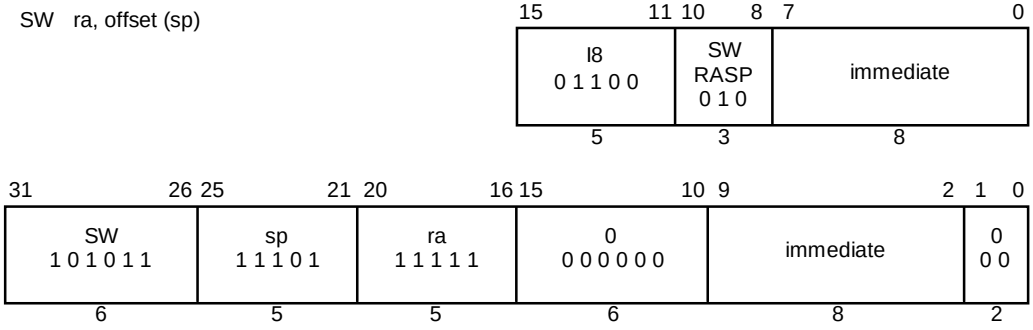
SW ry, offset (rx)




SW rx, offset (sp)

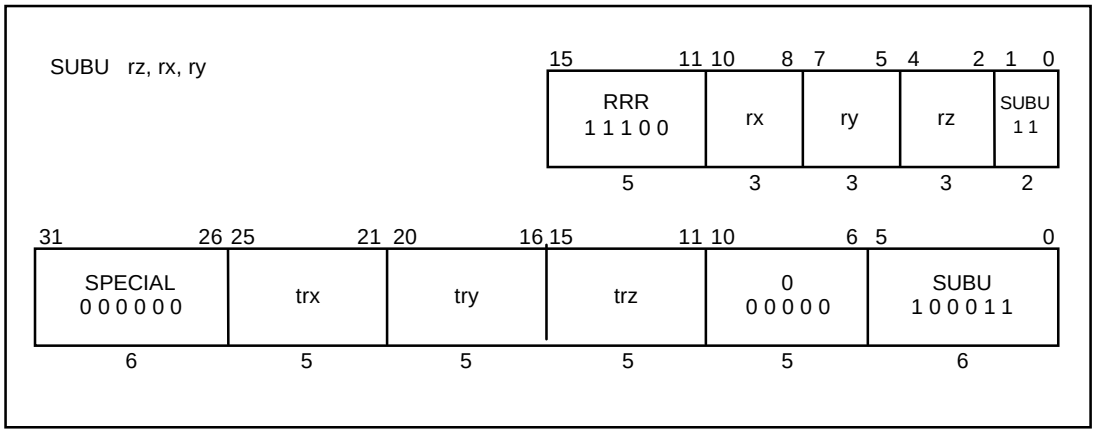



SW ra, offset (sp)



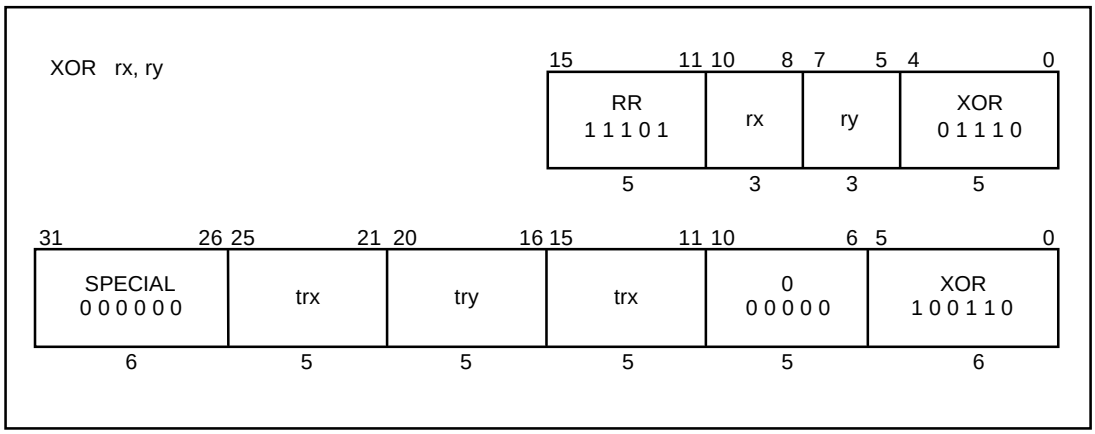
# SUBU

**Subtract Unsigned**




# XOR

**Exclusive OR**



# APPENDIX C VR4120A COPROCESSOR 0 HAZARDS

The VR4120A core avoids contention of its internal resources by causing a pipeline interlock in such cases as when the contents of the destination register of an instruction are used as a source in the succeeding instruction. Therefore, instructions such as NOP must not be inserted between instructions.

However, interlocks do not occur on the operations related to the CP0 registers and the TLB. Therefore, contention of internal resources should be considered when composing a program that manipulates the CP0 registers or the TLB. The CP0 hazards define the number of NOP instructions that is required to avoid contention of internal resources, or the number of instructions unrelated to contention. This chapter describes the CP0 hazards.

The CP0 hazards of the VR4120A core are as or less stringent than those of the VR4000. Table C-1 lists the Coprocessor 0 hazards of the VR4120A core. Code that complies with these hazards will run without modification on the VR4000.

The contents of the CP0 registers or the bits in the "Source" column of this table can be used as a source after they are fixed.

The contents of the CP0 registers or the bits in the "Destination" column of this table can be available as a destination after they are stored.

Based on this table, the number of NOP instructions required between instructions related to the TLB is computed by the following formula, and so is the number of instructions unrelated to contention:

(Destination Hazard number of A) – [(Source Hazard number of B) + 1]

As an example, to compute the number of instructions required between an MTC0 and a subsequent MFC0 instruction, this is:

(5) – (3 + 1) = 1 instruction

The CP0 hazards do not generate interlocks of pipeline. Therefore, the required number of instruction must be controlled by program.

**Table C-1. VR4120A CPU Coprocessor 0 Hazards**

| Operation | Source | | Destination | |
|---|---|---|---|---|
| | Source Name | No. of Cycles | Destination Name | No. of Cycles |
| MTC0 | | | cpr rd | 5 |
| MFC0 | cpr rd | 3 | | |
| TLBR | Index, TLB | 2 | PageMask, EntryHi, EntryLo0, EntryLo1 | 5 |
| TLBWI<br>TLBWR | Index or Random, PageMask, EntryHi, EntryLo0, EntryLo1 | 2 | TLB | 5 |
| TLBP | PageMask, EntryHi | 2 | Index | 6 |
| ERET | EPC or ErrorEPC, TLB | 2 | Status.EXL, Status.ERL | 4 |
| | Status | 2 | | |
| CACHE Index Load Tag | | | TagLo, TagHi, PErr | 5 |
| CACHE Index Store Tag | TagLo, TagHi, PErr | 3 | | |
| CACHE Hit ops. | cache line | 3 | cache line | 5 |
| Coprocessor usable test | Status.CU, Status.KSU, Status.EXL, Status.ERL | 2 | | |
| Instruction fetch | EntryHi.ASID, Status.KSU, Status.EXL, Status.ERL, Status.RE, Config.K0C | 2 | | |
| | TLB | 2 | | |
| Instruction fetch exception | | | EPC, Status | 4 |
| | | | Cause, BadVAddr, Context, XContext | 5 |
| Interrupt signals | Cause.IP, Status.IM, Status.IE, Status.EXL, Status.ERL | 2 | | |
| Load/Store | EntryHi.ASID, Status.KSU, Status.EXL, Status.ERL, Status.RE, Config.K0C, TLB | 3 | | |
| | Config.AD, Config.EP | 3 | | |
| | WatchHi, WatchLo | 3 | | |
| Load/Store exception | | | EPC, Status, Cause, BadVAddr, Context, XContext | 5 |
| TLB shutdown | | | Status.TS | 2 (Inst.),<br>4 (Data) |

**Cautions 1. If the setting of the K0 bit in the Config register is changed to uncached mode by MTC0, the accessed memory area is switched to the uncached one at the instruction fetch of the third instruction after MTC0.**

**2. A stall of several instructions occurs if a jump or branch instruction is executed immediately after the setting of the ITS bit in the Status register.**

**Remarks 1.** The instruction following MTC0 must not be MFC0.

**2.** The five instructions following MTC0 to Status register that changes KSU and sets EXL and ERL may be executed in the new mode, and not kernel mode. This can be avoided by setting EXL first, leaving KSU set to kernel, and later changing KSU.

**3.** There must be two non-load, non-CACHE instructions between a store and a CACHE instruction directed to the same primary cache line as the store.

The status during execution of the following instruction for which CP0 hazards must be considered is described below.

**(1) MTC0**

Destination: The completion of writing to a destination register (CP0) of MTC0.

**(2) MFC0**

Source: The confirmation of a source register (CP0) of MFC0.

**(3) TLBR**

Source: The confirmation of the status of TLB and the Index register before the execution of TLBR.

Destination: The completion of writing to a destination register (CP0) of TLBR.

**(4) TLBWI, TLBWR**

Source: The confirmation of a source register of these instructions and registers used to specify a TLB entry.

Destination: The completion of writing to TLB by these instructions.

**(5) TLBP**

Source: The confirmation of the PageMask register and the EntryHi register before the execution of TLBP.

Destination: The completion of writing the result of execution of TLBP to the Index register.

**(6) ERET**

Source: The confirmation of registers containing information necessary for executing ERET.

Destination: The completion of the processor state transition by the execution of ERET.

**(7) CACHE Index Load Tag**

Destination: The completion of writing the results of execution of this instruction to the related registers.

**(8) CACHE Index Store Tag**

Source: The confirmation of registers containing information necessary for executing this instruction.

**(9) Coprocessor Usable Test**

Source: The confirmation of modes set by the bits of the CP0 registers in the "Source" column.

**Examples 1.** When accessing the CP0 registers in User mode after the contents of the CU0 bit of the Status register are modified, or when executing an instruction such as TLB instructions, CACHE instructions, or branch instructions that use the resource of the CP0.

**2.** When accessing the CP0 registers in the operating mode set in the Status register after the KSU, EXL, and ERL bits of the Status register are modified.

**(10) Instruction Fetch**

Source: The confirmation of the operating mode and TLB necessary for instruction fetch.

**Examples 1.** When changing the operating mode from User to Kernel and fetching instructions after the KSU, EXL, and ERL bits of the Status register are modified.

**2.** When fetching instructions using the modified TLB entry after TLB modification.

**(11) Instruction Fetch Exception**

Destination: The completion of writing to registers containing information related to the exception when an exception occurs on instruction fetch.

**(12) Interrupts**

Source: The confirmation of registers judging the condition of occurrence of interrupt when an interrupt factor is detected.

**(13) Loads/Sores**

Source: The confirmation of the operating mode related to the address generation of Load/Store instructions, TLB entries, the cache mode set in the K0 bit of the Config register, and the registers setting the condition of occurrence of a Watch exception.

**Example** When Loads/Stores are executed in the kernel field after changing the mode from User to Kernel.

**(14) Load/Store Exception**

Destination: The completion of writing to registers containing information related to the exception when an exception occurs on load or store operation.

**(15) TLB Shutdown**

Destination: The completion of writing to the TS bit of the Status register when a TLB shutdown occurs.

Table C-2 indicates examples of calculation.

**Table C-2. Calculation Example of CP0 Hazard and Number of Instructions Inserted**

| Destination | Source | Contending Internal Resource | Number of Instructions Inserted | Formula |
|---|---|---|---|---|
| TLBWR/TLBWI | TLBP | TLB Entry | 2 | 5 – (2 + 1) |
| TLBWR/TLBWI | Load or Store using newly modified TLB | TLB Entry | 1 | 5 – (3 + 1) |
| TLBWR/TLBWI | Instruction fetch using newly modified TLB | TLB Entry | 2 | 5 – (2 + 1) |
| MTC0, Status [CU] | Coprocessor instruction that requires the setting of CU | Status [CU] | 2 | 5 – (2 + 1) |
| TLBR | MFC0 EntryHi | EntryHi | 1 | 5 – (3 + 1) |
| MTC0 EntryLo0 | TLBWR/TLBWI | EntryLo0 | 2 | 5 – (2 + 1) |
| TLBP | MFC0 Index | Index | 2 | 6 – (3 + 1) |
| MTC0 EntryHi | TLBP | EntryHi | 2 | 5 – (2 + 1) |
| MTC0 EPC | ERET | EPC | 2 | 5 – (2 + 1) |
| MTC0 Status | ERET | Status | 2 | 5 – (2 + 1) |
| MTC0 Status [IE] Note | Instruction that causes an interrupt | Status [IE] | 2 | 5 – (2 + 1) |

**Note** The number of hazards is undefined if the instruction execution sequence is changed by exceptions. In such a case, the minimum number of hazards until the IE bit value is confirmed may be the same as the maximum number of hazards until an interrupt request occurs that is pending and enabled.

**[MEMO]**

NEC

# Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

*Thank you for your kind support.*


**North America**
NEC Electronics Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288
1-408-588-6130

**Europe**
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

**South America**
NEC do Brasil S.A.
Fax: +55-11-6465-6829

**Hong Kong, Philippines, Oceania**
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

**Korea**
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: 02-528-4411

**Taiwan**
NEC Electronics Taiwan Ltd.
Fax: 02-2719-5951

**Asian Nations except Philippines**
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

**Japan**
NEC Semiconductor Technical Hotline
Fax: 044-548-7900


I would like to report the following error/make the following suggestion:

Document title: _____

Document number: _____ Page number: _____

If possible, please fax the referenced page or drawing.

| Document Rating | Excellent | Good | Acceptable | Poor |
|---|---|---|---|---|
| Clarity | ❑ | ❑ | ❑ | ❑ |
| Technical Accuracy | ❑ | ❑ | ❑ | ❑ |
| Organization | ❑ | ❑ | ❑ | ❑ |