

1

PRODUCT OVERVIEW

INTRODUCTION

Samsung's KS32C6200 32-bit RISC microcontroller is a cost-effective and high-performance microcontroller solution for general-purpose applications or ink-jet printers. The KS32C6200 provides two-channel UART, two-channel DMA, ROM/SRAM/DRAM controller, three-channel timer, parallel port interface, programmable I/O ports and peripherals for ink-jet printers.

An outstanding feature of the KS32C6200 is its CPU core, a 32-bit RISC processor (ARM7TDMI) designed by Advanced RISC Machines, Ltd. The ARM7TDMI core is a low-power, general-purpose, microprocessor macro-cell that was developed for use in application-specific and customer-specific integrated circuits. Its simple, elegant, and fully static design is particularly suitable for cost-sensitive and power-sensitive applications.

The KS32C6200 was developed using the ARM7TDMI core 0.5- μ m CMOS standard cells, and a memory compiler. Most of the on-chip function blocks were designed using an HDL synthesizer. The KS32C6200 has been fully verified in Samsung's MCU test environment.

By providing a complete set of common system peripherals, the KS32C6200 minimizes overall system costs and eliminates the need to configure additional components.

The integrated on-chip functions that are described in this document include:

- ROM/SRAM/ DRAM controller
- Two K-byte instruction/data cache and controller
- Derasterizer
- Shifter
- Two-channel DMA controller
- Two-channel UART (asynchronous serial I/O)
- Three 16-bit timers
- Tone generator
- Parallel port interface controller (PPIC)
- Watch-dog timer
- Interrupt controller
- Programmable I/O ports

FEATURES

Architecture

- Completely integrated system for embedded applications
- Fully 16-bit/32-bit RISC architecture
- Efficient and powerful ARM7TDMI CPU core
- Cost-effective JTAG-based debugging solution

System Manager

- 32 M-byte address space
- 8-bit or 16-bit external bus support for ROM, SRAM, DRAM, and external I/Os
- Programmable memory access cycles and chip-select logics
- Supports EDO DRAM and self-refresh mode DRAM
- Supports asymmetric or symmetric address DRAM
- Cost-effective memory-to-peripheral interface

Instruction/Data Cache

- Two way set associative cache with 2 K bytes (512 instruction/data words)
- Pseudo LRU (Least Recently Used)
- Four depth write buffer

DMA (Direct Memory Access) Controller

- Two-channel general-purpose DMA controller
- Memory-to-memory, parallel port-to-memory, memory-to-parallel port, UART-to-memory, and memory-to-UART data transfers without CPU intervention
- Initiated by software or external DMA request
- Increments or decrements source or 8-bit, 16-bit and 32-bit data transfers

Derasterizer/Shifter

- 16-bit X 16-bit rotation by 90/270 degree for raster data rotation
- Reverses 16-bit data
- Left/right shifting/rotating 7 half words to the selective direction

Parallel Port Interface Controller

- DMA-based or interrupt-based operation
- Supports IEEE Standard 1284 communication

modes (compatibility mode, nibble mode, byte mode, and ECP mode)

- Supports ECP protocol with or without run-length encoding (RLE)
- Automatic handshaking mode for any forward or reverse protocol with software enable/disable

UART (Serial I/O)

- Two-channel UART (serial I/O port) with DMA-based or interrupt-based operation; supports 5-bit, 6-bit, 7-bit, or 8-bit serial data transmit/receive
- Programmable baud rates
- Infra-red (IR) Tx/Rx support

Tone Generator

- Programmable square wave generator

Timers

- Three programmable 16-bit timers
- Interval mode operation

I/O Ports

- Five programmable I/O ports (GIOP)
- Six input ports (GIP)
- 13 output ports (GOP)
- 10 extra-output ports (EOPA, EOPB)
- Each port pin can be configured individually as input, output, or I/O for a dedicated signal

Watch-Dog Timer

- 16-bit watch-dog timer for periodic reset or interrupts

Interrupts

- 15 interrupt sources (including two external interrupt sources)
- Normal or fast interrupt modes (IRQ, FIQ)

Operating Voltage Range

- 4.75 to 5.25 volts

Operating Frequency

- Up to 33 MHz

Package Type

- 160-pin TQFP

BLOCK DIAGRAM

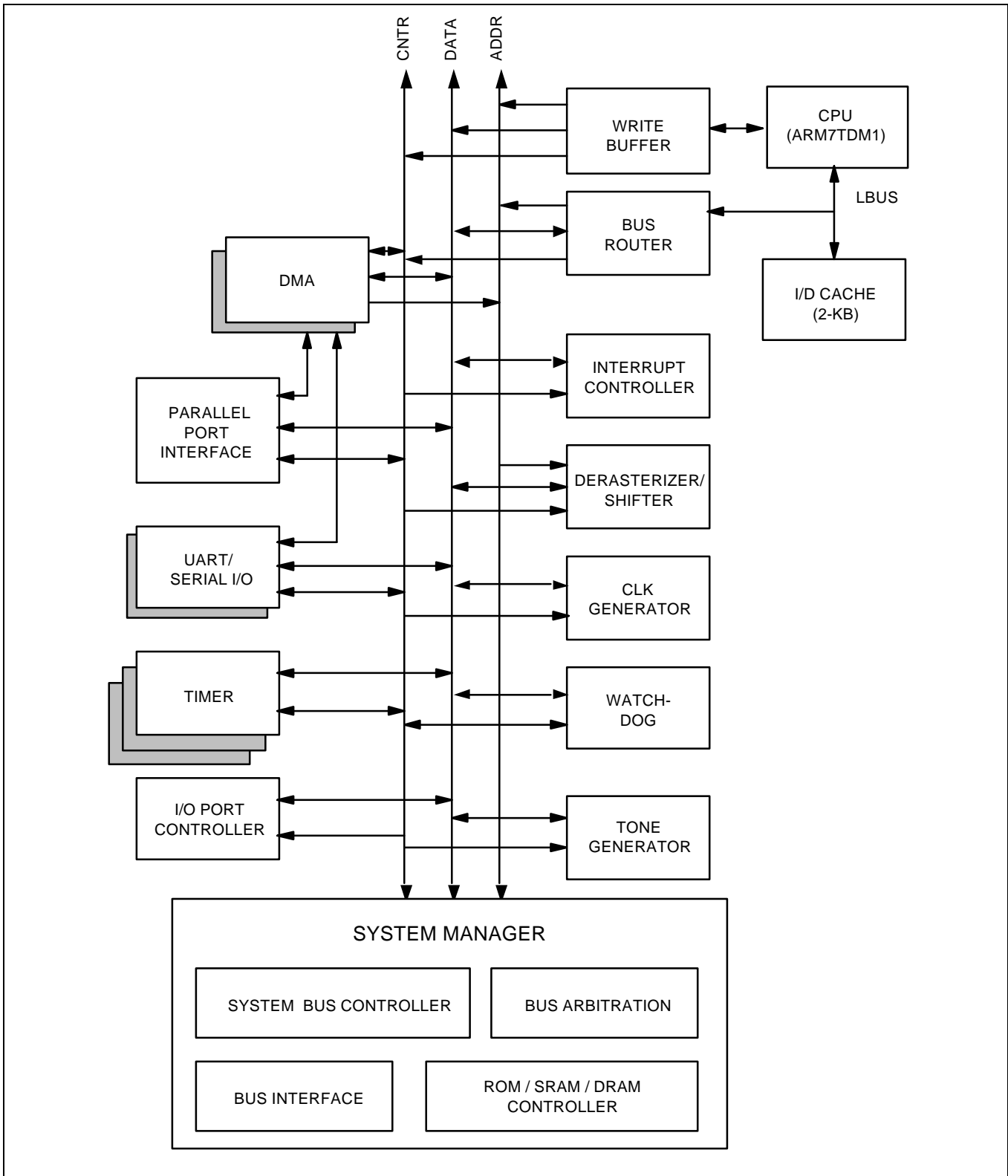


Figure 1-1. KS32C6200 Block Diagram

PIN ASSIGNMENTS

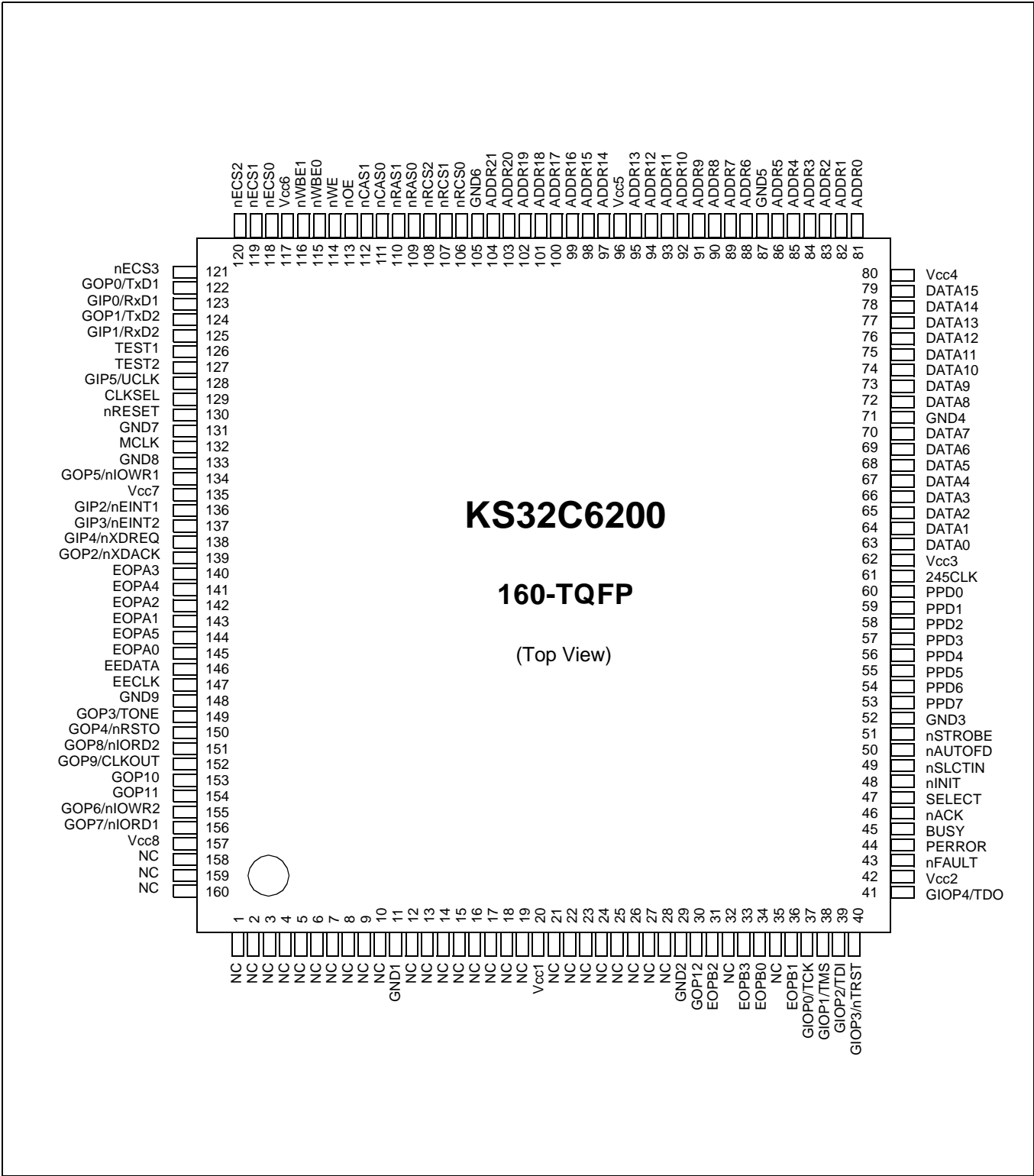


Figure 1-2. KS32C6200 Pin Assignments

SIGNAL DESCRIPTIONS

Table 1-1. KS32C6000 Signal Descriptions

Signal	Pin Number	I/O Pin Type	Description
MCLK	132	I ₂	KS32C6200 master clock. It has a 50% duty cycle and a maximum operating frequency of 33 MHz. When the CLKSEL is "1", the maximum MCLK frequency is 66 MHz.
CLKSEL	129	I ₂	Clock select. When CLKSEL is "0", MCLK is used as the master clock. When CLKSEL is "1", the MCLK/2, internally divided by two, is used as the master clock.
nRESET	130	I ₄	Not reset. nRESET is the global reset input for the KS32C6200. For a system reset, nRESET must be held to Low level for at least 65 machine cycles.
nSLCTIN	49	I ₁	Not select information. This input signal is used by parallel port interface to request "on-line" status information.
nSTROBE	51	I ₁	Not strobe. The nSTROBE input indicates when valid data is present on the parallel port data bus, PPD[7:0].
nAUTOFD	50	I ₁	Not autofeed. The nAUTOFD input indicates whether data on the parallel port data bus, PPD[7:0], is an autofeed command. Otherwise, the bus signals are interpreted as data only.
nINIT	48	I ₁	Not parallel port initialization. The nINIT input signal initializes the parallel port's input control.
nACK	46	O ₁	Not parallel port acknowledge. The nACK output signal is issued whenever a transfer on the parallel port data bus is completed.
BUSY	45	O ₁	Parallel port busy. The BUSY output signal indicates that the KS32C6200 parallel port is currently busy.
SELECT	47	O ₁	Parallel port select. The SELECT output signal indicates whether the device connected to the KS32C6200 parallel port is "on-line" or "off-line".
PERROR	44	O ₁	Parallel port paper error. PERROR output indicates that a problem exists with the paper in the printer. It could indicate that the printer has a paper jam or that the printer is out of paper.
nFAULT	43	O ₁	Not parallel port fault. The nFAULT output indicates that an error condition exists with the printer. This signal can be used to indicate that the printer is out of ink or to inform the user that the printer is not turned on.
PPD[7:0]	53–60	I/O ₁	Parallel port data bus. This 8-bit, tri-state bus is used to exchange data between the KS32C6200 and an external host (peripheral).
245CLK	61	I/O ₄ ⁽¹⁾	Output for the direction of transceiver connected to PPD[7:0]
EECLK	147	O ₁	Clock line to the serial EEPROM. You can generate this signal by software.
EEDATA	146	I/O ₂	Data interface line with the serial EEPROM. You can generate this signal by software.

Table 1-1. KS32C6000 Signal Descriptions

Signal	Pin Number	I/O Pin Type	Description
ADDR[21:0]	81–86 88–95 97–104	I/O ₃ ⁽⁴⁾	KS32C6200 address bus. The 22-bit address data bus, ADDR[21:0], covers the full 4 M half-word (16-bit) address range of each ROM/SRAM, DRAM, and external I/O bank. A byte access for the DRAM can be discriminated by CAS[1:0] and SRAM/ROM by WBE[1:0].
DATA[15:0]	63–70 72–79	I/O ₃	External bi-directional 16-bit data bus.
nRAS[1:0]	109–110	O ₁	Not row address strobe for DRAM. The KS32C6200 supports up to two DRAM banks. Each nRAS output is corresponding to each bank.
nCAS[1:0]	111–112	O ₁	Not column address strobe for DRAM. The two nCAS outputs indicate the byte selections whenever a DRAM bank is accessed.
nOE	113	I/O ₄ ⁽¹⁾	Not output enable. Whenever a memory access occurs, the nOE output controls the output enable port of the specific memory device.
nWE	114	O ₁	Not write enable. Whenever a memory access occurs, the nWE output controls the write enable port of the specific memory device.
nWBE[1:0]	115–116	I/O ₂ ⁽¹⁾	Not byte write enable.
RXD1/GIP[0]	123	I ₄	Receive data input for the UART0. RXD1 is the channel of UART0 input signal to receive a serial data. / General input port 0.
RXD2/GIP[1]	125	I ₄	Receive data input for the UART1. RXD2 is the channel of UART1 input signal to receive serial data. / General input port 1.
nEINT1/GIP[2]	136	I ₃	External interrupt request input. / General input port 2.
nEINT2/GIP[3]	137	I ₃	External interrupt request input. / General input port 3.
nXDREG/GIP[4]	138	I ₃	External DMA request. / General input port 4.
UCLK/GIP[5]	128	I ₁	The external UART clock input. MCLK can be used as the UART clock. You can use UCLK, with an appropriate divided by factor, if a very precious baud rate clock is required. / General input port 5.
TXD1/GOP[0]	122	O ₁	Transmit data output for the UART0. TXD1 is the channel of UART0 output to transmit a serial data. / General output port 0.
TXD2/GOP[1]	124	O ₁	Transmit data output for the UART1. TXD2 is the channel of UART1 output to transfer a serial data. / General output port 1.
nXDACK/GOP[2]	139	O ₁	DMA acknowledge. This active low output signal is generated whenever a DMA transfer operation is completed. / General output port 2.
TONE/GOP[3]	149	O ₁	Tone generator output. / General output port 3.
nRSTO/GOP[4]	150	O ₃ ⁽³⁾	Reset-out from watch-dog timer. / General output port 4.
nIOWR1/GOP[5]	134	O ₁	Write strobe to the sub-region 1 of I/O bank 3. / General output port 5.
nIOWR2/GOP[6]	155	O ₁	Write strobe to the sub-region 2 of I/O bank 3. / General output port 6.
nIORD1/GOP[7]	156	O ₁	Read strobe from the sub-region 1 of I/O bank 3. / General output port 7.
nIORD2/GOP[8]	151	O ₁	Read strobe from the sub-region 2 of I/O bank 3. / General output port 8.

Table 1-1. KS32C6000 Signal Descriptions

Signal	Pin Number	I/O Pin Type	Description
CLKOUT/GOP[9]	152	O ₁	Clock for external chip. / General output port 9.
GOP[10]	153	O ₁	General output port 10
GOP[11]	154	O ₁	General output port 11
GOP[12]	30	O ₁	General output port 12
GIOP[4:0]/JTAG	37–41	I/O ₅	General-purpose input/output port. / JTAG test logic (TCK, TMS, TDI, nTRST, TDO)
EOPA[5:0]	140–145	O ₁	Extra output port A
EOPB[3:0]	31,33 34,36	O ₁	Extra output port B
TEST1	126	I ₂	Test 1 pin. This pin should be connected to GND for a normal operation.
TEST2	127	I ₂	Test 2 pin. This pin should be connected to GND for a normal operation.
nECS[3:0]	118–121	O ₁	Not external chip select. Four I/O banks are provided for external memory-mapped I/O operations. Each I/O bank contains up to 4 M half-words. The nECS signals indicate that an external I/O bank is selected.
nRCS[2]	108	O ₂	Not ROM/SRAM chip select. The KS32C6200 can access up to three external ROM/SRAM banks. The nRCS[0] corresponds to the ROM/SRAM bank 0, the nRCS[1] to bank 1, and nRCS[2] to bank 2.
nRCS[1:0]	106–107	O ₁	
VCC	20,42,62 ,80,96, 117,135, 157	PWR	System power provides the normal DC supply. Externally connected to the VCC board plane.
GND	11,29,52 ,71,87, 105,131, 133,148	PWR	System ground. Externally connected to the ground board plane.

NOTES

1. The pins are input only at the chip test mode.
2. The prefix “n” in the pin descriptions shows that the pins are active low signals.
3. O₃ is open-drain output. Pull-up resistor must be installed.

PIN TYPE DESCRIPTION

Table 1-2. I/O Type Description

Pin Type	Description
I ₁	TTL schmitt trigger level input buffer
I ₂	TTL level input buffer
I ₃	TTL schmitt trigger level input buffer with pull-up resistor
I ₄	CMOS schmitt trigger level input
O ₁	Normal output buffer
O ₂	Tri-state output buffer
O ₃	Open-drain output buffer
I/O ₁	TTL schmitt trigger level input with pull-up resistor and tri-state output with medium slew-rate
I/O ₂	TTL level with pull-up resistor and tri-state output
I/O ₃	TTL schmitt trigger level input with pull-up resistor and tri-state output with medium slew-rate
I/O ₄	TTL schmitt trigger level input with pull-up resistor and tri-state output
I/O ₅	TTL level input and tri-state output

KS32C6200 CPU CORE

OVERVIEW

The KS32C6200 microcontroller uses the ARM7TDMI processor, designed by Advanced RISC Machines, Ltd., as its CPU core. Samsung's product design offers the advantages of small size, low power consumption, and low price for high-performance devices such as laser beam printers and ink-jet printers.

The ARM7TDMI core is a fully static CMOS implementation. This implementation allows the system clock to be stopped in any part of the cycle with extremely low residual power consumption and no loss of state. The core's architecture is based on Reduced Instruction Set Computer (RISC) principles.

The instruction set and its related decode mechanism are, therefore, much simpler than microprogramming Complex Instruction Set Computer (CISC) systems. This results in a high instruction throughput and impressive real-time interrupt response. The ARM7TDMI has a 32-bit address bus.

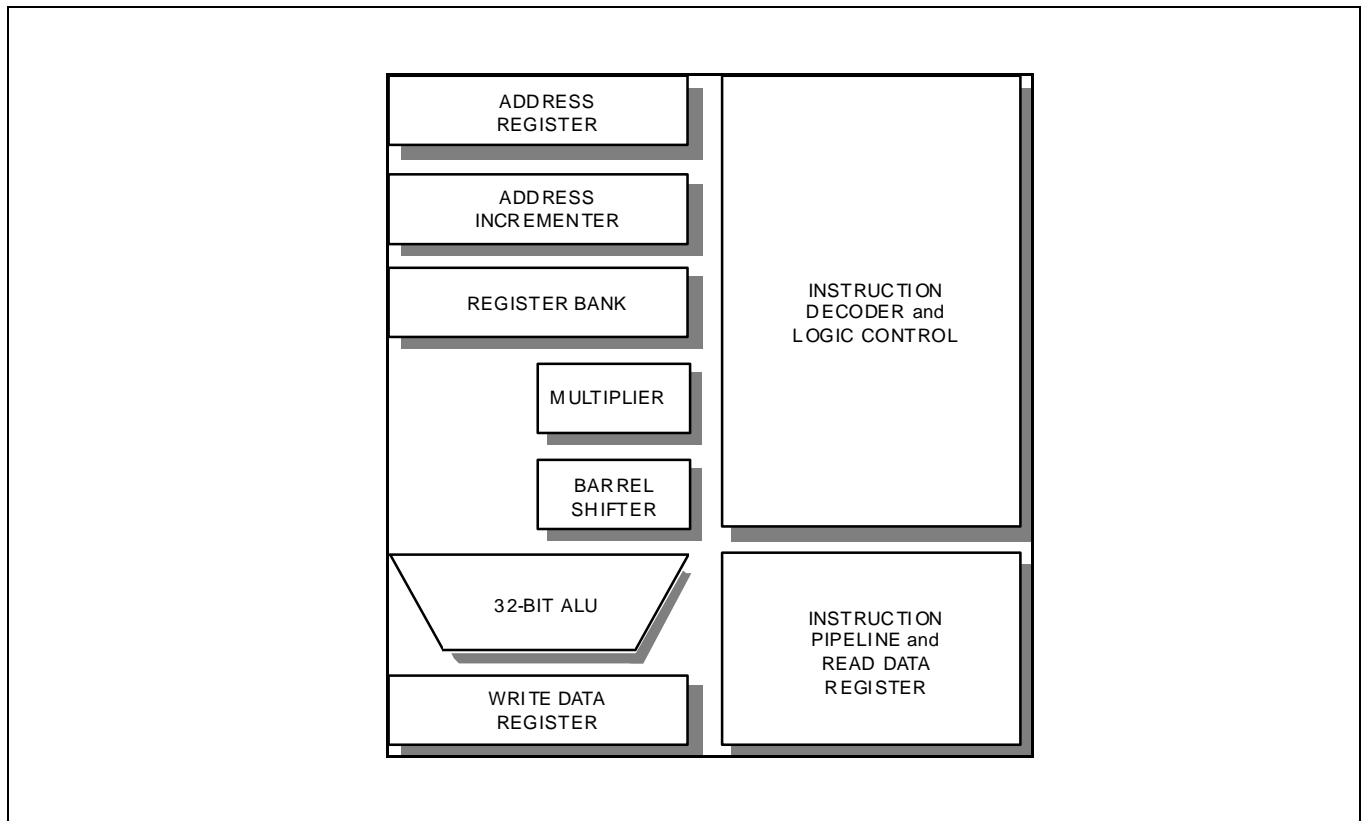


Figure 1-3. ARM7TDMI CPU Core Block Diagram

INSTRUCTION SET

The KS32C6200 instruction set has eleven basic instruction types:

- Two instruction types use the on-chip arithmetic logic unit, barrel shifter, and multiplier to perform high-speed operations on the data in a bank of 31 registers, each 32-bit wide.
- Three types control data transfer between memory and the registers. One is optimized for flexibility of addressing, another for rapid context switching, and the third for swapping data.
- Three types control the flow and privilege level of program execution.

Three types are dedicated to the control of external coprocessors. These instructions extend the off-chip functionality of the instruction set in an open and uniform way.

The ARM instruction set is a good target for compilers of many different high-level languages. Where required for critical code segments, assembly code programming is also straightforward, unlike some RISC processors which depend on sophisticated compiler technology to manage complicated instruction interdependencies.

Pipelining is employed so that all parts of the processor and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory.

MEMORY INTERFACE

The CPU memory interface has been designed to allow the performance potential to be realized without incurring high costs in the memory system. Speed-critical control signals are pipelined to allow system control functions to be implemented in standard low-power logic, and these control signals facilitate the exploitation of the fast local access modes offered by industry standard dynamic RAMs.

OPERATING MODES

The CPU core supports a 32-bit data bus and a 32-bit address bus. The data types the processor supports are bytes (8 bits) and words (32 bits), where words must be aligned to four-byte boundaries.

Instructions are exactly one word, and data operations such as ADD are only performed on word quantities. Loads and stores can transfer bytes or words. The CPU supports six operating modes, five of which are visible to the programmer:

- User mode: the normal program execution state
- FIQ (Fast Interrupt Request) mode: designed to support a data transfer or channel process
- IRQ (Interrupt ReQuest) mode: used for general purpose interrupt handling
- Supervisor mode: a protected mode for the operating system

Undefined mode: entered when an undefined instruction is executed

SPECIAL REGISTERS

Table 1-3. Special Function Registers

Group	Register	Offset	R/W	Description	Reset Value
System Manager	SYSCFG	0000h	R/W	System Register Access Configuration register	1001h
	ROMCON	3000h	R/W	ROM control register	02003002h
	SRAMCON0	3004h	R/W	SRAM control register 0	00000000h
	SRAMCON1	3008h	R/W	SRAM control register 1	00000000h
	DRAMCON0	301ch	R/W	DRAM control register 0	00000000h
	DRAMCON1	3020h	R/W	DRAM control register 1	00000000h
	REFCON	3024h	R/W	DRAM refresh control register	00000000h
	EXTCON0	300ch	R/W	I/O bank 0 control register	00000000h
	EXTCON1	3010h	R/W	I/O bank 1 control register	00000000h
	EXTCON2	3014h	R/W	I/O bank 2 control register	00000000h
	EXTCON3	3018h	R/W	I/O bank 3 control register	00000000h
Cache	CACHNAB0	1000h	R/W	Non-cacheable area begin 0 register	00000000h
	CACHNAE0	1800h	R/W	Non-cacheable area end 0 register	00000000h
	CACHNAB1	2000h	R/W	Non-cacheable area begin 1 register	00000000h
	CACHNAE1	2800h	R/W	Non-cacheable area end 1 register	00000000h
Derasterizer	DRAST0	6000h	R/W	Derasterizer data register 0	xxxxh
	DRAST1	6004h	R/W	Derasterizer data register 1	xxxxh
	DRAST2	6008h	R/W	Derasterizer data register 2	xxxxh
	DRAST3	600ch	R/W	Derasterizer data register 3	xxxxh
	DRAST4	6010h	R/W	Derasterizer data register 4	xxxxh
	DRAST5	6014h	R/W	Derasterizer data register 5	xxxxh
	DRAST6	6018h	R/W	Derasterizer data register 6	xxxxh
	DRAST7	601ch	R/W	Derasterizer data register 7	xxxxh
	DRAST8	6020h	R/W	Derasterizer data register 8	xxxxh
	DRAST9	6024h	R/W	Derasterizer data register 9	xxxxh
	DRAST10	6028h	R/W	Derasterizer data register 10	xxxxh
	DRAST11	602ch	R/W	Derasterizer data register 11	xxxxh
	DRAST12	6030h	R/W	Derasterizer data register 12	xxxxh
	DRAST13	6034h	R/W	Derasterizer data register 13	xxxxh
	DRAST14	6038h	R/W	Derasterizer data register 14	xxxxh
	DRAST15	603ch	R/W	Derasterizer data register 15	xxxxh

Table 1-3. Special Function Registers

Group	Register	Offset	R/W	Description	Reset Value
Shift Control	DATARVS	7000h	R/W	Data reverser	0000h
	SFTCON	7004h	R/W	Shift control register	0h
	SFTCNT	7008h	R/W	Shift count register	00h
	SFTDATA0	700ch	R/W	Shift word data register 0	xxxxxxxxh
	SFTDATA1	7010h	R/W	Shift word data register 1	xxxxxxxxh
	SFTDATA2	7014h	R/W	Shift word data register 2	xxxxxxxxh
	SFTDATA3	7018h	R/W	Shift word data register 3	xxxxxxxxh
	SFTDATA4	701ch	R/W	Shift word data register 4	xxxxxxxxh
	SFTDATA5	7020h	R/W	Shift word data register 5	xxxxxxxxh
	SFTDATA6	7024h	R/W	Shift word data register 6	xxxxxxxxh
Timer	TCON	5800h	R/W	System timer control register	000h
	TBCNT0	5804h	R/W	Timer base/count register 0	xxxxh
	TBCNT1	5808h	R/W	Timer base/count register 1	xxxxh
	TBCNT2	581ch	R/W	Timer base/count register 2	xxxxh
DMA	DMACON0	c000h	R/W	DMA0 control register	0000h
	DMASRC0	c004h	R/W	DMA0 source address register	xxxxxxxxh
	DMADST0	c008h	R/W	DMA0 destination address register	xxxxxxxxh
	DMACNT0	c00ch	R/W	DMA0 transfer count register	xxxxxxxxh
	DMACON1	c800h	R/W	DMA1 control register	00000h
	DMASRC1	c804h	R/W	DMA1 source address register	xxxxxxxxh
	DMADST1	c808h	R/W	DMA1 destination address register	xxxxxxxxh
	DMACNT1	c80ch	R/W	DMA1 transfer count register	xxxxxxxxh
Parallel Port	PPDATA	b000h	R/W	Parallel port data register	000h
	PPSTAT	b004h	R/W	Parallel port status register	7e8h
	PPACKWTH	b008h	R/W	Parallel port acknowledge width register	xxxh
	PPCON	b00ch	R/W	Parallel port control register	0000h
	PPINTEN	b010h	R/W	Parallel port enable interrupt event register	000h
	PPINTPND	b014h	R/W	Parallel port interrupt pending register	000h
UART	ULCON0	e000h	R/W	UART channel 0 line control register	00h
	ULCON1	e800h	R/W	UART Channel 1 line control register	00h
	UCON0	e004h	R/W	UART channel 0 control register	00h
	UCON1	e804h	R/W	UART channel 1 control register	00h
	USTAT0	e008h	R	UART channel 0 status register	c0h

Table 1-3. Special Function Registers

Group	Register	Offset	R/W	Description	Reset Value
UART	USTAT1	e808h	R	UART channel 1 status register	c0h
	UTXBUF0	e00ch	W	UART channel 0 transmit buffer register	00h
	UTXBUF1	e80ch	W	UART channel 1 transmit buffer register	00h
	URXBUF0	e010h	R	UART channel 0 receive buffer register	00h
	URXBUF1	e810h	R	UART channel 1 receive buffer register	00h
	UBRDIV0	e014h	R/W	Baud-rate divisor register 0	0001h
	UBRDIV1	e814h	R/W	Baud-rate divisor register 1	0001h
Tone Generator	TONDATA	f004h	R/W	Tone generator data and control register	00h
Watch-Dog Timer	WTCNT	f804h	R/W	Watch-dog timer count register	00000003h
	WTCON	f800h	R/W	Watch-dog timer control register	00000021h
I/O Ports	IOPMOD	4808h	R/W	I/O port mode register	00000000h
	IOPDATA	4804h	R/W	I/O port data register	xxxxh
	TSTCON	4800h	R/W	Test control register	00000600h
	EERAMCON	5000h	R/W	EERAM control register	0000000xh
	EOP	0x8004	R/W	Extra-output port A register	000003c0h
	EOPL	0x8000	R/W	Extra-output port latch register	800h
	EOPB	0x9010	R/W	Extra-output port B register	0000cf0fh
Interrupt Controller	INTMOD	4000h	R/W	Interrupt mode register	00000000h
	INTPND	4004h	R/W	Interrupt pending register	00000000h
	INTMSK	4008h	R/W	Interrupt mask register	00000000h

NOTES

2

Programmer's Model

OVERVIEW

KS32C6200 was developed using the advanced ARM7TDMI core designed by Advanced RISC Machines, Ltd.

PROCESSOR OPERATING STATES

From the programmer's point of view, the ARM7TDMI can be in one of two states:

- *ARM state* which executes 32-bit, word-aligned ARM instructions.
- *THUMB state* which operates with 16-bit, halfword-aligned THUMB instructions. In this state, the PC uses bit 1 to select between alternate halfwords.

NOTE

Transition between these two states does not affect the processor mode or the contents of the registers.

SWITCHING STATE

Entering THUMB State

Entry into THUMB state can be achieved by executing a BX instruction with the state bit (bit 0) set in the operand register.

Transition to THUMB state will also occur automatically on return from an exception (IRQ, FIQ, UNDEF, ABORT, SWI etc.), if the exception was entered with the processor in THUMB state.

Entering ARM State

Entry into ARM state happens:

1. On execution of the BX instruction with the state bit clear in the operand register.
2. On the processor taking an exception (IRQ, FIQ, RESET, UNDEF, ABORT, SWI etc.). In this case, the PC is placed in the exception mode's link register, and execution commences at the exception's vector address.

MEMORY FORMATS

ARM7TDMI views memory as a linear collection of bytes numbered upwards from zero. Bytes 0 to 3 hold the first stored word, bytes 4 to 7 the second and so on. ARM7TDMI can treat words in memory as being stored either in *Big-Endian* or *Little-Endian* format.

NOTE

The KS32C6200 is configured to the big-endian format.

BIG-ENDIAN FORMAT

In Big-Endian format, the most significant byte of a word is stored at the lowest numbered byte and the least significant byte at the highest numbered byte. Byte 0 of the memory system is therefore connected to data lines 31 through 24.

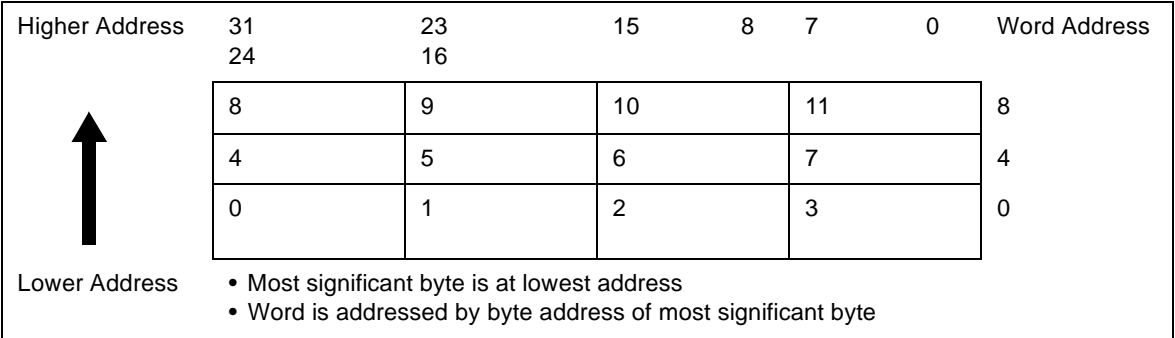


Figure 2-1. Big-Endian Addresses of Bytes within Words

NOTE

The data locations in the external memory are different with Figure 2-1 in the KS32C6200. Please refer to the chapter 4, System Manager.

LITTLE-ENDIAN FORMAT

In Little-Endian format, the lowest numbered byte in a word is considered the word’s least significant byte, and the highest numbered byte the most significant. Byte 0 of the memory system is therefore connected to data lines 7 through 0.

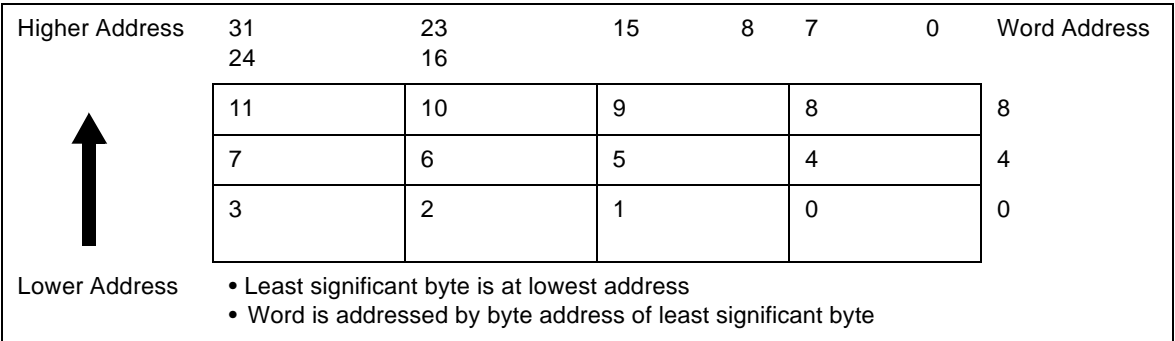


Figure 2-2. Little-Endian Addresses of Bytes within Words

INSTRUCTION LENGTH

Instructions are either 32 bits long (in ARM state) or 16 bits long (in THUMB state).

Data Types

ARM7TDMI supports byte (8-bit), halfword (16-bit) and word (32-bit) data types. Words must be aligned to four-byte boundaries and half words to two-byte boundaries.

OPERATING MODES

ARM7TDMI supports seven modes of operation:

- User (usr): The normal ARM program execution state
- FIQ (fiq): Designed to support a data transfer or channel process
- IRQ (irq): Used for general-purpose interrupt handling
- Supervisor (svc): Protected mode for the operating system
- Abort mode (abt): Entered after a data or instruction prefetch abort
- System (sys): A privileged user mode for the operating system
- Undefined (und): Entered when an undefined instruction is executed

Mode changes may be made under software control, or may be brought about by external interrupts or exception processing. Most application programs will execute in User mode. The non-user modes—known as *privileged modes*—are entered in order to service interrupts or exceptions, or to access protected resources.

REGISTERS

ARM7TDMI has a total of 37 registers - 31 general-purpose 32-bit registers and six status registers - but these cannot all be seen at once. The processor state and operating mode dictate which registers are available to the programmer.

The ARM State Register Set
















In ARM state, 16 general registers and one or two status registers are visible at any one time. In privileged (non-User) modes, mode-specific banked registers are switched in. Figure 2-3 shows which registers are available in each mode: the banked registers are marked with a shaded triangle.

The ARM state register set contains 16 directly accessible registers: R0 to R15. All of these except R15 are general-purpose, and may be used to hold either data or address values. In addition to these, there is a seventeenth register used to store status information






Register 14	is used as the subroutine link register. This receives a copy of R15 when a Branch and Link (BL) instruction is executed. At all other times it may be treated as a general-purpose register. The corresponding banked registers R14_svc, R14_irq, R14_fiq, R14_abt and R14_und are similarly used to hold the return values of R15 when interrupts and exceptions arise, or when Branch and Link instructions are executed within interrupt or exception routines.
Register 15	holds the Program Counter (PC). In ARM state, bits [1:0] of R15 are zero and bits [31:2] contain the PC. In THUMB state, bit [0] is zero and bits [31:1] contain the PC.
Register 16	is the CPSR (Current Program Status Register). This contains condition code flags and the current mode bits.

FIQ mode has seven banked registers mapped to R8-14 (R8_fiq-R14_fiq). In ARM state, many FIQ handlers do not need to save any registers. User, IRQ, Supervisor, Abort and Undefined each have two banked registers mapped to R13 and R14, allowing each of these modes to have a private stack pointer and link registers.

ARM State General Registers and Program Counter

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	 R8_fiq	R8	R8	R8	R8
R9	 R9_fiq	R9	R9	R9	R9
R10	 R10_fiq	R10	R10	R10	R10
R11	 R11_fiq	R11	R11	R11	R11
R12	 R12_fiq	R12	R12	R12	R12
R13	 R13_fiq	 R13_svc	 R13_abt	 R13_irq	 R13_und
R14	 R14_fiq	 R14_svc	 R14_abt	 R14_irq	 R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

ARM State Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	 SPSR_fiq	 SPSR_svc	 SPSR_abt	 SPSR_irq	 SPSR_und

 = banked register

Figure 2-3. Register Organization in ARM State

The THUMB State Register Set

The THUMB state register set is a subset of the ARM state set. The programmer has direct access to eight general registers, R0-R7, as well as the Program Counter (PC), a stack pointer register (SP), a link register (LR), and the CPSR. There are banked Stack Pointers, Link Registers and Saved Process Status Registers (SPSRs) for each privileged mode. This is shown in Figure 2-4.

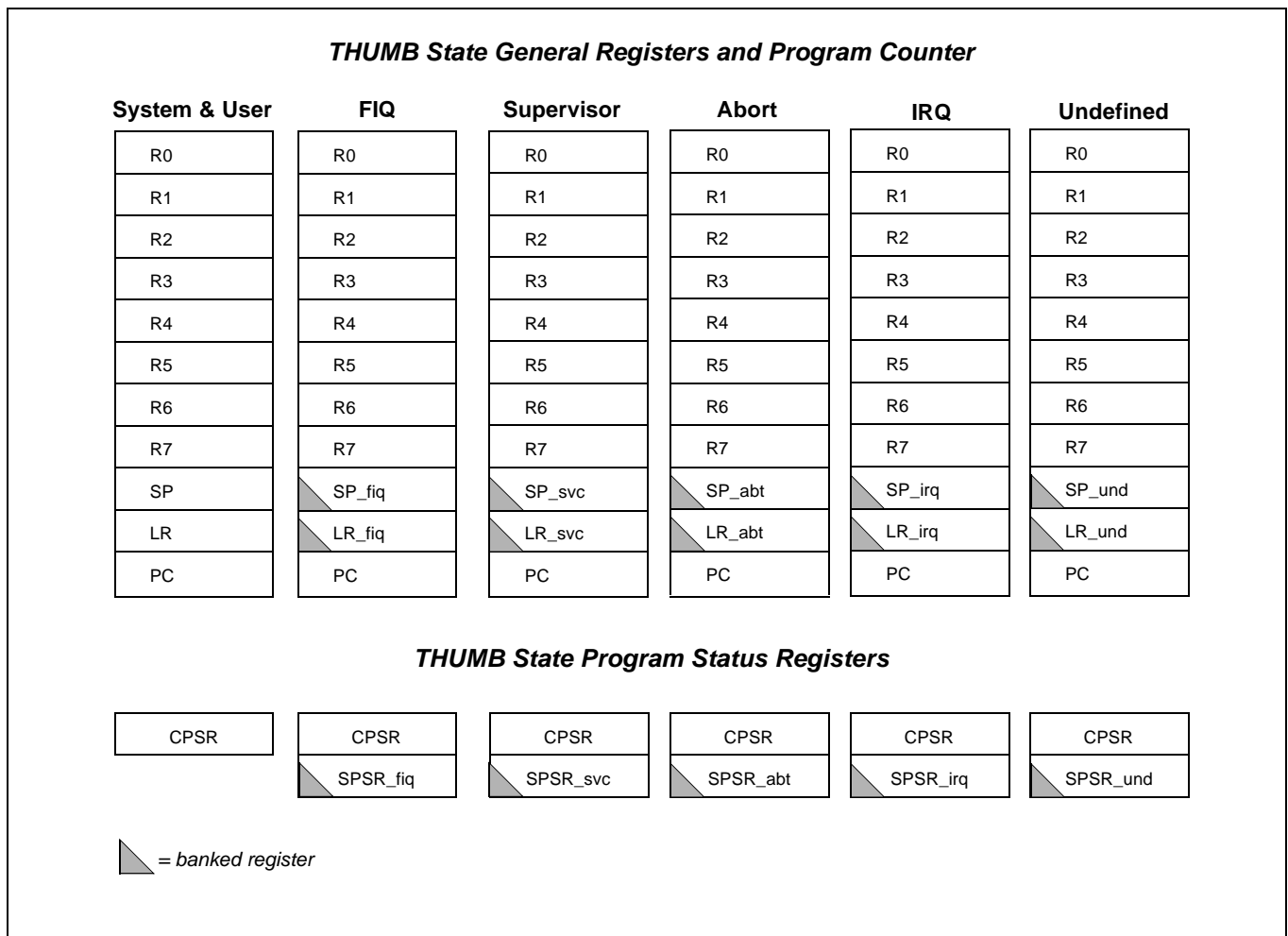


Figure 2-4. Register Organization in THUMB State

The relationship between ARM and THUMB state registers

The THUMB state registers relate to the ARM state registers in the following way:

- THUMB state R0-R7 and ARM state R0-R7 are identical
- THUMB state CPSR and SPSRs and ARM state CPSR and SPSRs are identical
- THUMB state SP maps onto ARM state R13
- THUMB state LR maps onto ARM state R14
- The THUMB state Program Counter maps onto the ARM state Program Counter (R15)

This relationship is shown in Figure 2-5.

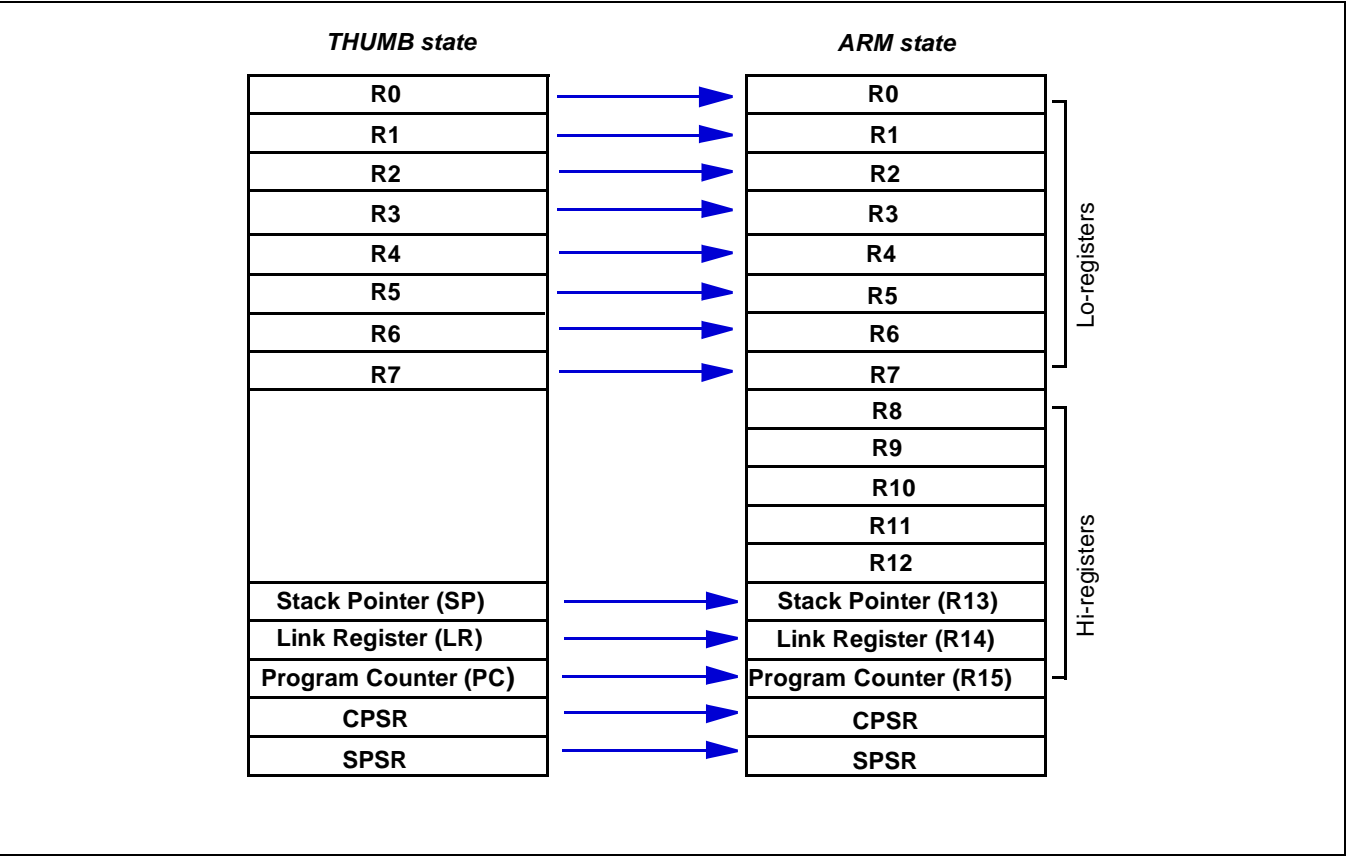


Figure 2-5. Mapping of THUMB State Registers onto ARM State Registers

Accessing Hi-Registers in THUMB State

In THUMB state, registers R8-R15 (the *Hi registers*) are not part of the standard register set. However, the assembly language programmer has limited access to them, and can use them for fast temporary storage.

A value may be transferred from a register in the range R0-R7 (a *Lo register*) to a Hi register, and from a Hi register to a Lo register, using special variants of the MOV instruction. Hi register values can also be compared against or added to Lo register values with the CMP and ADD instructions. For more information, refer to Figure 3-34.

THE PROGRAM STATUS REGISTERS

The ARM7TDMI contains a Current Program Status Register (CPSR), plus five Saved Program Status Registers (SPSRs) for use by exception handlers. These register's functions are:

- Hold information about the most recently performed ALU operation
- Control the enabling and disabling of interrupts
- Set the processor operating mode

The arrangement of bits is shown in Figure 2-6.

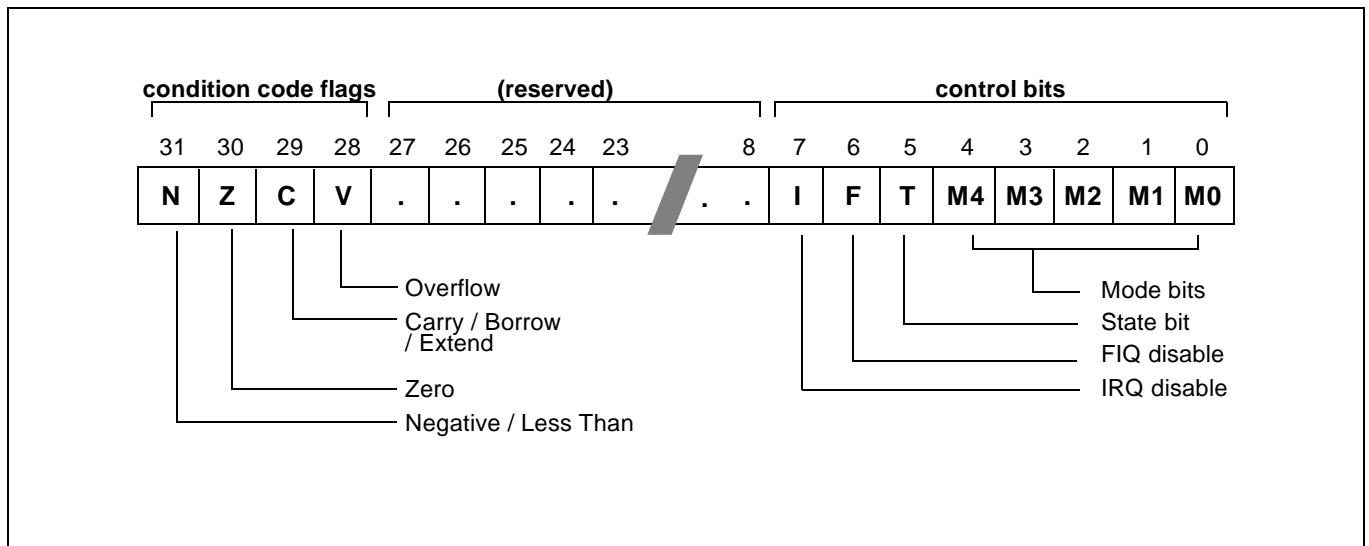


Figure 2-6 . Program Status Register Format

The Condition Code Flags

The N, Z, C and V bits are the condition code flags. These may be changed as a result of arithmetic and logical operations, and may be tested to determine whether an instruction should be executed.

In ARM state, all instructions may be executed conditionally: see Table 3-2 for details.

In THUMB state, only the Branch instruction is capable of conditional execution: see Figure 3-46 for details.

The Control Bits

The bottom 8 bits of a PSR (incorporating I, F, T and M[4:0]) are known collectively as the control bits. These will change when an exception arises. If the processor is operating in a privileged mode, they can also be manipulated by software.

<i>The T bit</i>	<p>This reflects the operating state. When this bit is set, the processor is executing in THUMB state, otherwise it is executing in ARM state. This is reflected on the TBIT external signal.</p> <p>Note that the software must never change the state of the TBIT in the CPSR. If this happens, the processor will enter an unpredictable state.</p>
<i>Interrupt disable bits</i>	<p>The I and F bits are the interrupt disable bits. When set, these disable the IRQ and FIQ interrupts respectively.</p>
<i>The mode bits</i>	<p>The M4, M3, M2, M1 and M0 bits (M[4:0]) are the mode bits. These determine the processor's operating mode, as shown in Table 2-1. Not all combinations of the mode bits define a valid processor mode. Only those explicitly described shall be used. The user should be aware that if any illegal value is programmed into the mode bits, M[4:0], then the processor will enter an unrecoverable state. If this occurs, reset should be applied.</p>

Table 2-1. PSR Mode Bit Values

M[4:0]	Mode	Visible THUMB state registers	Visible ARM state registers
10000	User	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR
10001	FIQ	R7..R0, LR_fiq, SP_fiq PC, CPSR, SPSR_fiq	R7..R0, R14_fiq..R8_fiq, PC, CPSR, SPSR_fiq
10010	IRQ	R7..R0, LR_irq, SP_irq PC, CPSR, SPSR_irq	R12..R0, R14_irq..R13_irq, PC, CPSR, SPSR_irq
10011	Supervisor	R7..R0, LR_svc, SP_svc, PC, CPSR, SPSR_svc	R12..R0, R14_svc..R13_svc, PC, CPSR, SPSR_svc
10111	Abort	R7..R0, LR_abt, SP_abt, PC, CPSR, SPSR_abt	R12..R0, R14_abt..R13_abt, PC, CPSR, SPSR_abt
11011	Undefined	R7..R0 LR_und, SP_und, PC, CPSR, SPSR_und	R12..R0, R14_und..R13_und, PC, CPSR
11111	System	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR

Reserved bits

The remaining bits in the PSRs are reserved. When changing a PSR's flag or control bits, you must ensure that these unused bits are not altered. Also, your program should not rely on them containing specific values, since in future processors they may read as one or zero.

EXCEPTIONS

Exceptions arise whenever the normal flow of a program has to be halted temporarily, for example to service an interrupt from a peripheral. Before an exception can be handled, the current processor state must be preserved so that the original program can resume when the handler routine has finished.

It is possible for several exceptions to arise at the same time. If this happens, they are dealt with in a fixed order. See Exception Priorities on page 2-14.

Action on Entering an Exception

When handling an exception, the ARM7TDMI:

1. Preserves the address of the next instruction in the appropriate Link Register. If the exception has been entered from ARM state, then the address of the next instruction is copied into the Link Register (that is, current PC + 4 or PC + 8 depending on the exception. See Table 2-2 on for details). If the exception has been entered from THUMB state, then the value written into the Link Register is the current PC offset by a value such that the program resumes from the correct place on return from the exception. This means that the exception handler need not determine which state the exception was entered from. For example, in the case of SWI, MOVS PC, R14_svc will always return to the next instruction regardless of whether the SWI was executed in ARM or THUMB state.
2. Copies the CPSR into the appropriate SPSR
3. Forces the CPSR mode bits to a value which depends on the exception
4. Forces the PC to fetch the next instruction from the relevant exception vector

It may also set the interrupt disable flags to prevent otherwise unmanageable nestings of exceptions.

If the processor is in THUMB state when an exception occurs, it will automatically switch into ARM state when the PC is loaded with the exception vector address.

Action on Leaving an Exception

On completion, the exception handler:

1. Moves the Link Register, minus an offset where appropriate, to the PC. (The offset will vary depending on the type of exception.)
2. Copies the SPSR back to the CPSR
3. Clears the interrupt disable flags, if they were set on entry

NOTE

An explicit switch back to THUMB state is never needed, since restoring the CPSR from the SPSR automatically sets the T bit to the value it held immediately prior to the exception.

Exception Entry/Exit Summary

Table 2-2 summarises the PC value preserved in the relevant R14 on exception entry, and the recommended instruction for exiting the exception handler.

Table 2-2. Exception Entry/Exit

	Return Instruction	Previous State		Notes
		ARM R14_x	THUMB R14_x	
BL	MOV PC, R14	PC + 4	PC + 2	1
SWI	MOVS PC, R14_svc	PC + 4	PC + 2	1
UDEF	MOVS PC, R14_und	PC + 4	PC + 2	1
FIQ	SUBS PC, R14_fiq, #4	PC + 4	PC + 4	2
IRQ	SUBS PC, R14_irq, #4	PC + 4	PC + 4	2
PABT	SUBS PC, R14_abt, #4	PC + 4	PC + 4	1
DABT	SUBS PC, R14_abt, #8	PC + 8	PC + 8	3
RESET	NA	—	—	4

NOTES

1. Where PC is the address of the BL/SWI/Undefined Instruction fetch which had the prefetch abort.
2. Where PC is the address of the instruction which did not get executed since the FIQ or IRQ took priority.
3. Where PC is the address of the Load or Store instruction which generated the data abort.
4. The value saved in R14_svc upon reset is unpredictable.

FIQ

The FIQ (Fast Interrupt Request) exception is designed to support a data transfer or channel process, and in ARM state has sufficient private registers to remove the need for register saving (thus minimising the overhead of context switching).

FIQ is externally generated by taking the **nFIQ** input LOW. This input can except either synchronous or asynchronous transitions, depending on the state of the **ISYNC** input signal. When **ISYNC** is LOW, **nFIQ** and **nIRQ** are considered asynchronous, and a cycle delay for synchronization is incurred before the interrupt can affect the processor flow.

Irrespective of whether the exception was entered from ARM or Thumb state, a FIQ handler should leave the interrupt by executing

SUBS PC,R14_fiq,#4

FIQ may be disabled by setting the CPSR's F flag (but note that this is not possible from User mode). If the F flag is clear, ARM7TDMI checks for a LOW level on the output of the FIQ synchroniser at the end of each instruction.

IRQ

The IRQ (Interrupt Request) exception is a normal interrupt caused by a LOW level on the **nIRQ** input. IRQ has a lower priority than FIQ and is masked out when a FIQ sequence is entered. It may be disabled at any time by setting the I bit in the CPSR, though this can only be done from a privileged (non-User) mode.

Irrespective of whether the exception was entered from ARM or Thumb state, an IRQ handler should return from the interrupt by executing

```
SUBS PC,R14_irq,#4
```

Abort

An abort indicates that the current memory access cannot be completed. It can be signalled by the external **ABORT** input. ARM7TDMI checks for the abort exception during memory access cycles.

There are two types of abort:

- *Prefetch abort*: occurs during an instruction prefetch.
- *Data abort*: occurs during a data access.

If a prefetch abort occurs, the prefetched instruction is marked as invalid, but the exception will not be taken until the instruction reaches the head of the pipeline. If the instruction is not executed - for example because a branch occurs while it is in the pipeline - the abort does not take place.

If a data abort occurs, the action taken depends on the instruction type:

- Single data transfer instructions (LDR, STR) write back modified base registers: the Abort handler must be aware of this.
- The swap instruction (SWP) is aborted as though it had not been executed.
- Block data transfer instructions (LDM, STM) complete. If write-back is set, the base is updated. If the instruction would have overwritten the base with data (ie it has the base in the transfer list), the overwriting is prevented. All register overwriting is prevented after an abort is indicated, which means in particular that R15 (always the last register to be transferred) is preserved in an aborted LDM instruction.

The abort mechanism allows the implementation of a demand paged virtual memory system. In such a system the processor is allowed to generate arbitrary addresses. When the data at an address is unavailable, the Memory Management Unit (MMU) signals an abort. The abort handler must then work out the cause of the abort, make the requested data available, and retry the aborted instruction. The application program needs no knowledge of the amount of memory available to it, nor is its state in any way affected by the abort.

After fixing the reason for the abort, the handler should execute the following irrespective of the state (ARM or Thumb):

```
SUBS PC,R14_abt,#4    ; for a prefetch abort, or
SUBS PC,R14_abt,#8    ; for a data abort
```

This restores both the PC and the CPSR, and retries the aborted instruction.

Software Interrupt

The software interrupt instruction (SWI) is used for entering Supervisor mode, usually to request a particular supervisor function. A SWI handler should return by executing the following irrespective of the state (ARM or Thumb):

```
MOV PC,R14_svc
```

This restores the PC and CPSR, and returns to the instruction following the SWI.

NOTE

nFIQ, nIRQ, ISYNC, LOCK, BIGEND, and ABORT pins exist only in the ARM7TDMI CPU core.

Undefined Instruction

When ARM7TDMI comes across an instruction which it cannot handle, it takes the undefined instruction trap. This mechanism may be used to extend either the THUMB or ARM instruction set by software emulation.

After emulating the failed instruction, the trap handler should execute the following irrespective of the state (ARM or Thumb):

```
MOVS PC,R14_und
```

This restores the CPSR and returns to the instruction following the undefined instruction.

Exception Vectors

The following table shows the exception vector addresses.

Table 2-3. Exception Vectors

Address	Exception	Mode on entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	Reserved	Reserved
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

Exception Priorities

When multiple exceptions arise at the same time, a fixed priority system determines the order in which they are handled:

Highest priority:

1. Reset
2. Data abort
3. FIQ
4. IRQ
5. Prefetch abort

Lowest priority:

6. Undefined Instruction, Software interrupt.

Not All Exceptions Can Occur at Once:

Undefined Instruction and Software Interrupt are mutually exclusive, since they each correspond to particular (non-overlapping) decodings of the current instruction.

If a data abort occurs at the same time as a FIQ, and FIQs are enabled (ie the CPSR's F flag is clear), ARM7TDMI enters the data abort handler and then immediately proceeds to the FIQ vector. A normal return from FIQ will cause the data abort handler to resume execution. Placing data abort at a higher priority than FIQ is necessary to ensure that the transfer error does not escape detection. The time for this exception entry should be added to worst-case FIQ latency calculations.

INTERRUPT LATENCIES

The worst case latency for FIQ, assuming that it is enabled, consists of the longest time the request can take to pass through the synchroniser ($T_{syncmax}$ if asynchronous), plus the time for the longest instruction to complete (T_{ldm} , the longest instruction is an LDM which loads all the registers including the PC), plus the time for the data abort entry (T_{exc}), plus the time for FIQ entry (T_{fiq}). At the end of this time ARM7TDMI will be executing the instruction at 0x1C.

$T_{syncmax}$ is 3 processor cycles, T_{ldm} is 20 cycles, T_{exc} is 3 cycles, and T_{fiq} is 2 cycles. The total time is therefore 28 processor cycles. This is just over 1.4 microseconds in a system which uses a continuous 20 MHz processor clock. The maximum IRQ latency calculation is similar, but must allow for the fact that FIQ has higher priority and could delay entry into the IRQ handling routine for an arbitrary length of time. The minimum latency for FIQ or IRQ consists of the shortest time the request can take through the synchroniser ($T_{syncmin}$) plus T_{fiq} . This is 4 processor cycles.

RESET

When the **nRESET** signal goes LOW, ARM7TDMI abandons the executing instruction and then continues to fetch instructions from incrementing word addresses.

When **nRESET** goes HIGH again, ARM7TDMI:

1. Overwrites R14_svc and SPSR_svc by copying the current values of the PC and CPSR into them. The value of the saved PC and SPSR is not defined.
2. Forces M[4:0] to 10011 (Supervisor mode), sets the I and F bits in the CPSR, and clears the CPSR's T bit.
3. Forces the PC to fetch the next instruction from address 0x00.
4. Execution resumes in ARM state.

3

Instruction set

INSTRUCTION SET SUMMAY

This chapter describes the ARM instruction set and the THUMB instruction set in the ARM7TDMI core.

FORMAT SUMMARY

The ARM instruction set formats are shown below.

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	
Cond	0	0	I	Opcode				S	Rn				Rd				Operand 2												<i>Data Processing / PSR Transfer</i>			
Cond	0	0	0	0	0	0	A	S	Rd				Rn				Rs				1	0	0	1	Rm				<i>Multiply</i>			
Cond	0	0	0	0	1	U	A	S	RdHi				RdLo				Rn				1	0	0	1	Rm				<i>Multiply Long</i>			
Cond	0	0	0	1	0	B	0	0	Rn				Rd				0	0	0	0	1	0	0	1	Rm				<i>Single Data Swap</i>			
Cond	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	Rn				<i>Branch and Exchange</i>		
Cond	0	0	0	P	U	0	W	L	Rn				Rd				0	0	0	0	1	S	H	1	Rm				<i>Halfword Data Transfer: register offset</i>			
Cond	0	0	0	P	U	1	W	L	Rn				Rd				Offset				1	S	H	1	Offset				<i>Halfword Data Transfer: immediate offset</i>			
Cond	0	1	I	P	U	B	W	L	Rn				Rd				Offset												<i>Single Data Transfer</i>			
Cond	0	1	1																									1			<i>Undefined</i>	
Cond	1	0	0	P	U	S	W	L	Rn				Register List																<i>Block Data Transfer</i>			
Cond	1	0	1	L	Offset																								<i>Branch</i>			
Cond	1	1	0	P	U	N	W	L	Rn				CRd				CP#				Offset								<i>Coprocessor Data Transfer</i>			
Cond	1	1	1	0	CP Opc				CRn				CRd				CP#				CP				0	CRm				<i>Coprocessor Data Operation</i>		
Cond	1	1	1	0	CP Opc				L	CRn				Rd				CP#				CP				1	CRm				<i>Coprocessor Register Transfer</i>	
Cond	1	1	1	1	Ignored by processor																								<i>Software Interrupt</i>			
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	

Figure 3-1. ARM Instruction Set Format

NOTE

Some instruction codes are not defined but do not cause the Undefined instruction trap to be taken, for instance a Multiply instruction with bit 6 changed to a 1. These instructions should not be used, as their action may change in future ARM implementations.

INSTRUCTION SUMMARY

Table 3-1. The ARM Instruction Set

Mnemonic	Instruction	Action
ADC	Add with carry	$Rd = Rn + Op2 + \text{Carry}$
ADD	Add	$Rd = Rn + Op2$
AND	AND	$Rd = Rn \text{ AND } Op2$
B	Branch	$R15 = \text{address}$
BIC	Bit Clear	$Rd = Rn \text{ AND NOT } Op2$
BL	Branch with Link	$R14 = R15, R15 = \text{address}$
BX	Branch and Exchange	$R15 = Rn,$ $T \text{ bit} = Rn[0]$
CDP	Coprocessor Data Processing	(Coprocessor-specific)
CMN	Compare Negative	CPSR flags: $= Rn + Op2$
CMP	Compare	CPSR flags: $= Rn - Op2$
EOR	Exclusive OR	$Rd = (Rn \text{ AND NOT } Op2)$ $\text{OR } (Op2 \text{ AND NOT } Rn)$
LDC	Load coprocessor from memory	Coprocessor load
LDM	Load multiple registers	Stack manipulation (Pop)
LDR	Load register from memory	$Rd = (\text{address})$
MCR	Move CPU register to coprocessor register	$cRn = rRn \{<op>cRm\}$
MLA	Multiply Accumulate	$Rd = (Rm * Rs) + Rn$
MOV	Move register or constant	$Rd = Op2$
MRC	Move from coprocessor register to CPU register	$Rn = cRn \{<op>cRm\}$
MRS	Move PSR status/flags to register	$Rn = \text{PSR}$
MSR	Move register to PSR status/flags	$\text{PSR} = Rm$
MUL	Multiply	$Rd = Rm * Rs$
MVN	Move negative register	$Rd = 0xFFFFFFFF \text{ EOR } Op2$
ORR	OR	$Rd = Rn \text{ OR } Op2$
RSB	Reverse Subtract	$Rd = Op2 - Rn$
RSC	Reverse Subtract with Carry	$Rd = Op2 - Rn - 1 + \text{Carry}$

Table 3-1. The ARM Instruction Set (Continued)

Mnemonic	Instruction	Action
SBC	Subtract with Carry	$Rd = Rn - Op2 - 1 + \text{Carry}$
STC	Store coprocessor register to memory	address: = CRn
STM	Store Multiple	Stack manipulation (Push)
STR	Store register to memory	<address>: = Rd
SUB	Subtract	$Rd = Rn - Op2$
SWI	Software Interrupt	OS call
SWP	Swap register with memory	$Rd = [Rn], [Rn] := Rm$
TEQ	Test bitwise equality	CPSR flags: = Rn EOR Op2
TST	Test bits	CPSR flags: = Rn AND Op2

THE CONDITION FIELD

In ARM state, all instructions are conditionally executed according to the state of the CPSR condition codes and the instruction's condition field. This field (bits 31:28) determines the circumstances under which an instruction is to be executed. If the state of the C, N, Z and V flags fulfils the conditions encoded by the field, the instruction is executed, otherwise it is ignored.

There are sixteen possible conditions, each represented by a two-character suffix that can be appended to the instruction's mnemonic. For example, a Branch (B in assembly language) becomes BEQ for "Branch if Equal", which means the Branch will only be taken if the Z flag is set.

In practice, fifteen different conditions may be used: these are listed in Table 3-2. The sixteenth (1111) is reserved, and must not be used.

In the absence of a suffix, the condition field of most instructions is set to "Always" (suffix AL). This means the instruction will always be executed regardless of the CPSR condition codes.

Table 3-2. Condition Code Summary

Code	Suffix	Flags	Meaning
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	negative
0101	PL	N clear	positive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always

BRANCH AND EXCHANGE (BX)

This instruction is only executed if the condition is true. The various conditions are defined in Table 3-2.

This instruction performs a branch by copying the contents of a general register, Rn, into the program counter, PC. The branch causes a pipeline flush and refill from the address specified by Rn. This instruction also permits the instruction set to be exchanged. When the instruction is executed, the value of Rn[0] determines whether the instruction stream will be decoded as ARM or THUMB instructions.

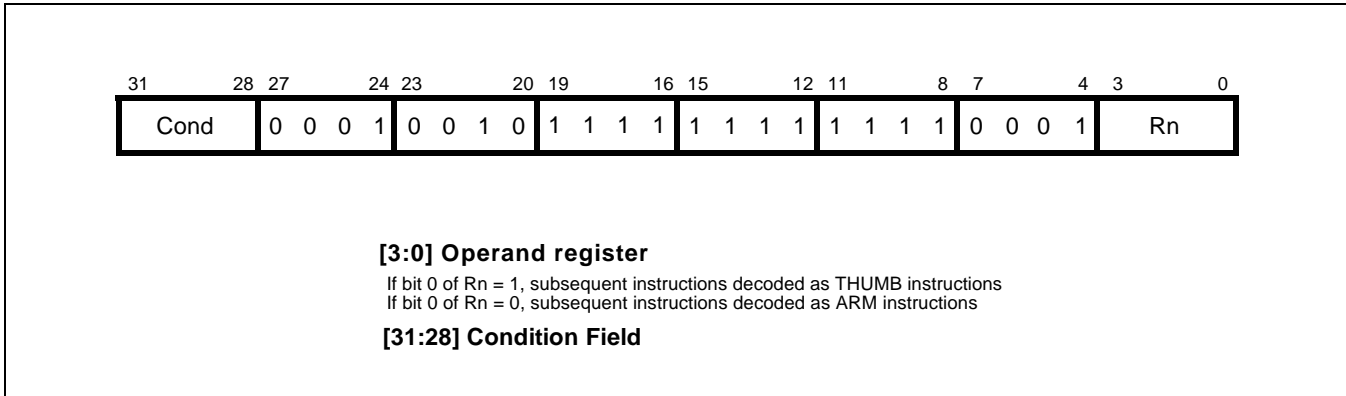


Figure 3-2. Branch and Exchange Instructions

INSTRUCTION CYCLE TIMES

The BX instruction takes 2S + 1N cycles to execute, where S and N are defined as sequential (S-cycle) and non-sequential (N-cycle), respectively.

ASSEMBLER SYNTAX

BX - branch and exchange.

BX {cond} Rn

{cond} Two character condition mnemonic. See Table 3-2.

Rn is an expression evaluating to a valid register number.

USING R15 AS AN OPERAND

If R15 is used as an operand, the behaviour is undefined.

Examples

```
ADR      R0, Into_THUMB + 1      ; Generate branch target address
                                           ; and set bit 0 high - hence
                                           ; arrive in THUMB state.
BX       R0                      ; Branch and change to THUMB
                                           ; state.
CODE16                                       ; Assemble subsequent code as
Into_THUMB                                ; THUMB instructions
.
.
.
ADR R5, Back_to_ARM                  : Generate branch target to word aligned address
                                           ; - hence bit 0 is low and so change back to ARM state.
BX R5                                ; Branch and change back to ARM state.
.
.
.
ALIGN                                       ; Word align
CODE32                                   ; Assemble subsequent code as ARM instructions
Back_to_ARM
```

BRANCH AND BRANCH WITH LINK (B, BL)

The instruction is only executed if the condition is true. The various conditions are defined Table 3-2. The instruction encoding is shown in Figure 3-3, below.

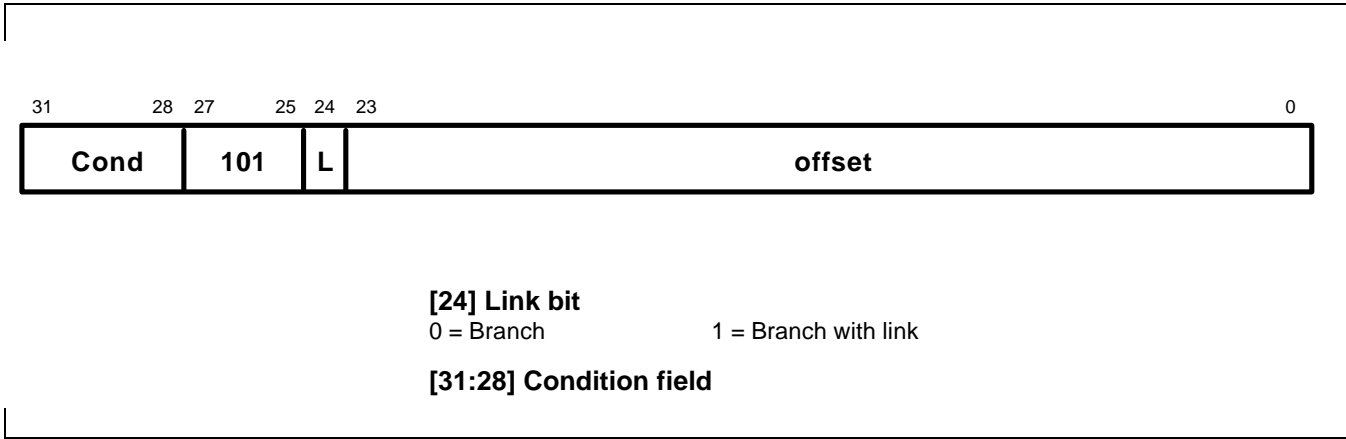


Figure 3-3. Branch Instructions

Branch instructions contain a signed 2's complement 24 bit offset. This is shifted left two bits, sign extended to 32 bits, and added to the PC. The instruction can therefore specify a branch of +/- 32Mbytes. The branch offset must take account of the prefetch operation, which causes the PC to be 2 words (8 bytes) ahead of the current instruction.

Branches beyond +/- 32Mbytes must use an offset or absolute destination which has been previously loaded into a register. In this case the PC should be manually saved in R14 if a Branch with Link type operation is required.

THE LINK BIT

Branch with Link (BL) writes the old PC into the link register (R14) of the current bank. The PC value written into R14 is adjusted to allow for the prefetch, and contains the address of the instruction following the branch and link instruction. Note that the CPSR is not saved with the PC and R14[1:0] are always cleared.

To return from a routine called by Branch with Link use MOV PC,R14 if the link register is still valid or LDM Rn!,{..PC} if the link register has been saved onto a stack pointed to by Rn.

INSTRUCTION CYCLE TIMES

Branch and Branch with Link instructions take 2S + 1N incremental cycles, where S and N are defined as sequential (S-cycle) and internal (I-cycle).

ASSEMBLER SYNTAX

Items in {} are optional. Items in <> must be present.

B{L}{cond} <expression>

{L} Used to request the Branch with Link form of the instruction. If absent, R14 will not be affected by the instruction.

{cond} A two-character mnemonic as shown in Table 3-2. If absent then AL (ALways) will be used.

<expression> The destination. The assembler calculates the offset.

EXAMPLES

here	BAL	here	; Assembles to 0xEAFFFFFEE (note effect of PC offset).
	B	there	; Always condition used as default.
	CMP	R1,#0	; Compare R1 with zero and branch to fred
			; if R1 was zero, otherwise continue.
	BEQ	fred	; Continue to next instruction.
	BL	sub+ROM	; Call subroutine at computed address.
	ADDS	R1,#1	; Add 1 to register 1, setting CPSR flags
			; on the result then call subroutine if
	BLCC	sub	; the C flag is clear, which will be the
			; case unless R1 held 0xFFFFFFFF.

DATA PROCESSING

The data processing instruction is only executed if the condition is true. The conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-4.

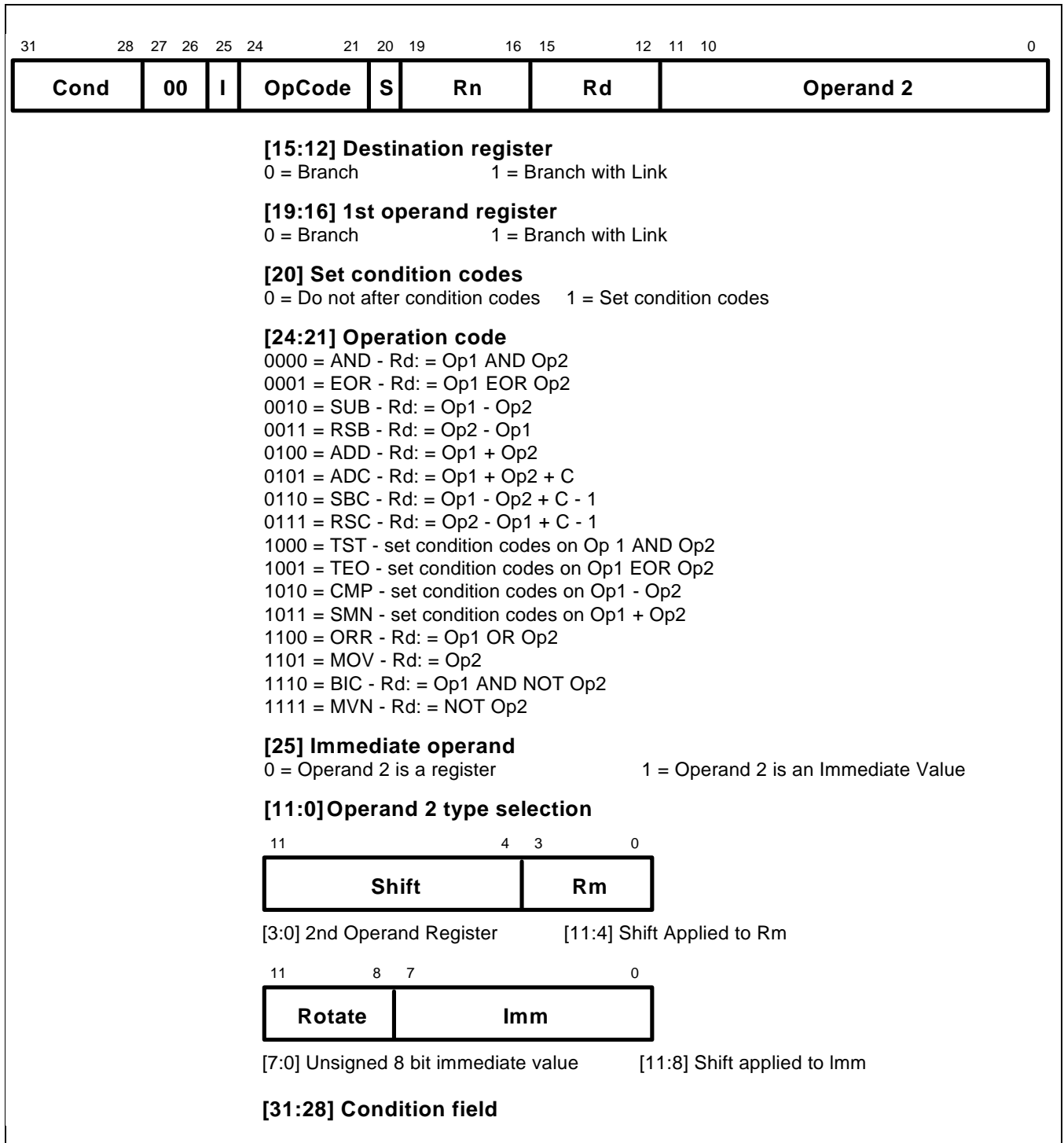


Figure 3-4. Data Processing Instructions

The instruction produces a result by performing a specified arithmetic or logical operation on one or two operands. The first operand is always a register (Rn).

The second operand may be a shifted register (Rm) or a rotated 8 bit immediate value (Imm) according to the value of the I bit in the instruction. The condition codes in the CPSR may be preserved or updated as a result of this instruction, according to the value of the S bit in the instruction.

Certain operations (TST, TEQ, CMP, CMN) do not write the result to Rd. They are used only to perform tests and to set the condition codes on the result and always have the S bit set. The instructions and their effects are listed in Table 3-3.

CPSR FLAGS

The data processing operations may be classified as logical or arithmetic. The logical operations (AND, EOR, TST, TEQ, ORR, MOV, BIC, MVN) perform the logical action on all corresponding bits of the operand or operands to produce the result. If the S bit is set (and Rd is not R15, see below) the V flag in the CPSR will be unaffected, the C flag will be set to the carry out from the barrel shifter (or preserved when the shift operation is LSL #0), the Z flag will be set if and only if the result is all zeros, and the N flag will be set to the logical value of bit 31 of the result.

Table 3-3. ARM Data Processing Instructions

Assembler Mnemonic	Op-Code	Action
AND	0000	Operand1 AND operand2
EOR	0001	Operand1 EOR operand2
SUB	0010	Operand1 – operand2
RSB	0011	Operand2 operand1
ADD	0100	Operand1 + operand2
ADC	0101	Operand1 + operand2 + carry
SBC	0110	Operand1 – operand2 + carry – 1
RSC	0111	Operand2 – operand1 + carry – 1
TST	1000	As AND, but result is not written
TEQ	1001	As EOR, but result is not written
CMP	1010	As SUB, but result is not written
CMN	1011	As ADD, but result is not written
ORR	1100	Operand1 OR operand2
MOV	1101	Operand2 (operand1 is ignored)
BIC	1110	Operand1 AND NOT operand2 (Bit clear)
MVN	1111	NOT operand2 (operand1 is ignored)

The arithmetic operations (SUB, RSB, ADD, ADC, SBC, RSC, CMP, CMN) treat each operand as a 32 bit integer (either unsigned or 2's complement signed, the two are equivalent). If the S bit is set (and Rd is not R15) the V flag in the CPSR will be set if an overflow occurs into bit 31 of the result; this may be ignored if the operands were considered unsigned, but warns of a possible error if the operands were 2's complement signed. The C flag will be set to the carry out of bit 31 of the ALU, the Z flag will be set if and only if the result was zero, and the N flag will be set to the value of bit 31 of the result (indicating a negative result if the operands are considered to be 2's complement signed).

SHIFTS

When the second operand is specified to be a shifted register, the operation of the barrel shifter is controlled by the Shift field in the instruction. This field indicates the type of shift to be performed (logical left or right, arithmetic right or rotate right). The amount by which the register should be shifted may be contained in an immediate field in the instruction, or in the bottom byte of another register (other than R15). The encoding for the different shift types is shown in Figure 3-5.

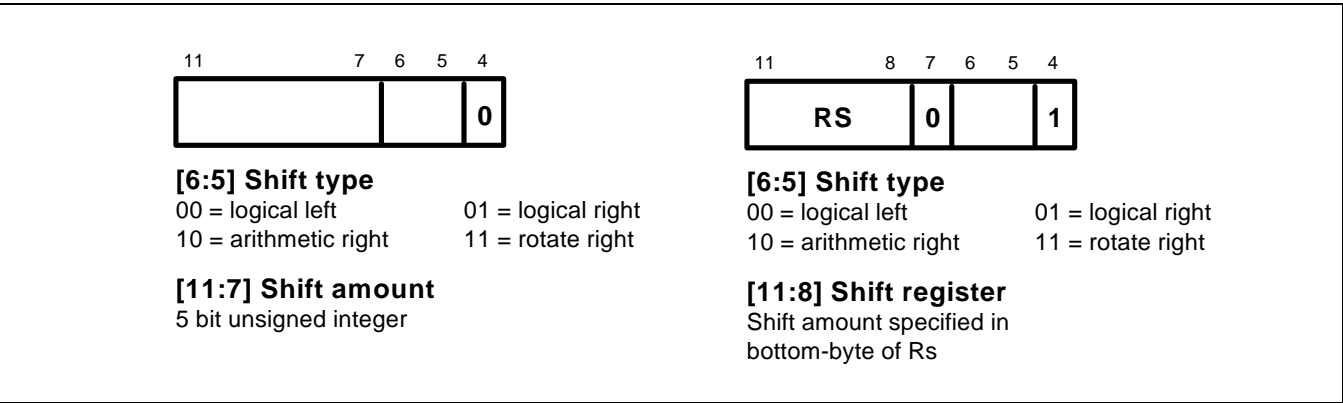


Figure 3-5. ARM Shift Operations

Instruction specified shift amount

When the shift amount is specified in the instruction, it is contained in a 5 bit field which may take any value from 0 to 31. A logical shift left (LSL) takes the contents of Rm and moves each bit by the specified amount to a more significant position. The least significant bits of the result are filled with zeros, and the high bits of Rm which do not map into the result are discarded, except that the least significant discarded bit becomes the shifter carry output which may be latched into the C bit of the CPSR when the ALU operation is in the logical class (see above). For example, the effect of LSL #5 is shown in Figure 3-6.

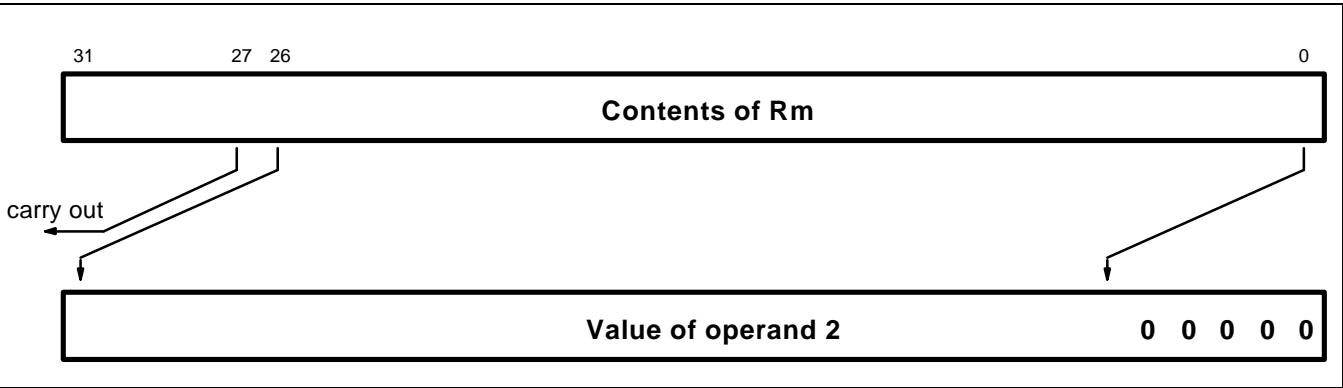


Figure 3-6. Logical Shift Left

NOTE

LSL #0 is a special case, where the shifter carry out is the old value of the CPSR C flag. The contents of Rm are used directly as the second operand. A logical shift right (LSR) is similar, but the contents of Rm are moved to less significant positions in the result. LSR #5 has the effect shown in Figure 3-7.

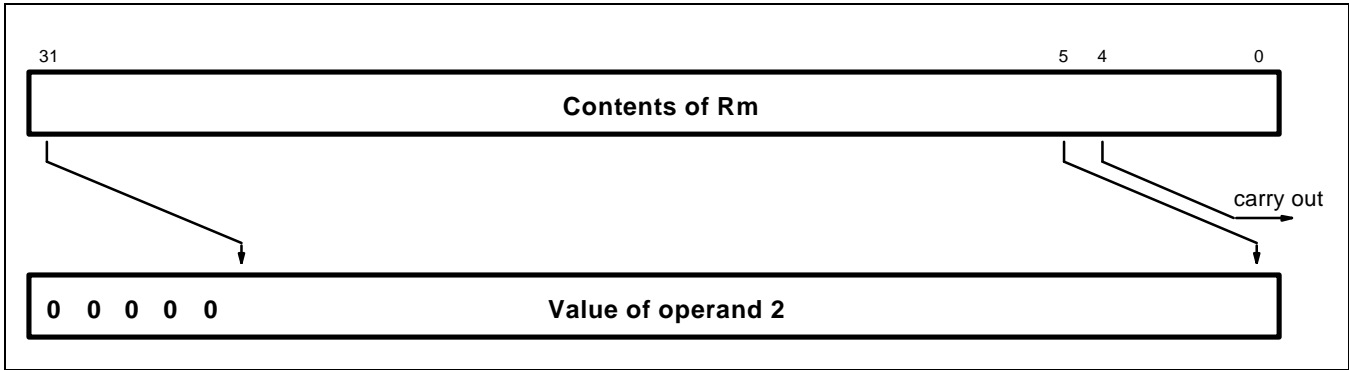


Figure 3-7. Logical Shift Right

The form of the shift field which might be expected to correspond to LSR #0 is used to encode LSR #32, which has a zero result with bit 31 of Rm as the carry output. Logical shift right zero is redundant as it is the same as logical shift left zero, so the assembler will convert LSR #0 (and ASR #0 and ROR #0) into LSL #0, and allow LSR #32 to be specified.

An arithmetic shift right (ASR) is similar to logical shift right, except that the high bits are filled with bit 31 of Rm instead of zeros. This preserves the sign in 2's complement notation. For example, ASR #5 is shown in Figure 3-8.

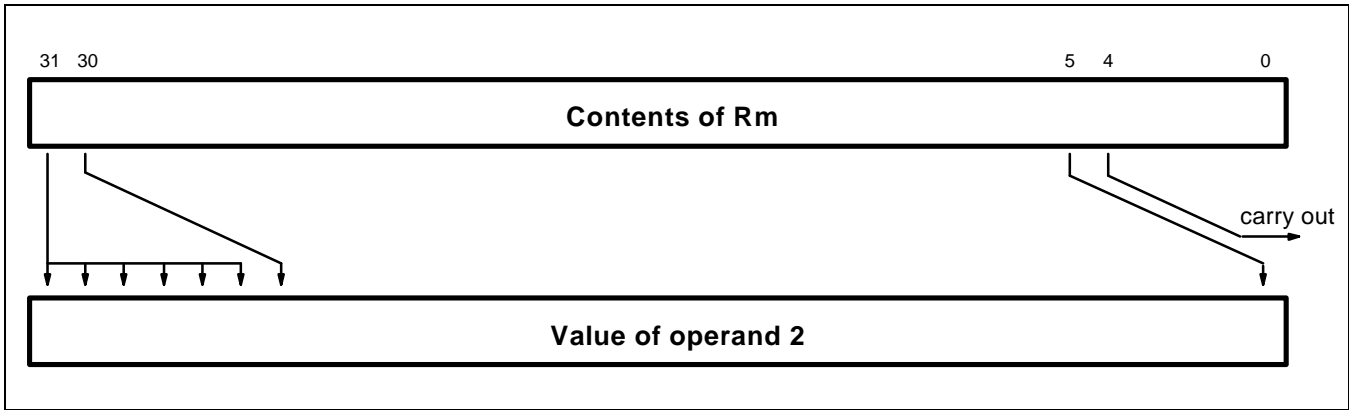


Figure 3-8. Arithmetic Shift Right

The form of the shift field which might be expected to give ASR #0 is used to encode ASR #32. Bit 31 of Rm is again used as the carry output, and each bit of operand 2 is also equal to bit 31 of Rm. The result is therefore all ones or all zeros, according to the value of bit 31 of Rm.

Rotate right (ROR) operations reuse the bits which “overshoot” in a logical shift right operation by reintroducing them at the high end of the result, in place of the zeros used to fill the high end in logical right operations. For example, ROR #5 is shown in Figure 3-9. The form of the shift field which might be expected to give ROR #0 is

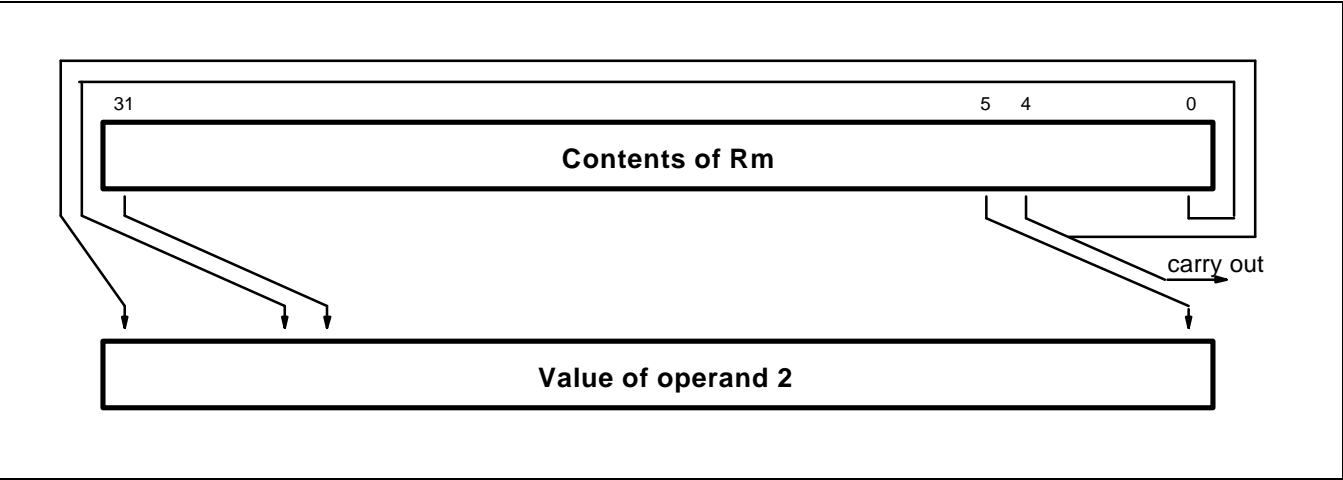


Figure 3-9. Rotate Right

used to encode a special function of the barrel shifter, rotate right extended (RRX). This is a rotate right by one bit position of the 33 bit quantity formed by appending the CPSR C flag to the most significant end of the contents of Rm as shown in Figure 3-10.

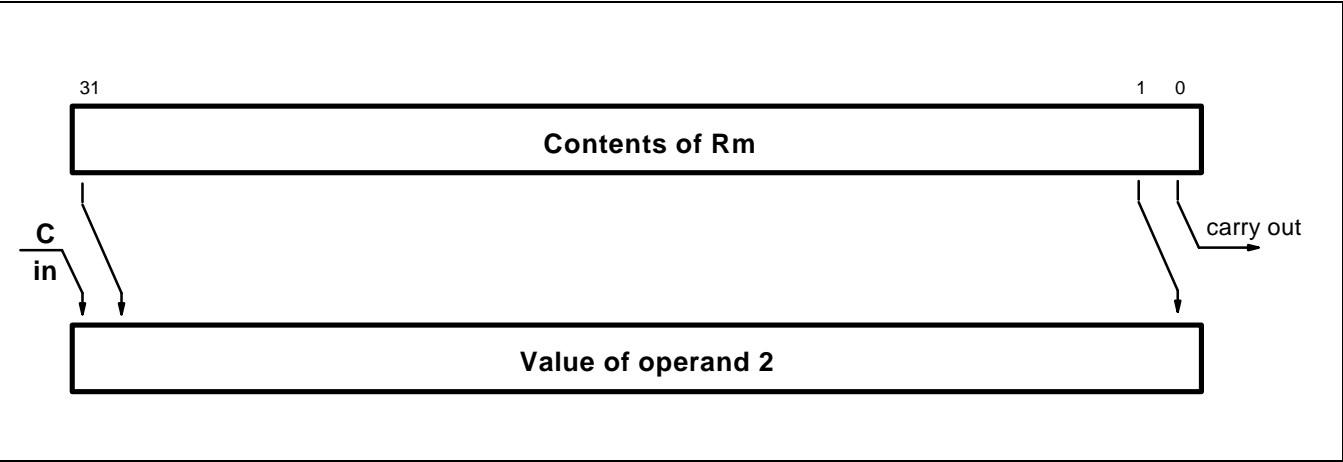


Figure 3-10. Rotate Right Extended

Register specified shift amount

Only the least significant byte of the contents of Rs is used to determine the shift amount. Rs can be any general register other than R15.

If this byte is zero, the unchanged contents of Rm will be used as the second operand, and the old value of the CPSR C flag will be passed on as the shifter carry output.

If the byte has a value between 1 and 31, the shifted result will exactly match that of an instruction specified shift with the same value and shift operation.

If the value in the byte is 32 or more, the result will be a logical extension of the shift described above:

- 1 LSL by 32 has result zero, carry out equal to bit 0 of Rm.
- 2 LSL by more than 32 has result zero, carry out zero.
- 3 LSR by 32 has result zero, carry out equal to bit 31 of Rm.
- 4 LSR by more than 32 has result zero, carry out zero.
- 5 ASR by 32 or more has result filled with and carry out equal to bit 31 of Rm.
- 6 ROR by 32 has result equal to Rm, carry out equal to bit 31 of Rm.
- 7 ROR by n where n is greater than 32 will give the same result and carry out as ROR by n-32; therefore repeatedly subtract 32 from n until the amount is in the range 1 to 32 and see above.

NOTE

The zero in bit 7 of an instruction with a register controlled shift is compulsory; a one in this bit will cause the instruction to be a multiply or undefined instruction.

IMMEDIATE OPERAND ROTATES

The immediate operand rotate field is a 4 bit unsigned integer which specifies a shift operation on the 8 bit immediate value. This value is zero extended to 32 bits, and then subject to a rotate right by twice the value in the rotate field. This enables many common constants to be generated, for example all powers of 2.

WRITING TO R15

When Rd is a register other than R15, the condition code flags in the CPSR may be updated from the ALU flags as described above.

When Rd is R15 and the S flag in the instruction is not set the result of the operation is placed in R15 and the CPSR is unaffected.

When Rd is R15 and the S flag is set the result of the operation is placed in R15 and the SPSR corresponding to the current mode is moved to the CPSR. This allows state changes which atomically restore both PC and CPSR. This form of instruction should not be used in User mode.

USING R15 AS AN OPERAND

If R15 (the PC) is used as an operand in a data processing instruction the register is used directly.

The PC value will be the address of the instruction, plus 8 or 12 bytes due to instruction prefetching. If the shift amount is specified in the instruction, the PC will be 8 bytes ahead. If a register is used to specify the shift amount the PC will be 12 bytes ahead.

TEQ, TST, CMP AND CMN OPCODES

NOTE

TEQ, TST, CMP and CMN do not write the result of their operation but do set flags in the CPSR. An assembler should always set the S flag for these instructions even if this is not specified in the mnemonic.

The TEQP form of the TEQ instruction used in earlier ARM processors must not be used: the PSR transfer operations should be used instead.

The action of TEQP in the ARM7TDMI is to move SPSR_<mode> to the CPSR if the processor is in a privileged mode and to do nothing if in User mode.

INSTRUCTION CYCLE TIMES

Data Processing instructions vary in the number of incremental cycles taken as follows:

Table 3-4. Incremental Cycle Times

Processing Type	Cycles
Normal Data Processing	1S
Data Processing with register specified shift	1S + 1I
Data Processing with PC written	2S + 1N
Data Processing with register specified shift and PC written	2S + 1N + 1I

NOTE: S, N and I are as defined sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle) respectively.

ASSEMBLER SYNTAX

- MOV,MVN (single operand instructions).
<opcode>{cond}{S} Rd,<Op2>
- CMP,CMN,TEQ,TST (instructions which do not produce a result).
<opcode>{cond} Rn,<Op2>
- AND,EOR,SUB,RSB,ADD,ADC,SBC,RSC,ORR,BIC
<opcode>{cond}{S} Rd,Rn,<Op2>

where:

<Op2>	Rm{,<shift>} or,<#expression>
{cond}	A two-character condition mnemonic. See Table 3-2.
{S}	Set condition codes if S present (implied for CMP, CMN, TEQ, TST).
Rd, Rn and Rm	Expressions evaluating to a register number.
<#expression>	If this is used, the assembler will attempt to generate a shifted immediate 8-bit field to match the expression. If this is impossible, it will give an error.
<shift>	<Shiftname> <register> or <shiftname> #expression, or RRX (rotate right one bit with extend).
<shiftname>s	ASL, LSL, LSR, ASR, ROR. (ASL is a synonym for LSL, they assemble to the same code.)

EXAMPLES

ADDEQ	R2,R4,R5	; If the Z flag is set make R2:=R4+R5
TEQS	R4,#3	; Test R4 for equality with 3.
		; (The S is in fact redundant as the
		; assembler inserts it automatically.)
SUB	R4,R5,R7,LSR R2	; Logical right shift R7 by the number in
		; the bottom byte of R2, subtract result
		; from R5, and put the answer into R4.
MOV	PC,R14	; Return from subroutine.
MOVS	PC,R14	; Return from exception and restore CPSR
		; from SPSR_mode.

PSR TRANSFER (MRS, MSR)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2.

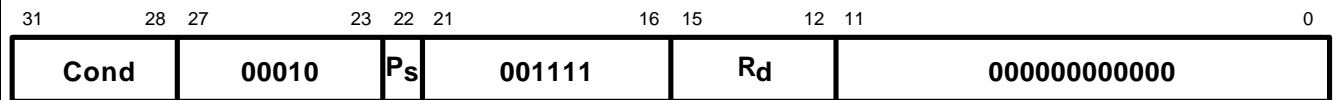
The MRS and MSR instructions are formed from a subset of the Data Processing operations and are implemented using the TEQ, TST, CMN and CMP instructions without the S flag set. The encoding is shown in Figure 3-11.

These instructions allow access to the CPSR and SPSR registers. The MRS instruction allows the contents of the CPSR or SPSR_<mode> to be moved to a general register. The MSR instruction allows the contents of a general register to be moved to the CPSR or SPSR_<mode> register.

The MSR instruction also allows an immediate value or register contents to be transferred to the condition code flags (N,Z,C and V) of CPSR or SPSR_<mode> without affecting the control bits. In this case, the top four bits of the specified register contents or 32 bit immediate value are written to the top four bits of the relevant PSR.

OPERAND RESTRICTIONS

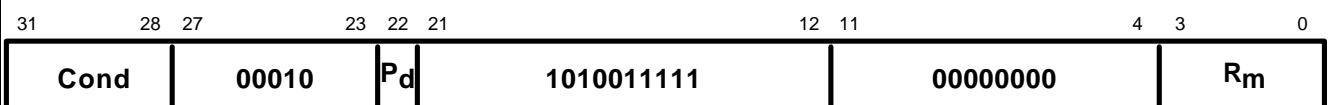
- In user mode, the control bits of the CPSR are protected from change, so only the condition code flags of the CPSR can be changed. In other (privileged) modes the entire CPSR can be changed.
- Note that the software must never change the state of the T bit in the CPSR. If this happens, the processor will enter an unpredictable state.
- The SPSR register which is accessed depends on the mode at the time of execution. For example, only SPSR_fiq is accessible when the processor is in FIQ mode.
- You must not specify R15 as the source or destination register.
- Also, do not attempt to access an SPSR in User mode, since no such register exists.

MRS (transfer PSR contents to a register)

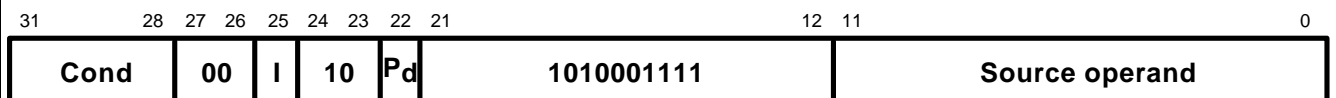
[15:12] Destination register

[22] Source PSR

0 = CPSR 1 = SPSR_<current mode>

[31:28] Condition field**MRS (transfer register contents to PSR)****[3:0] Source register****[22] Destination PSR**

0 = CPSR 1 = SPSR_<current mode>

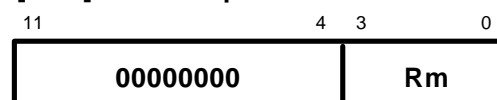
[31:28] Condition field**MRS (transfer register contents or immediate value to PSR flag bits only)****[22] Destination PSR**

0 = CPSR 1 = SPSR_<current mode>

[25] Immediate Operand

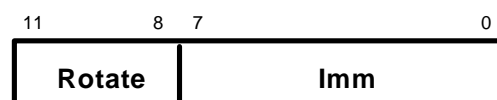
0 = Source operand is a register

1 = SPSR_<current mode>

[11:0] Source operand

[3:0] Source register

[11:4] Source operand is an immediate value



[7:0] Unsigned 8 bit immediate value

[11:8] Shift applied to Imm

[31:28] Condition field**Figure 3-11. PSR Transfer**

RESERVED BITS

Only twelve bits of the PSR are defined in ARM7TDMI (N,Z,C,V,I,F, T & M[4:0]); the remaining bits are reserved for use in future versions of the processor. Refer to Figure 2-6 for a full description of the PSR bits.

To ensure the maximum compatibility between ARM7TDMI programs and future processors, the following rules should be observed:

- The reserved bits should be preserved when changing the value in a PSR.
- Programs should not rely on specific values from the reserved bits when checking the PSR status, since they may read as one or zero in future processors.

A read-modify-write strategy should therefore be used when altering the control bits of any PSR register; this involves transferring the appropriate PSR register to a general register using the MRS instruction, changing only the relevant bits and then transferring the modified value back to the PSR register using the MSR instruction.

EXAMPLES

The following sequence performs a mode change:

MRS	R0,CPSR	; Take a copy of the CPSR.
BIC	R0,R0,#0x1F	; Clear the mode bits.
ORR	R0,R0,#new_mode	; Select new mode
MSR	CPSR,R0	; Write back the modified CPSR.

When the aim is simply to change the condition code flags in a PSR, a value can be written directly to the flag bits without disturbing the control bits. The following instruction sets the N,Z,C and V flags:

MSR	CPSR_flg,#0xF0000000	; Set all the flags egardless of their previous state
		; (does not affect any control bits).

No attempt should be made to write an 8 bit immediate value into the whole PSR since such an operation cannot preserve the reserved bits.

INSTRUCTION CYCLE TIMES

PSR transfers take 1S incremental cycles, where S is defined as Sequential (S-cycle).

ASSEMBLER SYNTAX

- MRS - transfer PSR contents to a register
MRS{cond} Rd,<psr>
- MSR - transfer register contents to PSR
MSR{cond} <psr>,Rm
- MSR - transfer register contents to PSR flag bits only
MSR{cond} <psrf>,Rm

The most significant four bits of the register contents are written to the N,Z,C & V flags respectively.

- MSR - transfer immediate value to PSR flag bits only
MSR{cond} <psrf>,<#expression>

The expression should symbolise a 32 bit value of which the most significant four bits are written to the N,Z,C and V flags respectively.

Key:

{cond}	Two-character condition mnemonic. See Table 3-2..
Rd and Rm	Expressions evaluating to a register number other than R15
<psr>	CPSR, CPSR_all, SPSR or SPSR_all. (CPSR and CPSR_all are synonyms as are SPSR and SPSR_all)
<psrf>	CPSR_flg or SPSR_flg
<#expression>	Where this is used, the assembler will attempt to generate a shifted immediate 8-bit field to match the expression. If this is impossible, it will give an error.

EXAMPLES

In User mode the instructions behave as follows:

MSR	CPSR_all,Rm	; CPSR[31:28] <- Rm[31:28]
MSR	CPSR_flg,Rm	; CPSR[31:28] <- Rm[31:28]
MSR	CPSR_flg,#0xA0000000	; CPSR[31:28] <- 0xA (set N,C; clear Z,V)
MRS	Rd,CPSR	; Rd[31:0] <- CPSR[31:0]

In privileged modes the instructions behave as follows:

MSR	CPSR_all,Rm	; CPSR[31:0] <- Rm[31:0]
MSR	CPSR_flg,Rm	; CPSR[31:28] <- Rm[31:28]
MSR	CPSR_flg,#0x50000000	; CPSR[31:28] <- 0x5 (set Z,V; clear N,C)
MSR	SPSR_all,Rm	; SPSR_<mode>[31:0] <- Rm[31:0]
MSR	SPSR_flg,Rm	; SPSR_<mode>[31:28] <- Rm[31:28]
MSR	SPSR_flg,#0xC0000000	; SPSR_<mode>[31:28] <- 0xC (set N,Z; clear C,V)
MRS	Rd,SPSR	; Rd[31:0] <- SPSR_<mode>[31:0]

MULTIPLY AND MULTIPLY-ACCUMULATE (MUL, MLA)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-12.

The multiply and multiply-accumulate instructions use an 8 bit Booth's algorithm to perform integer multiplication.

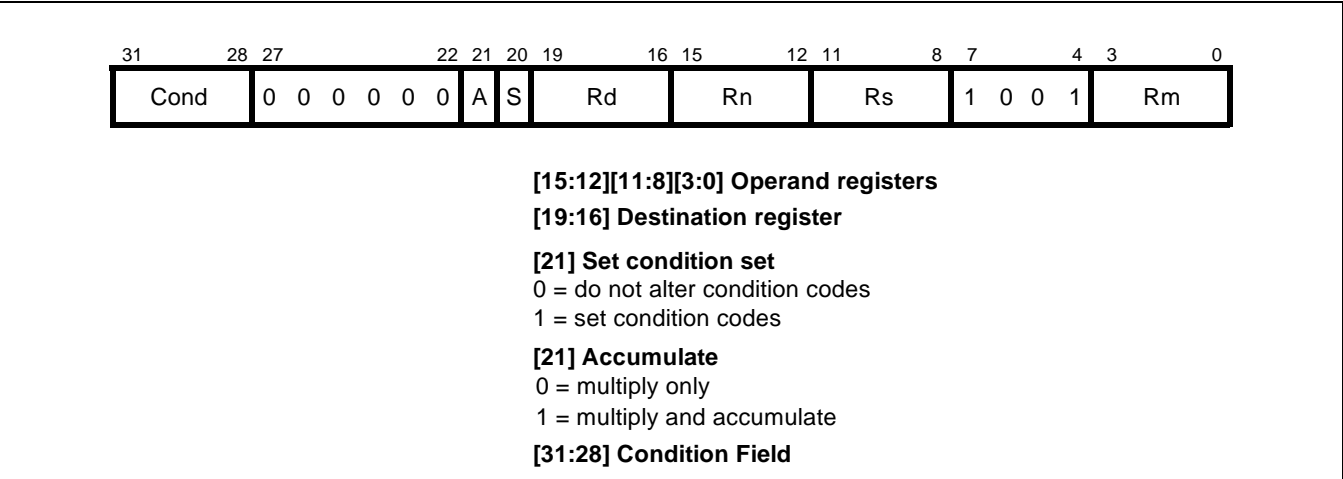


Figure 3-12. Multiply Instructions

The multiply form of the instruction gives $Rd:=Rm*Rs$. Rn is ignored, and should be set to zero for compatibility with possible future upgrades to the instruction set. The multiply-accumulate form gives $Rd:=Rm*Rs+Rn$, which can save an explicit ADD instruction in some circumstances. Both forms of the instruction work on operands which may be considered as signed (2's complement) or unsigned integers.

The results of a signed multiply and of an unsigned multiply of 32 bit operands differ only in the upper 32 bits—the low 32 bits of the signed and unsigned results are identical. As these instructions only produce the low 32 bits of a multiply, they can be used for both signed and unsigned multiplies.

For example consider the multiplication of the operands:

Operand A	Operand B	Result
0xFFFFFFFF6	0x0000001	0xFFFFFFFF38

If the Operands Are Interpreted as Signed

Operand A has the value -10, operand B has the value 20, and the result is -200 which is correctly represented as 0xFFFFFFFF38.

If the Operands Are Interpreted as Unsigned

Operand A has the value 4294967286, operand B has the value 20 and the result is 85899345720, which is represented as 0x13FFFFFF38, so the least significant 32 bits are 0xFFFFFFFF38.

Operand Restrictions

The destination register Rd must not be the same as the operand register Rm . $R15$ must not be used as an operand or as the destination register.

All other register combinations will give correct results, and Rd , Rn and Rs may use the same register when required.

CPSR FLAGS

Setting the CPSR flags is optional, and is controlled by the S bit in the instruction. The N (Negative) and Z (Zero) flags are set correctly on the result (N is made equal to bit 31 of the result, and Z is set if and only if the result is zero). The C (Carry) flag is set to a meaningless value and the V (oVerflow) flag is unaffected.

INSTRUCTION CYCLE TIMES

MUL takes $1S + mI$ and MLA $1S + (m+1)I$ cycles to execute, where S and I are defined as sequential (S-cycle) and internal (I-cycle), respectively.

m	The number of 8 bit multiplier array cycles is required to complete the multiply, which is controlled by the value of the multiplier operand specified by Rs. Its possible values are as follows
1	If bits [32:8] of the multiplier operand are all zero or all one.
2	If bits [32:16] of the multiplier operand are all zero or all one.
3	If bits [32:24] of the multiplier operand are all zero or all one.
4	In all other cases.

ASSEMBLER SYNTAX

MUL{cond}{S} Rd,Rm,Rs

MLA{cond}{S} Rd,Rm,Rs,Rn

{cond} Two-character condition mnemonic. See Table 3-2..

{S} Set condition codes if S present

Rd, Rm, Rs and Rn Expressions evaluating to a register number other than R15.

EXAMPLES

MUL	R1,R2,R3	; R1:=R2*R3
MLAEQS	R1,R2,R3,R4	; Conditionally R1:=R2*R3+R4, Setting condition codes.

MULTIPLY LONG AND MULTIPLY-ACCUMULATE LONG (MULL,MLAL)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-13.

The multiply long instructions perform integer multiplication on two 32 bit operands and produce 64 bit results. Signed and unsigned multiplication each with optional accumulate give rise to four variations.

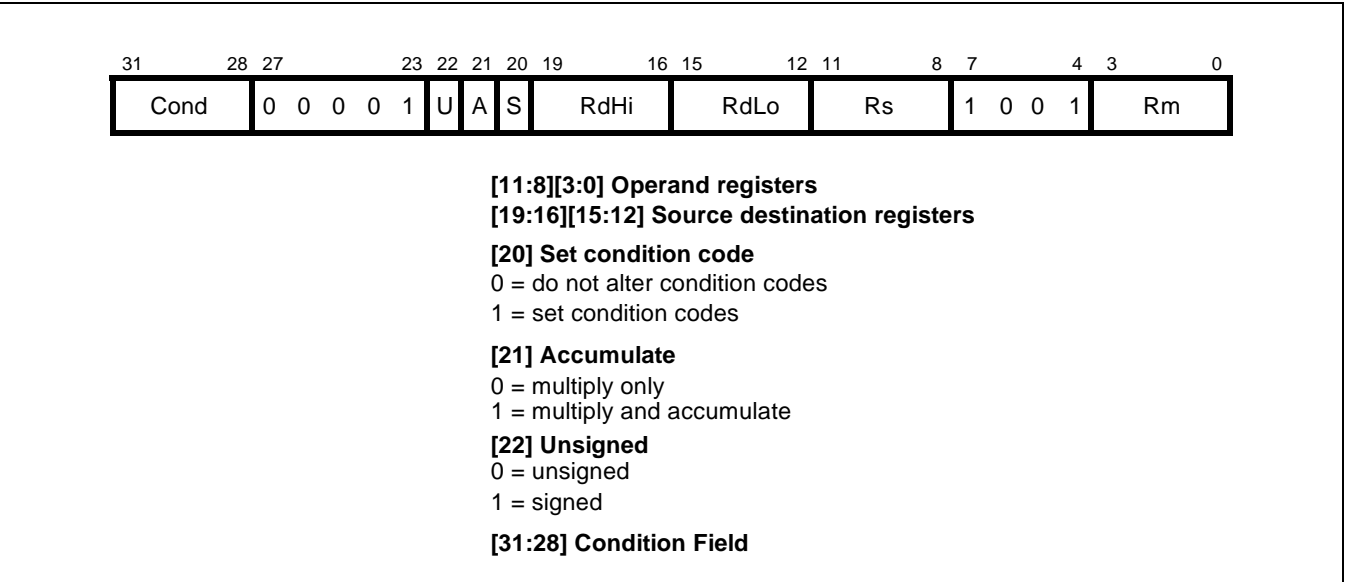


Figure 3-13. Multiply Long Instructions

The multiply forms (UMULL and SMULL) take two 32 bit numbers and multiply them to produce a 64 bit result of the form RdHi,RdLo := Rm * Rs. The lower 32 bits of the 64 bit result are written to RdLo, the upper 32 bits of the result are written to RdHi.

The multiply-accumulate forms (UMLAL and SMLAL) take two 32 bit numbers, multiply them and add a 64 bit number to produce a 64 bit result of the form RdHi,RdLo := Rm * Rs + RdHi,RdLo. The lower 32 bits of the 64 bit number to add is read from RdLo. The upper 32 bits of the 64 bit number to add is read from RdHi. The lower 32 bits of the 64 bit result are written to RdLo. The upper 32 bits of the 64 bit result are written to RdHi.

The UMULL and UMLAL instructions treat all of their operands as unsigned binary numbers and write an unsigned 64 bit result. The SMULL and SMLAL instructions treat all of their operands as two's-complement signed numbers and write a two's-complement signed 64 bit result.

OPERAND RESTRICTIONS

- R15 must not be used as an operand or as a destination register.
- RdHi, RdLo, and Rm must all specify different registers.

CPSR FLAGS

Setting the CPSR flags is optional, and is controlled by the S bit in the instruction. The N and Z flags are set correctly on the result (N is equal to bit 63 of the result, Z is set if and only if all 64 bits of the result are zero). Both the C and V flags are set to meaningless values.

INSTRUCTION CYCLE TIMES

MULL takes $1S + (m+1)I$ and MLAL $1S + (m+2)I$ cycles to execute, where m is the number of 8 bit multiplier array cycles required to complete the multiply, which is controlled by the value of the multiplier operand specified by Rs.

Its possible values are as follows:

For Signed Instructions SMULL, SMLAL:

- If bits [31:8] of the multiplier operand are all zero or all one.
- If bits [31:16] of the multiplier operand are all zero or all one.
- If bits [31:24] of the multiplier operand are all zero or all one.
- In all other cases.

For Unsigned Instructions UMULL, UMLAL:

- If bits [31:8] of the multiplier operand are all zero.
- If bits [31:16] of the multiplier operand are all zero.
- If bits [31:24] of the multiplier operand are all zero.
- In all other cases.

S and I are defined as sequential (S-cycle) and internal (I-cycle), respectively.

ASSEMBLER SYNTAX

Table 3-5. Assembler Syntax Descriptions

Mnemonic	Description	Purpose
UMULL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned Multiply Long	$32 \times 32 = 64$
UMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned Multiply & Accumulate Long	$32 \times 32 + 64 = 64$
SMULL{cond}{S} RdLo,RdHi,Rm,Rs	Signed Multiply Long	$32 \times 32 = 64$
SMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Signed Multiply & Accumulate Long	$32 \times 32 + 64 = 64$

where:

{cond}	Two-character condition mnemonic. See Table 3-2.
{S}	Set condition codes if S present
RdLo, RdHi, Rm, Rs	Expressions evaluating to a register number other than R15.

EXAMPLES

```

UMULL    R1,R4,R2,R3        ; R4,R1:=R2*R3
UMLALS   R1,R5,R2,R3        ; R5,R1:=R2*R3+R5,R1 also setting condition codes

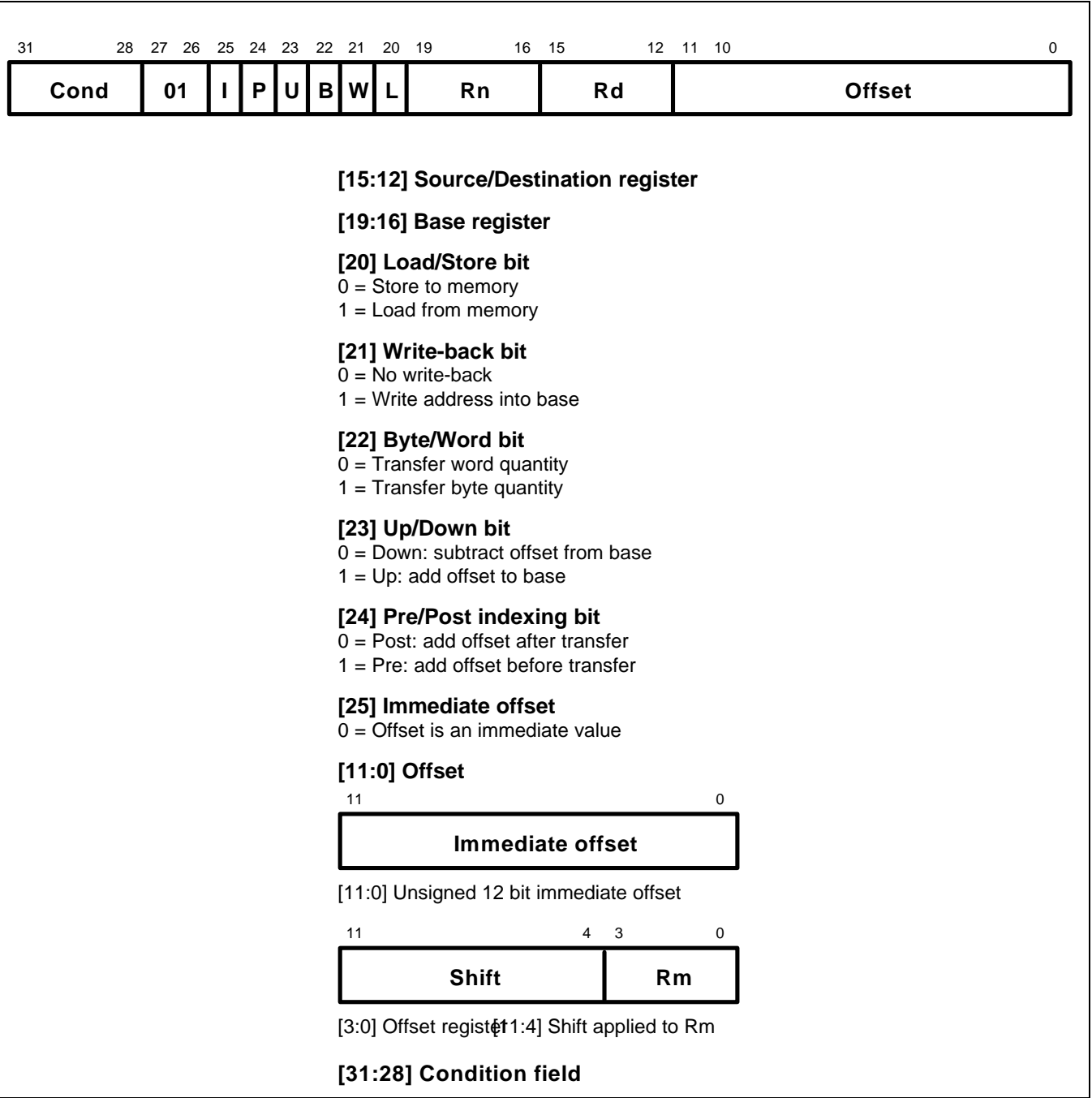
```

SINGLE DATA TRANSFER (LDR, STR)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-14.

The single data transfer instructions are used to load or store single bytes or words of data. The memory address used in the transfer is calculated by adding an offset to or subtracting an offset from a base register.

The result of this calculation may be written back into the base register if auto-indexing is required.



OFFSETS AND AUTO-INDEXING

The offset from the base may be either a 12 bit unsigned binary immediate value in the instruction, or a second register (possibly shifted in some way). The offset may be added to (U=1) or subtracted from (U=0) the base register Rn. The offset modification may be performed either before (pre-indexed, P=1) or after (post-indexed, P=0) the base is used as the transfer address.

The W bit gives optional auto increment and decrement addressing modes. The modified base value may be written back into the base (W=1), or the old base value may be kept (W=0). In the case of post-indexed addressing, the write back bit is redundant and is always set to zero, since the old base value can be retained by setting the offset to zero. Therefore post-indexed data transfers always write back the modified base. The only use of the W bit in a post-indexed data transfer is in privileged mode code, where setting the W bit forces non-privileged mode for the transfer, allowing the operating system to generate a user address in a system where the memory management hardware makes suitable use of this hardware.

SHIFTED REGISTER OFFSET

The 8 shift control bits are described in the data processing instructions section. However, the register specified shift amounts are not available in this instruction class. See Figure 3-5.

BYTES AND WORDS

This instruction class may be used to transfer a byte (B=1) or a word (B=0) between an ARM7TDMI register and memory.

The action of LDR(B) and STR(B) instructions is influenced by the **BIGEND** control signal of ARM7TDMI core. The two possible configurations are described below.

NOTE

The KS32C6200 is configured to the big-endian format.

Little-Endian Configuration

A byte load (LDRB) expects the data on data bus inputs 7 through 0 if the supplied address is on a word boundary, on data bus inputs 15 through 8 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register, and the remaining bits of the register are filled with zeros. Please see Figure 2-2.

A byte store (STRB) repeats the bottom 8 bits of the source register four times across data bus outputs 31 through 0. The external memory system should activate the appropriate byte subsystem to store the data.

A word load (LDR) will normally use a word aligned address. However, an address offset from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 0 to 7. This means that half-words accessed at offsets 0 and 2 from the word boundary will be correctly loaded into bits 0 through 15 of the register. Two shift operations are then required to clear or to sign extend the upper 16 bits.

A word store (STR) should generate a word aligned address. The word presented to the data bus is not affected if the address is not word aligned. That is, bit 31 of the register being stored always appears on data bus output 31.

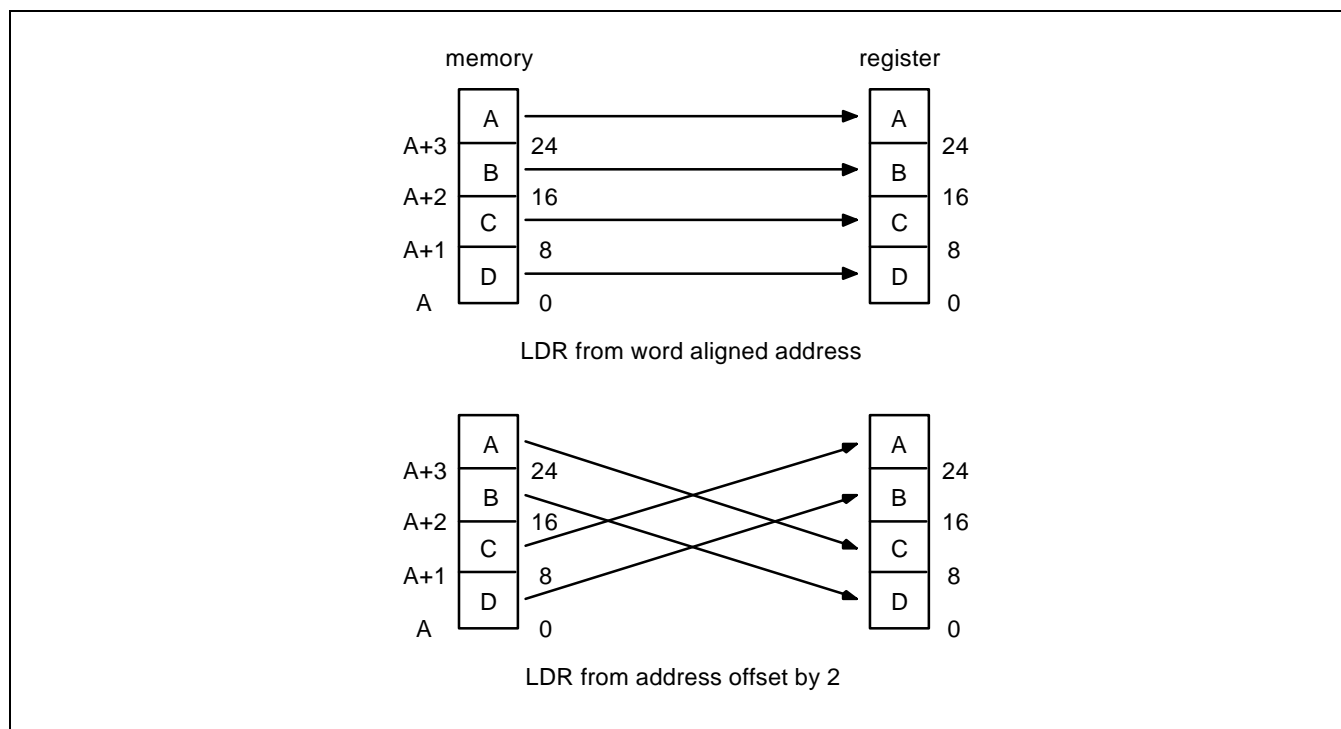


Figure 3-15. Little-Endian Offset Addressing

Big-Endian Configuration

A byte load (LDRB) expects the data on data bus inputs 31 through 24 if the supplied address is on a word boundary, on data bus inputs 23 through 16 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register and the remaining bits of the register are filled with zeros. Please see Figure 2-1.

A byte store (STRB) repeats the bottom 8 bits of the source register four times across data bus outputs 31 through 0. The external memory system should activate the appropriate byte subsystem to store the data.

A word load (LDR) should generate a word aligned address. An address offset of 0 or 2 from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 31 through 24. This means that half-words accessed at these offsets will be correctly loaded into bits 16 through 31 of the register. A shift operation is then required to move (and optionally sign extend) the data into the bottom 16 bits. An address offset of 1 or 3 from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 15 through 8.

A word store (STR) should generate a word aligned address. The word presented to the data bus is not affected if the address is not word aligned. That is, bit 31 of the register being stored always appears on data bus output 31.

USE OF R15

Write-back must not be specified if R15 is specified as the base register (Rn). When using R15 as the base register you must remember it contains an address 8 bytes on from the address of the current instruction.

R15 must not be specified as the register offset (Rm).

When R15 is the source register (Rd) of a register store (STR) instruction, the stored value will be address of the instruction plus 12.

RESTRICTION ON THE USE OF BASE REGISTER

When configured for late aborts, the following example code is difficult to unwind as the base register, Rn, gets updated before the abort handler starts. Sometimes it may be impossible to calculate the initial value.

After an abort, the following example code is difficult to unwind as the base register, Rn, gets updated before the abort handler starts. Sometimes it may be impossible to calculate the initial value.

Example:

```
LDR      R0,[R1],R1
```

Therefore a post-indexed LDR or STR where Rm is the same register as Rn should not be used.

DATA ABORTS

A transfer to or from a legal address may cause problems for a memory management system. For instance, in a system which uses virtual memory the required data may be absent from main memory. The memory manager can signal a problem by taking the processor ABORT input HIGH whereupon the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

INSTRUCTION CYCLE TIMES

Normal LDR instructions take $1S + 1N + 1I$ and LDR PC take $2S + 2N + 1I$ incremental cycles, where S, N and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively. STR instructions take $2N$ incremental cycles to execute.

ASSEMBLER SYNTAX

<LDR|STR>{cond}{B}{T} Rd,<Address>

where:

LDR	Load from memory into a register						
STR	Store from a register into memory						
{cond}	Two-character condition mnemonic. See Table 3-2.						
{B}	If B is present then byte transfer, otherwise word transfer						
{T}	If T is present the W bit will be set in a post-indexed instruction, forcing non-privileged mode for the transfer cycle. T is not allowed when a pre-indexed addressing mode is specified or implied.						
Rd	An expression evaluating to a valid register number.						
Rn and Rm	Expressions evaluating to a register number. If Rn is R15 then the assembler will subtract 8 from the offset value to allow for ARM7TDMI pipelining. In this case base write-back should not be specified.						
<Address>can be:							
1	<p>An expression which generates an address: The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.</p>						
2	<p>A pre-indexed addressing specification:</p> <table> <tr> <td>[Rn]</td><td>offset of zero</td></tr> <tr> <td>[Rn,<#expression>]{!}</td><td>offset of <expression> bytes</td></tr> <tr> <td>[Rn,{+/-}Rm{,<shift>}](!)</td><td>offset of +/- contents of index register, shifted by <shift></td></tr> </table>	[Rn]	offset of zero	[Rn,<#expression>]{!}	offset of <expression> bytes	[Rn,{+/-}Rm{,<shift>}](!)	offset of +/- contents of index register, shifted by <shift>
[Rn]	offset of zero						
[Rn,<#expression>]{!}	offset of <expression> bytes						
[Rn,{+/-}Rm{,<shift>}](!)	offset of +/- contents of index register, shifted by <shift>						
3	<p>A post-indexed addressing specification:</p> <table> <tr> <td>[Rn],<#expression></td><td>offset of <expression> bytes</td></tr> <tr> <td>[Rn],{+/-}Rm{,<shift>}</td><td>offset of +/- contents of index register, shifted as by <shift>.</td></tr> </table>	[Rn],<#expression>	offset of <expression> bytes	[Rn],{+/-}Rm{,<shift>}	offset of +/- contents of index register, shifted as by <shift>.		
[Rn],<#expression>	offset of <expression> bytes						
[Rn],{+/-}Rm{,<shift>}	offset of +/- contents of index register, shifted as by <shift>.						
<shift>	General shift operation (see data processing instructions) but you cannot specify the shift amount by a register.						
{!}	Writes back the base register (set the W bit) if ! is present.						

EXAMPLES

STR	R1,[R2,R4]!	; Store R1 at R2+R4 (both of which are registers) ; and write back address to R2.
STR	R1,[R2],R4	; Store R1 at R2 and write back R2+R4 to R2.
LDR	R1,[R2,#16]	; Load R1 from contents of R2+16, but don't write back.
LDR	R1,[R2,R3,LSL#2]	; Load R1 from contents of R2+R3*4.
LDREQB	R1,[R6,#5]	; Conditionally load byte at R6+5 into ; R1 bits 0 to 7, filling bits 8 to 31 with zeros.
STR	R1,PLACE	; Generate PC relative offset to address PLACE.
PLACE		

HALFWORD AND SIGNED DATA TRANSFER (LDRH/STRH/LDRSB/LDRSH)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-16.

These instructions are used to load or store half-words of data and also load sign-extended bytes or half-words of data. The memory address used in the transfer is calculated by adding an offset to or subtracting an offset from a base register. The result of this calculation may be written back into the base register if auto-indexing is required.

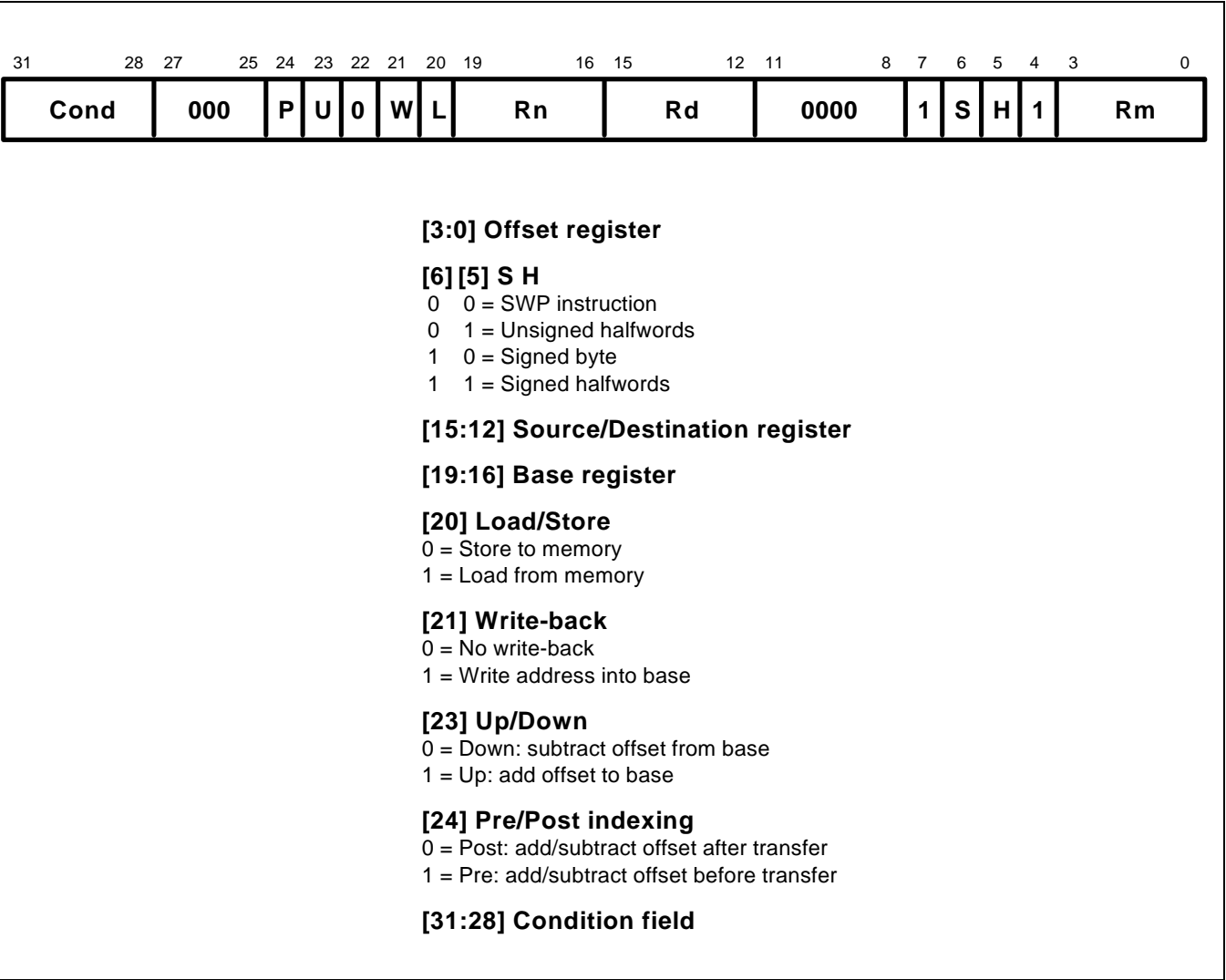


Figure 3-16. Halfword and Signed Data Transfer with Register Offset

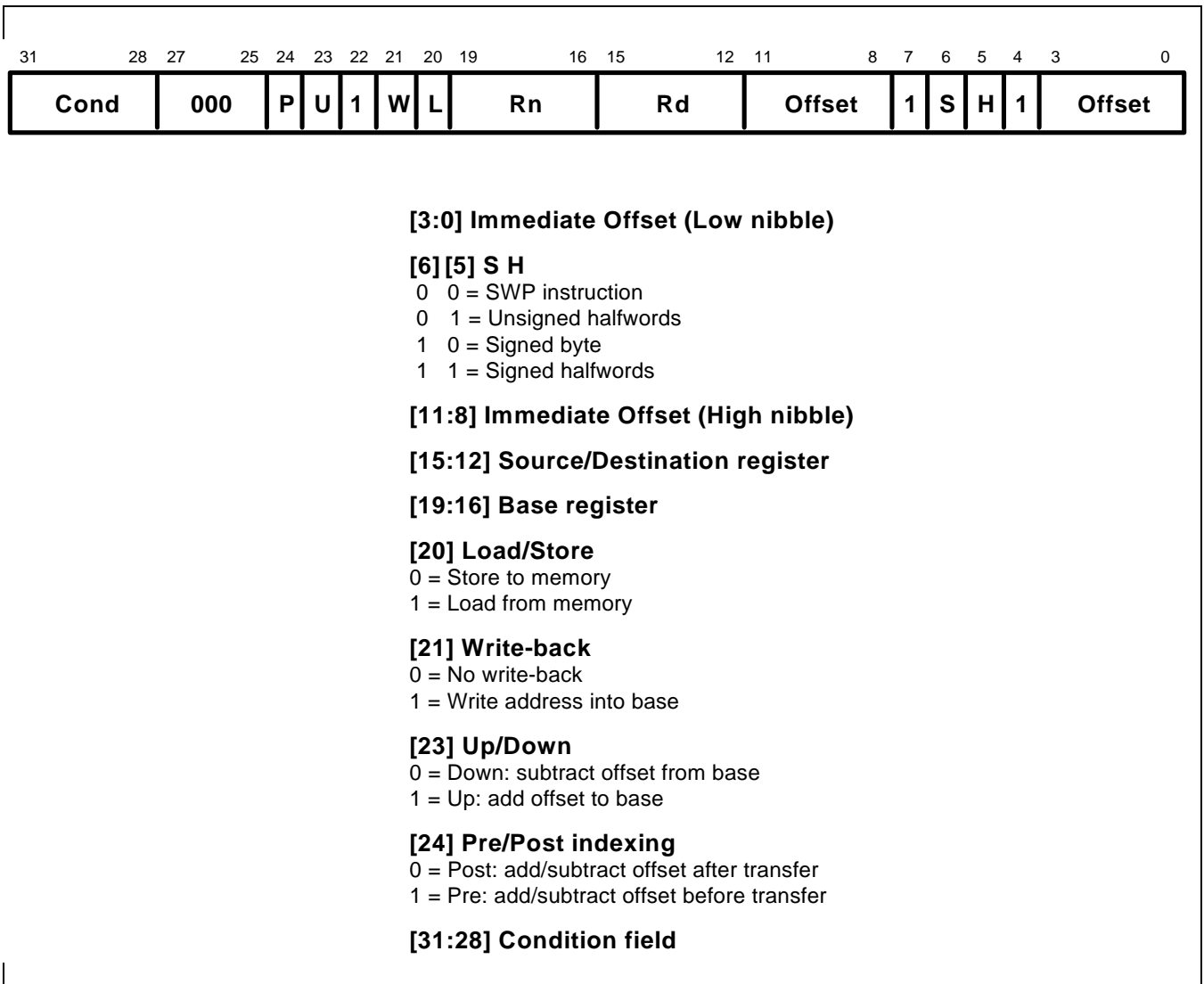


Figure 3-17. Halfword and Signed Data Transfer with Immediate Offset and Auto-Indexing

OFFSETS AND AUTO-INDEXING

The offset from the base may be either a 8-bit unsigned binary immediate value in the instruction, or a second register. The 8-bit offset is formed by concatenating bits 11 to 8 and bits 3 to 0 of the instruction word, such that bit 11 becomes the MSB and bit 0 becomes the LSB. The offset may be added to (U=1) or subtracted from (U=0) the base register Rn. The offset modification may be performed either before (pre-indexed, P=1) or after (post-indexed, P=0) the base register is used as the transfer address.

The W bit gives optional auto-increment and decrement addressing modes. The modified base value may be written back into the base (W=1), or the old base may be kept (W=0). In the case of post-indexed addressing, the write back bit is redundant and is always set to zero, since the old base value can be retained if necessary by setting the offset to zero. Therefore post-indexed data transfers always write back the modified base.

The Write-back bit should not be set high (W=1) when post-indexed addressing is selected.

HALFWORD LOAD AND STORES

Setting S=0 and H=1 may be used to transfer unsigned Half-words between an ARM7TDMI register and memory.

The action of LDRH and STRH instructions is influenced by the BIGEND control signal. The two possible configurations are described in the section below.

SIGNED BYTE AND HALFWORD LOADS

The S bit controls the loading of sign-extended data. When S=1 the H bit selects between Bytes (H=0) and Half-words (H=1). The L bit should not be set low (Store) when Signed (S=1) operations have been selected.

The LDRSB instruction loads the selected Byte into bits 7 to 0 of the destination register and bits 31 to 8 of the destination register are set to the value of bit 7, the sign bit.

The LDRSH instruction loads the selected Half-word into bits 15 to 0 of the destination register and bits 31 to 16 of the destination register are set to the value of bit 15, the sign bit.

The action of the LDRSB and LDRSH instructions is influenced by the BIGEND control signal. The two possible configurations are described in the following section.

ENDIANNESS AND BYTE/HALFWORD SELECTION

Little-Endian Configuration

A signed byte load (LDRSB) expects data on data bus inputs 7 through to 0 if the supplied address is on a word boundary, on data bus inputs 15 through to 8 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bit of the destination register, and the remaining bits of the register are filled with the sign bit, bit 7 of the byte. Please see Figure 2-2.

A halfword load (LDRSH or LDRH) expects data on data bus inputs 15 through to 0 if the supplied address is on a word boundary and on data bus inputs 31 through to 16 if it is a halfword boundary, (A[1]=1). The supplied address should always be on a halfword boundary. If bit 0 of the supplied address is HIGH then the ARM7TDMI will load an unpredictable value. The selected halfword is placed in the bottom 16 bits of the destination register. For unsigned half-words (LDRH), the top 16 bits of the register are filled with zeros and for signed half-words (LDRSH) the top 16 bits are filled with the sign bit, bit 15 of the halfword.

A halfword store (STRH) repeats the bottom 16 bits of the source register twice across the data bus outputs 31 through to 0. The external memory system should activate the appropriate halfword subsystem to store the data. Note that the address must be halfword aligned, if bit 0 of the address is HIGH this will cause unpredictable behaviour.

Big-Endian Configuration

A signed byte load (LDRSB) expects data on data bus inputs 31 through to 24 if the supplied address is on a word boundary, on data bus inputs 23 through to 16 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bit of the destination register, and the remaining bits of the register are filled with the sign bit, bit 7 of the byte. Please see Figure 2-1.

A halfword load (LDRSH or LDRH) expects data on data bus inputs 31 through to 16 if the supplied address is on a word boundary and on data bus inputs 15 through to 0 if it is a halfword boundary, ($A[1]=1$). The supplied address should always be on a halfword boundary. If bit 0 of the supplied address is HIGH then the ARM7TDMI will load an unpredictable value. The selected halfword is placed in the bottom 16 bits of the destination register. For unsigned half-words (LDRH), the top 16 bits of the register are filled with zeros and for signed half-words (LDRSH) the top 16 bits are filled with the sign bit, bit 15 of the halfword.

A halfword store (STRH) repeats the bottom 16 bits of the source register twice across the data bus outputs 31 through to 0. The external memory system should activate the appropriate halfword subsystem to store the data. Note that the address must be halfword aligned, if bit 0 of the address is HIGH this will cause unpredictable behaviour.

NOTE

The KS32C6200 is configured to the big-endian format.

USE OF R15

Write-back should not be specified if R15 is specified as the base register (R_n). When using R15 as the base register you must remember it contains an address 8 bytes on from the address of the current instruction.

R15 should not be specified as the register offset (R_m).

When R15 is the source register (R_d) of a Half-word store (STRH) instruction, the stored address will be address of the instruction plus 12.

DATA ABORTS

A transfer to or from a legal address may cause problems for a memory management system. For instance, in a system which uses virtual memory the required data may be absent from the main memory. The memory manager can signal a problem by taking the processor ABORT input HIGH whereupon the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

INSTRUCTION CYCLE TIMES

Normal LDR(H,SH,SB) instructions take $1S + 1N + 1I$. LDR(H,SH,SB) PC take $2S + 2N + 1I$ incremental cycles. S, N and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively. STRH instructions take 2N incremental cycles to execute.

ASSEMBLER SYNTAX

<LDR|STR>{cond}<H|SH|SB> Rd,<address>

LDR	Load from memory into a register
STR	Store from a register into memory
{cond}	Two-character condition mnemonic. See Table 3-2..
H	Transfer halfword quantity
SB	Load sign extended byte (Only valid for LDR)
SH	Load sign extended halfword (Only valid for LDR)
Rd	An expression evaluating to a valid register number.

<address> can be:

- 1 An expression which generates an address:
The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.
- 2 A pre-indexed addressing specification:

[Rn]	offset of zero
[Rn,<#expression>]{!}	offset of <expression> bytes
[Rn,{+/-}Rm]{!}	offset of +/- contents of index register
- 3 A post-indexed addressing specification:

[Rn,<#expression>	offset of <expression> bytes
[Rn,{+/-}Rm	offset of +/- contents of index register.
- 4 Rn and Rm are expressions evaluating to a register number. If Rn is R15 then the assembler will subtract 8 from the offset value to allow for ARM7TDMI pipelining. In this case base write-back should not be specified.
- {!} Writes back the base register (set the W bit) if ! is present.

EXAMPLES

LDRH	R1,[R2,-R3]!	; Load R1 from the contents of the halfword address ; contained in R2-R3 (both of which are registers) ; and write back address to R2
STRH	R3,[R4,#14]	; Store the halfword in R3 at R14+14 but don't write back.
LDRSB	R8,[R2],#-223	; Load R8 with the sign extended contents of the byte ; address contained in R2 and write back R2-223 to R2.
LDRNESH	R11,[R0]	; Conditionally load R11 with the sign extended contents of ; the halfword address contained in R0.
HERE		; Generate PC relative offset to address FRED.
STRH	R5,[PC,#(FRED-HERE-8)];	Store the halfword in R5 at address FRED
FRED		

BLOCK DATA TRANSFER (LDM, STM)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-18.

Block data transfer instructions are used to load (LDM) or store (STM) any subset of the currently visible registers. They support all possible stacking modes, maintaining full or empty stacks which can grow up or down memory, and are very efficient instructions for saving or restoring context, or for moving large blocks of data around main memory.

THE REGISTER LIST

The instruction can cause the transfer of any registers in the current bank (and non-user mode programs can also transfer to and from the user bank, see below). The register list is a 16 bit field in the instruction, with each bit corresponding to a register. A 1 in bit 0 of the register field will cause R0 to be transferred, a 0 will cause it not to be transferred; similarly bit 1 controls the transfer of R1, and so on.

Any subset of the registers, or all the registers, may be specified. The only restriction is that the register list should not be empty.

Whenever R15 is stored to memory the stored value is the address of the STM instruction plus 12.

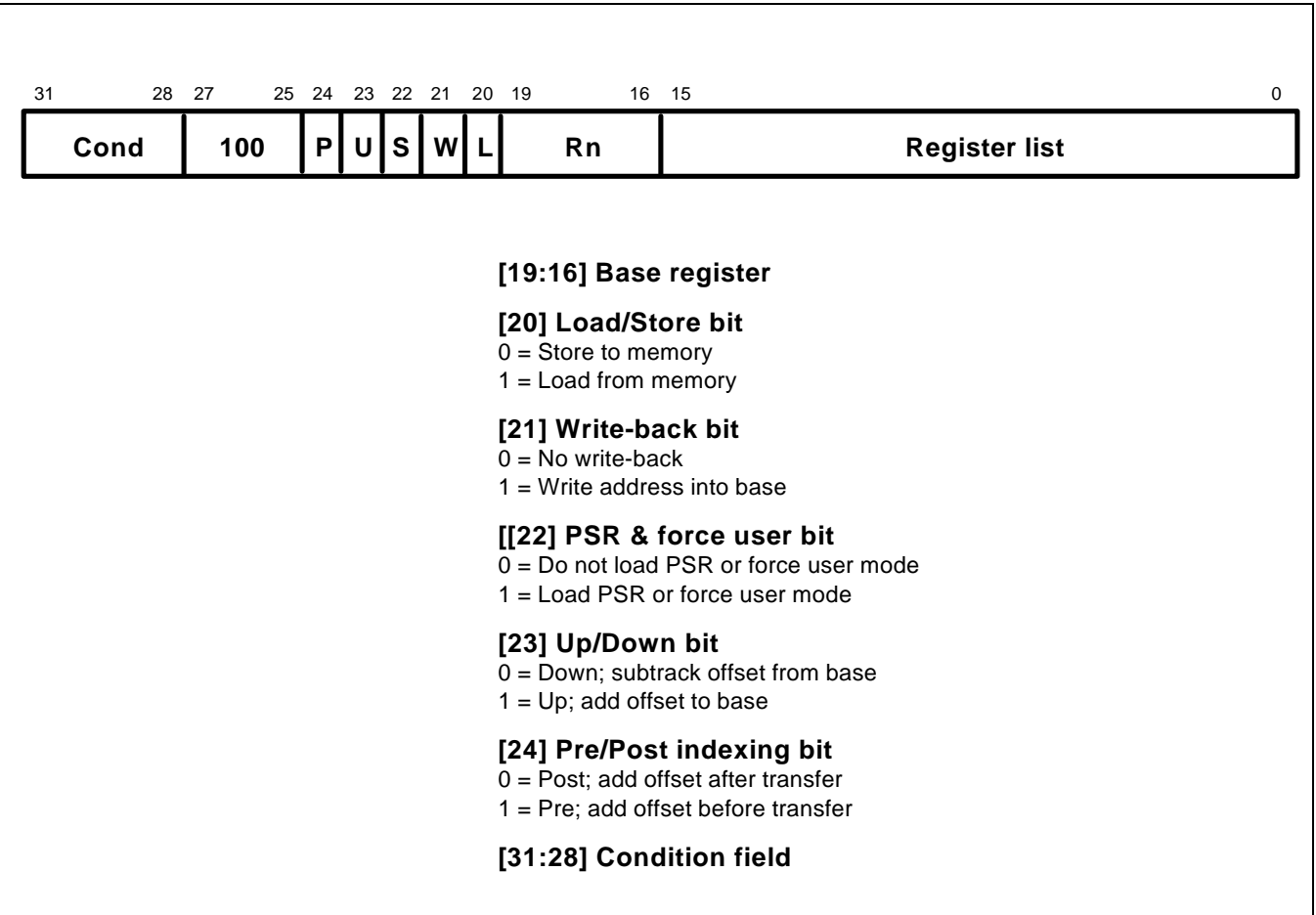


Figure 3-18. Block Data Transfer Instructions

ADDRESSING MODES

The transfer addresses are determined by the contents of the base register (Rn), the pre/post bit (P) and the up/down bit (U). The registers are transferred in the order lowest to highest, so R15 (if in the list) will always be transferred last. The lowest register also gets transferred to/from the lowest memory address. By way of illustration, consider the transfer of R1, R5 and R7 in the case where Rn=0x1000 and write back of the modified base is required (W=1). Figure 3.19–22 show the sequence of register transfers, the addresses used, and the value of Rn after the instruction has completed.

In all cases, had write back of the modified base not been required (W=0), Rn would have retained its initial value of 0x1000 unless it was also in the transfer list of a load multiple register instruction, when it would have been overwritten with the loaded value.

ADDRESS ALIGNMENT

The address should normally be a word aligned quantity and non-word aligned addresses do not affect the instruction. However, the bottom 2 bits of the address will appear on **A[1:0]** and might be interpreted by the memory system.

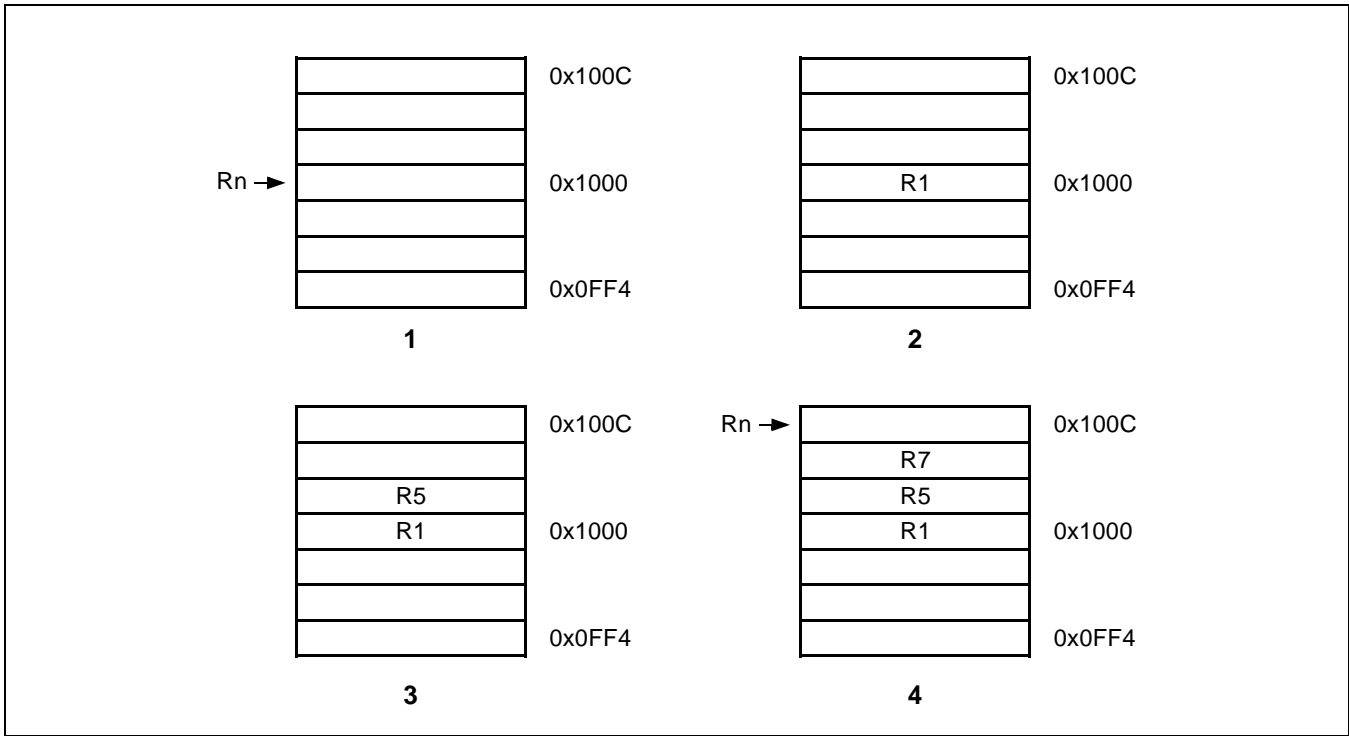


Figure 3-19. Post-Increment Addressing

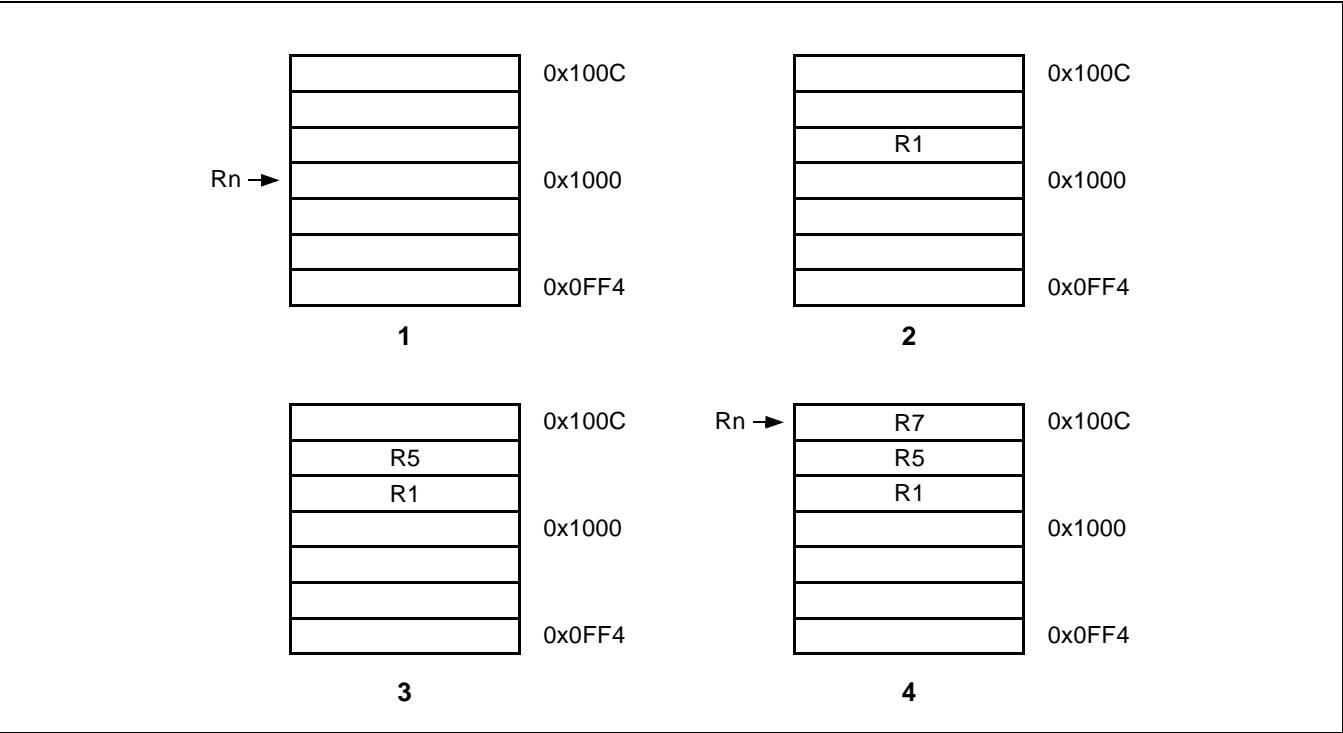


Figure 3-20. Pre-Increment Addressing

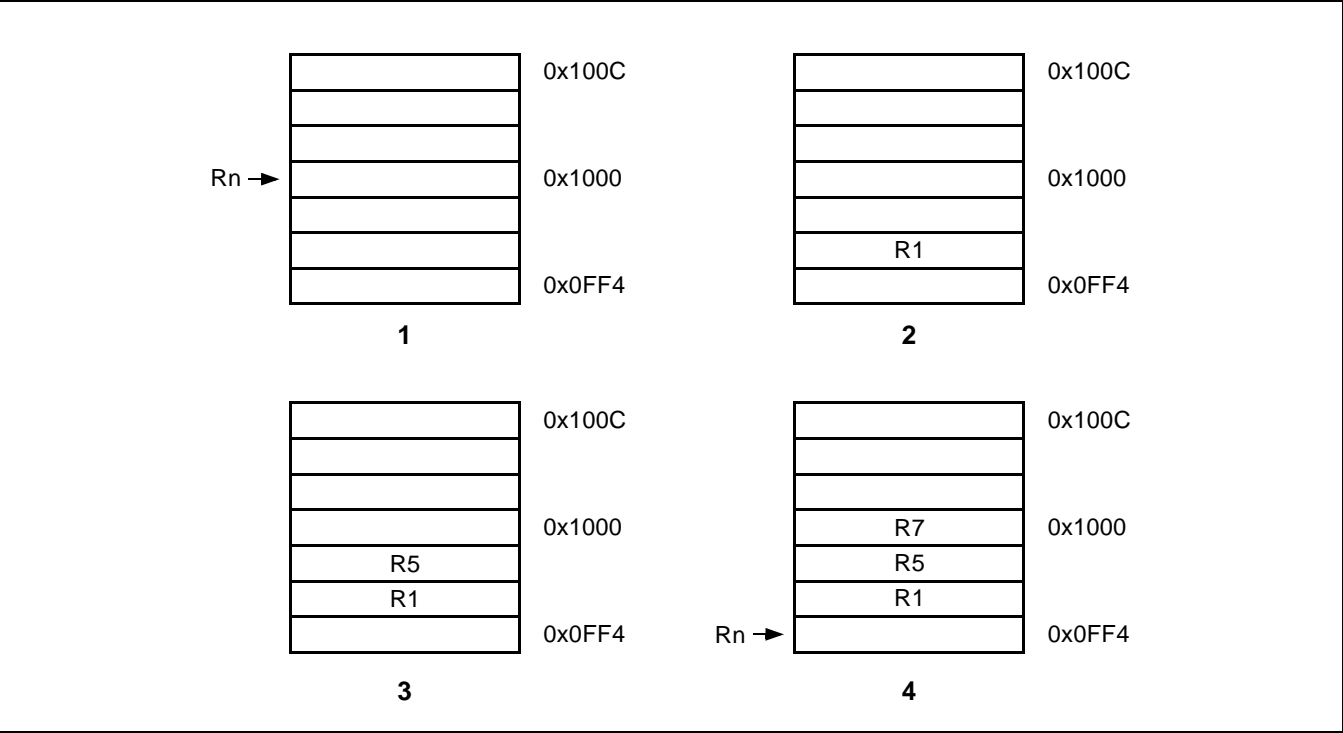


Figure 3-21. Post-Decrement Addressing

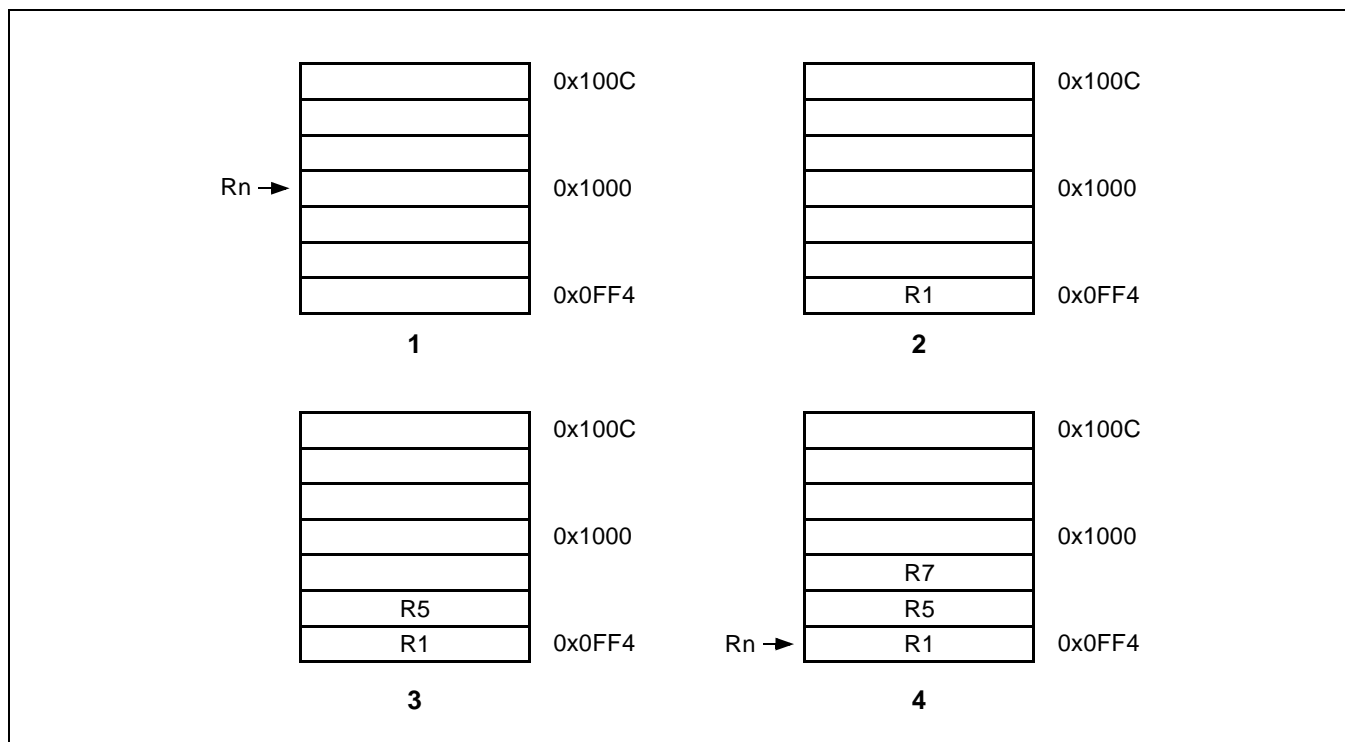


Figure 3-22. Pre-Decrement Addressing

USE OF THE S BIT

When the S bit is set in a LDM/STM instruction its meaning depends on whether or not R15 is in the transfer list and on the type of instruction. The S bit should only be set if the instruction is to execute in a privileged mode.

LDM with R15 in Transfer List and S Bit Set (Mode Changes)

If the instruction is a LDM then SPSR_<mode> is transferred to CPSR at the same time as R15 is loaded.

STM with R15 in Transfer List and S Bit Set (User Bank Transfer)

The registers transferred are taken from the User bank rather than the bank corresponding to the current mode. This is useful for saving the user state on process switches. Base write-back should not be used when this mechanism is employed.

R15 not in List and S Bit Set (User Bank Transfer)

For both LDM and STM instructions, the User bank registers are transferred rather than the register bank corresponding to the current mode. This is useful for saving the user state on process switches. Base write-back should not be used when this mechanism is employed.

When the instruction is LDM, care must be taken not to read from a banked register during the following cycle (inserting a dummy instruction such as MOV R0, R0 after the LDM will ensure safety).

USE OF R15 AS THE BASE

R15 should not be used as the base register in any LDM or STM instruction.

INCLUSION OF THE BASE IN THE REGISTER LIST

When write-back is specified, the base is written back at the end of the second cycle of the instruction. During a STM, the first register is written out at the start of the second cycle. A STM which includes storing the base, with the base as the first register to be stored, will therefore store the unchanged value, whereas with the base second or later in the transfer order, will store the modified value. A LDM will always overwrite the updated base if the base is in the list.

DATA ABORTS

Some legal addresses may be unacceptable to a memory management system, and the memory manager can indicate a problem with an address by taking the **ABORT** signal HIGH. This can happen on any transfer during a multiple register load or store, and must be recoverable if ARM7TDMI is to be used in a virtual memory system.

Aborts during STM Instructions

If the abort occurs during a store multiple instruction, ARM7TDMI takes little action until the instruction completes, whereupon it enters the data abort trap. The memory manager is responsible for preventing erroneous writes to the memory. The only change to the internal state of the processor will be the modification of the base register if write-back was specified, and this must be reversed by software (and the cause of the abort resolved) before the instruction may be retried.

Aborts during LDM Instructions

When ARM7TDMI detects a data abort during a load multiple instruction, it modifies the operation of the instruction to ensure that recovery is possible.

- Overwriting of registers stops when the abort happens. The aborting load will not take place but earlier ones may have overwritten registers. The PC is always the last register to be written and so will always be preserved.
- The base register is restored, to its modified value if write-back was requested. This ensures recoverability in the case where the base register is also in the transfer list, and may have been overwritten before the abort occurred.

The data abort trap is taken when the load multiple has completed, and the system software must undo any base modification (and resolve the cause of the abort) before restarting the instruction.

INSTRUCTION CYCLE TIMES

Normal LDM instructions take $nS + 1N + 1I$ and LDM PC takes $(n+1)S + 2N + 1I$ incremental cycles, where S, N and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively. STM instructions take $(n-1)S + 2N$ incremental cycles to execute, where n is the number of words transferred.

ASSEMBLER SYNTAX

<LDM|STM>{cond}<FD|ED|FA|EA|IA|IB|DA|DB> Rn{!},<Rlist>{^}

where:

{cond}	Two character condition mnemonic. See Table 3-2.
Rn	An expression evaluating to a valid register number
<Rlist>	A list of registers and register ranges enclosed in {} (e.g. {R0,R2-R7,R10}).
{!}	If present requests write-back (W=1), otherwise W=0.
{^}	If present set S bit to load the CPSR along with the PC, or force transfer of user bank when in privileged mode.

Addressing Mode Names

There are different assembler mnemonics for each of the addressing modes, depending on whether the instruction is being used to support stacks or for other purposes. The equivalence between the names and the values of the bits in the instruction are shown in the following table 3-6.

Table 3-6. Addressing Mode Names

Name	Stack	Other	L bit	P bit	U bit
Pre-Increment Load	LDMED	LDMIB	1	1	1
Post-Increment Load	LDMFD	LDMIA	1	0	1
Pre-Decrement Load	LDMEA	LDMDB	1	1	0
Post-Decrement Load	LDMFA	LDMDA	1	0	0
Pre-Increment Store	STMFA	STMIB	0	1	1
Post-Increment Store	STMEA	STMIA	0	0	1
Pre-Decrement Store	STMFD	STMDB	0	1	0
Post-Decrement Store	STMED	STMDA	0	0	0

FD, ED, FA, EA define pre/post indexing and the up/down bit by reference to the form of stack required. The F and E refer to a “full” or “empty” stack, i.e. whether a pre-index has to be done (full) before storing to the stack. The A and D refer to whether the stack is ascending or descending. If ascending, a STM will go up and LDM down, if descending, vice-versa.

IA, IB, DA, DB allow control when LDM/STM are not being used for stacks and simply mean Increment After, Increment Before, Decrement After, Decrement Before.

EXAMPLES

LDMFD	SP!,{R0,R1,R2}	; Unstack 3 registers.
STMIA	R0,{R0-R15}	; Save all registers.
LDMFD	SP!,{R15}	; R15 <- (SP), CPSR unchanged.
LDMFD	SP!,{R15}^	; R15 <- (SP), CPSR <- SPSR_mode
		; (allowed only in privileged modes).
STMFD	R13,{R0-R14}^	; Save user mode regs on stack
		; (allowed only in privileged modes).

These instructions may be used to save state on subroutine entry, and restore it efficiently on return to the calling routine:

STMED	SP!,{R0-R3,R14}	; Save R0 to R3 to use as workspace
		; and R14 for returning.
BL	somewhere	; This nested call will overwrite R14
LDMED	SP!,{R0-R3,R15}	; Restore workspace and return.

SINGLE DATA SWAP (SWP)

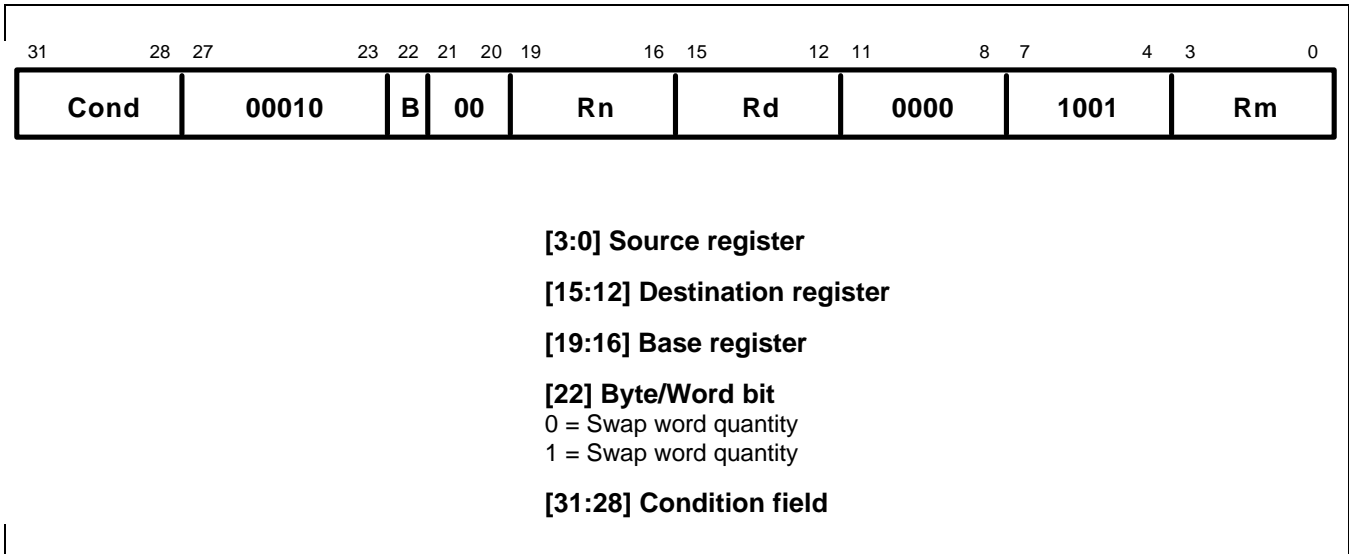


Figure 3-23. Swap Instruction

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-23.

The data swap instruction is used to swap a byte or word quantity between a register and external memory. This instruction is implemented as a memory read followed by a memory write which are “locked” together (the processor cannot be interrupted until both operations have completed, and the memory manager is warned to treat them as inseparable). This class of instruction is particularly useful for implementing software semaphores.

The swap address is determined by the contents of the base register (Rn). The processor first reads the contents of the swap address. Then it writes the contents of the source register (Rm) to the swap address, and stores the old memory contents in the destination register (Rd). The same register may be specified as both the source and destination.

The **LOCK** output goes HIGH for the duration of the read and write operations to signal to the external memory manager that they are locked together, and should be allowed to complete without interruption. This is important in multi-processor systems where the swap instruction is the only indivisible instruction which may be used to implement semaphores; control of the memory must not be removed from a processor while it is performing a locked operation.

BYTES AND WORDS

This instruction class may be used to swap a byte (B=1) or a word (B=0) between an ARM7TDMI register and memory. The SWP instruction is implemented as a LDR followed by a STR and the action of these is as described in the section on single data transfers. In particular, the description of Big and Little Endian configuration applies to the SWP instruction.

USE OF R15

Do not use R15 as an operand (Rd, Rn or Rs) in a SWP instruction.

DATA ABORTS

If the address used for the swap is unacceptable to a memory management system, the memory manager can flag the problem by driving ABORT HIGH. This can happen on either the read or the write cycle (or both), and in either case, the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

INSTRUCTION CYCLE TIMES

Swap instructions take $1S + 2N + 1I$ incremental cycles to execute, where S,N and I are defined as sequential (S-cycle), non-sequential, and internal (I-cycle), respectively.

ASSEMBLER SYNTAX

<SWP>{cond}{B} Rd,Rm,[Rn]

{cond} Two-character condition mnemonic. See Table 3-2.

{B} If B is present then byte transfer, otherwise word transfer

Rd,Rm,Rn Expressions evaluating to valid register numbers

EXAMPLES

SWP	R0,R1,[R2]	; Load R0 with the word addressed by R2, and ; store R1 at R2.
SWPB	R2,R3,[R4]	; Load R2 with the byte addressed by R4, and ; store bits 0 to 7 of R3 at R4.
SWPEQ	R0,R0,[R1]	; Conditionally swap the contents of the ; word addressed by R1 with R0.

SOFTWARE INTERRUPT (SWI)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-24, below.

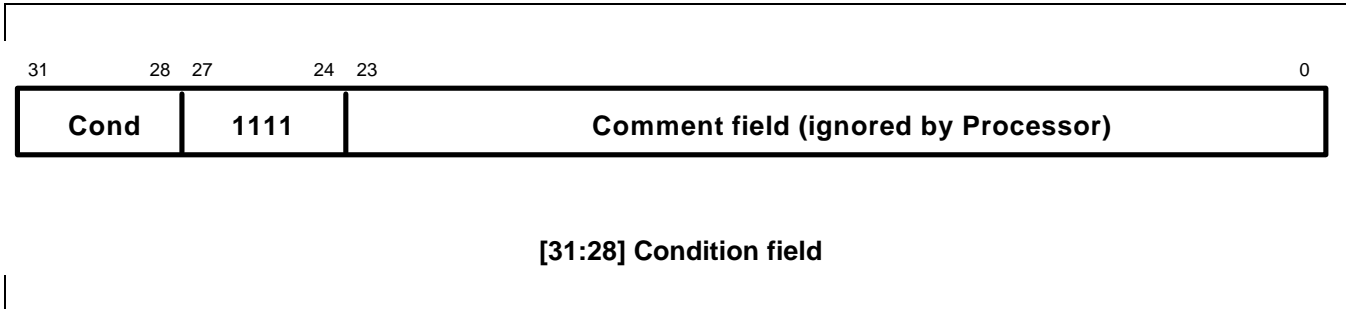


Figure 3-24. Software Interrupt Instruction

The software interrupt instruction is used to enter Supervisor mode in a controlled manner. The instruction causes the software interrupt trap to be taken, which effects the mode change. The PC is then forced to a fixed value (0x08) and the CPSR is saved in SPSR_svc. If the SWI vector address is suitably protected (by external memory management hardware) from modification by the user, a fully protected operating system may be constructed.

RETURN FROM THE SUPERVISOR

The PC is saved in R14_svc upon entering the software interrupt trap, with the PC adjusted to point to the word after the SWI instruction. MOVS PC,R14_svc will return to the calling program and restore the CPSR.

Note that the link mechanism is not re-entrant, so if the supervisor code wishes to use software interrupts within itself it must first save a copy of the return address and SPSR.

COMMENT FIELD

The bottom 24 bits of the instruction are ignored by the processor, and may be used to communicate information to the supervisor code. For instance, the supervisor may look at this field and use it to index into an array of entry points for routines which perform the various supervisor functions.

INSTRUCTION CYCLE TIMES

Software interrupt instructions take 2S + 1N incremental cycles to execute, where S and N are defined as sequential (S-cycle) and non-sequential (N-cycle).

ASSEMBLER SYNTAX

SWI{cond} <expression>

{cond} Two character condition mnemonic, Table 3-2.

<expression> Evaluated and placed in the comment field (which is ignored by ARM7TDMI).

EXAMPLES

```

SWI      ReadC           ; Get next character from read stream.
SWI      Writel+"k"      ; Output a "k" to the write stream.
SWINE    0               ; Conditionally call supervisor with 0 in comment field.

```

Supervisor code

The previous examples assume that suitable supervisor code exists, for instance:

```

0x08 B Supervisor      ; SWI entry point
EntryTable             ; Addresses of supervisor routines
DCD ZeroRtn
DCD ReadCRtn
DCD WritelRtn
...
Zero EQU 0
ReadC EQU 256
Writel EQU 512

Supervisor             ; SWI has routine required in bits 8-23 and data (if any) in
                       ; bits 0-7. Assumes R13_svc points to a suitable stack
STMFD R13,{R0-R2,R14}  ; Save work registers and return address.
LDR R0,[R14,#-4]        ; Get SWI instruction.
BIC R0,R0,#0xFF000000   ; Clear top 8 bits.
MOV R1,R0,LSR#8         ; Get routine offset.
ADR R2,EntryTable       ; Get start address of entry table.
LDR R15,[R2,R1,LSL#2]    ; Branch to appropriate routine.
WritelRtn              ; Enter with character in R0 bits 0-7.
...
LDMFD R13,{R0-R2,R15}^  ; Restore workspace and return,
                       ; restoring processor mode and flags.

```

COPROCESSOR DATA OPERATIONS (CDP)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-25.

This class of instruction is used to tell a coprocessor to perform some internal operation. No result is communicated back to ARM7TDMI, and it will not wait for the operation to complete. The coprocessor could contain a queue of such instructions awaiting execution, and their execution can overlap other activity, allowing the coprocessor and ARM7TDMI to perform independent tasks in parallel.

COPROCESSOR INSTRUCTIONS

The KS32C6200, unlike some other ARM-based processors, does not have an external coprocessor interface. It does not have a on-chip coprocessor also.

So then all coprocessor instructions will cause the undefined instruction trap to be taken on the KS32C6200. These coprocessor instructions can be emulated by the undefined trap handler. Even though external coprocessor can not be connected to the KS32C6200, the coprocessor instructions are still described here in full for completeness. (Remember that any external coprocessor described in this section is a software emulation.)

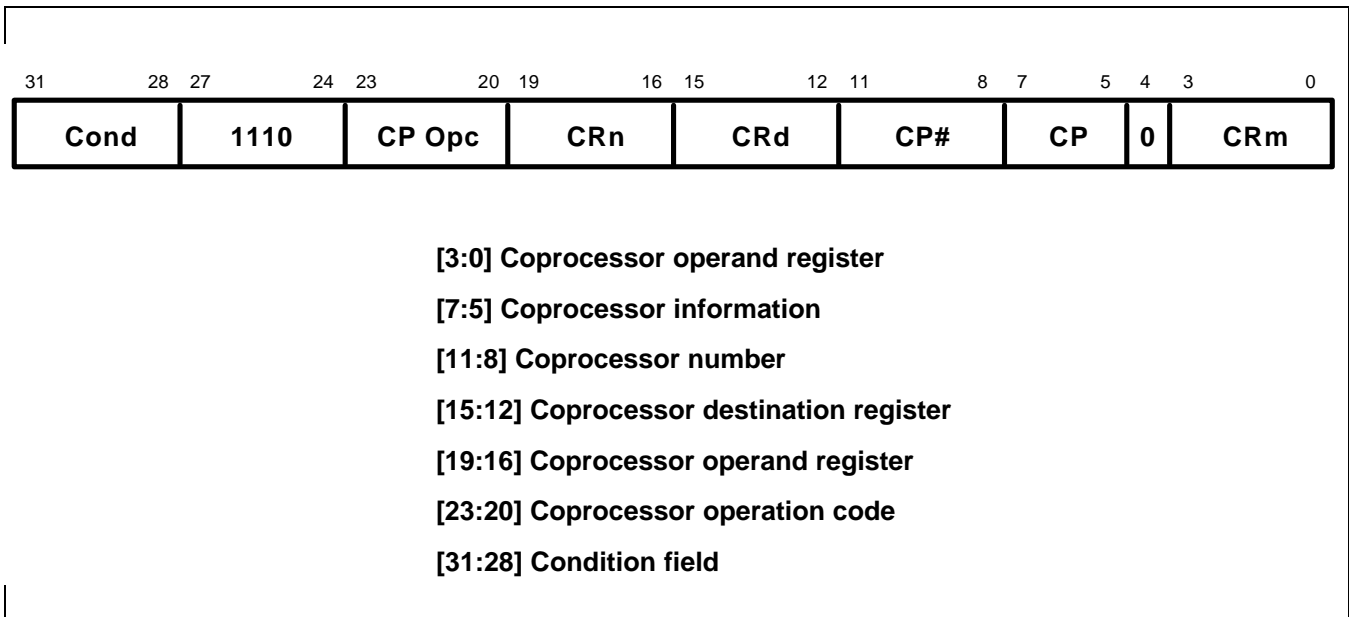


Figure 3-25. Coprocessor Data Operation Instruction

THE COPROCESSOR FIELDS

Only bit 4 and bits 24 to 31 are significant to ARM7TDMI. The remaining bits are used by coprocessors. The above field names are used by convention, and particular coprocessors may redefine the use of all fields except CP# as appropriate. The CP# field is used to contain an identifying number (in the range 0 to 15) for each coprocessor, and a coprocessor will ignore any instruction which does not contain its number in the CP# field.

The conventional interpretation of the instruction is that the coprocessor should perform an operation specified in the CP Opc field (and possibly in the CP field) on the contents of CRn and CRm, and place the result in CRd.

INSTRUCTION CYCLE TIMES

Coprocessor data operations take $1S + bI$ incremental cycles to execute, where b is the number of cycles spent in the coprocessor busy-wait loop.

S and I are defined as sequential (S-cycle) and internal (I-cycle).

ASSEMBLER SYNTAX

CDP{cond} p#,<expression1>,cd,cn,cm{,<expression2>}

{cond} Two character condition mnemonic. See Table 3-2.

p# The unique number of the required coprocessor

<expression1> Evaluated to a constant and placed in the CP Opc field

cd, cn and cm Evaluate to the valid coprocessor register numbers CRd, CRn and CRm respectively

<expression2> Where present is evaluated to a constant and placed in the CP field

EXAMPLES

CDP	p1,10,c1,c2,c3	; Request coproc 1 to do operation 10 ; on CR2 and CR3, and put the result in CR1.
CDPEQ	p2,5,c1,c2,c3,2	; If Z flag is set request coproc 2 to do operation 5 (type 2) ; on CR2 and CR3, and put the result in CR1.

COPROCESSOR DATA TRANSFERS (LDC, STC)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-26.

This class of instruction is used to load (LDC) or store (STC) a subset of a coprocessor's registers directly to memory. ARM7TDMI is responsible for supplying the memory address, and the coprocessor supplies or accepts the data and controls the number of words transferred.

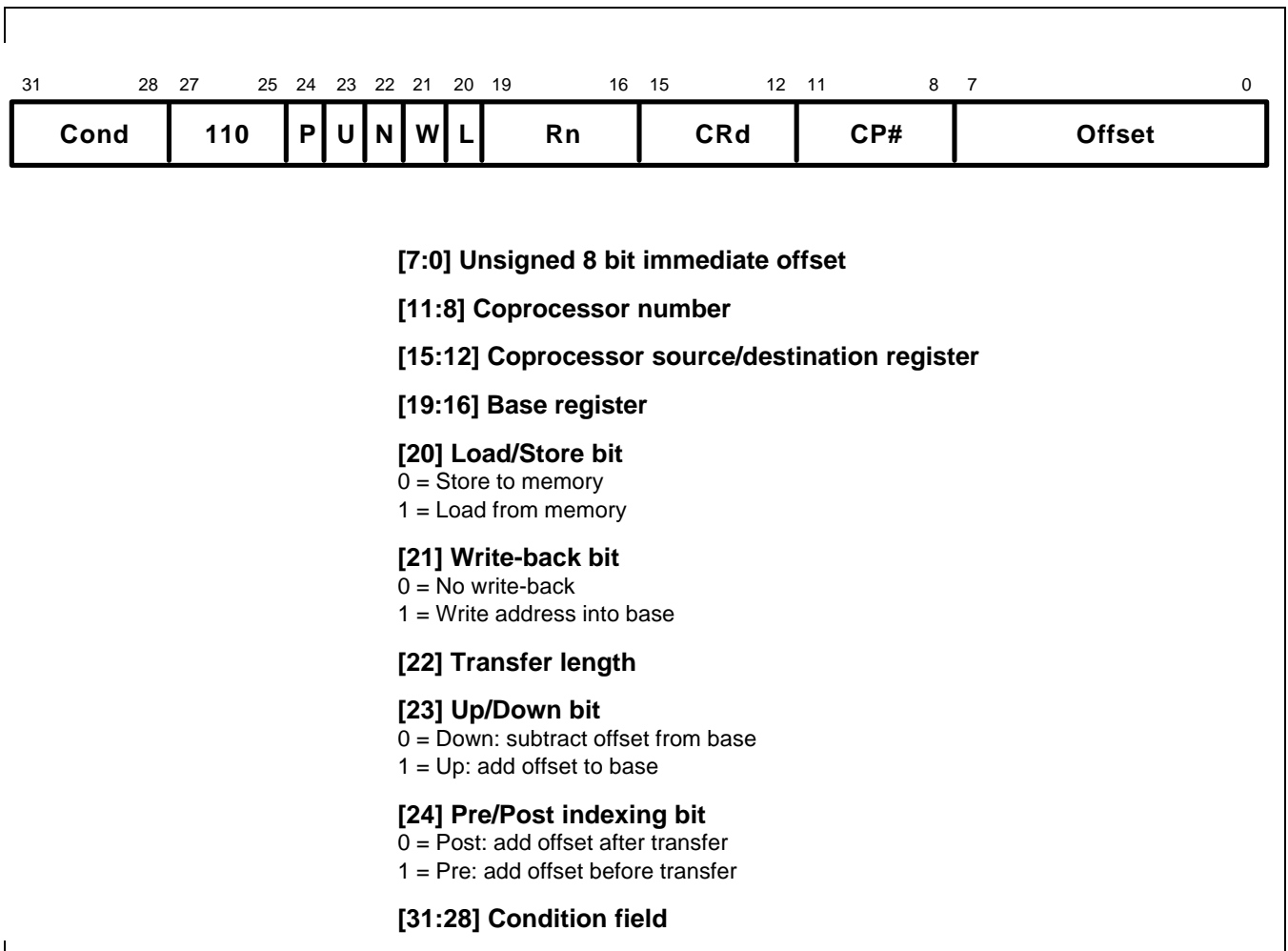


Figure 3-26. Coprocessor Data Transfer Instructions

THE COPROCESSOR FIELDS

The CP# field is used to identify the coprocessor which is required to supply or accept the data, and a coprocessor will only respond if its number matches the contents of this field.

The CRd field and the N bit contain information for the coprocessor which may be interpreted in different ways by different coprocessors, but by convention CRd is the register to be transferred (or the first register where more than one is to be transferred), and the N bit is used to choose one of two transfer length options. For instance N=0 could select the transfer of a single register, and N=1 could select the transfer of all the registers for context switching.

ADDRESSING MODES

ARM7TDMI is responsible for providing the address used by the memory system for the transfer, and the addressing modes available are a subset of those used in single data transfer instructions. Note, however, that the immediate offsets are 8 bits wide and specify word offsets for coprocessor data transfers, whereas they are 12 bits wide and specify byte offsets for single data transfers.

The 8 bit unsigned immediate offset is shifted left 2 bits and either added to (U=1) or subtracted from (U=0) the base register (Rn); this calculation may be performed either before (P=1) or after (P=0) the base is used as the transfer address. The modified base value may be overwritten back into the base register (if W=1), or the old value of the base may be preserved (W=0). Note that post-indexed addressing modes require explicit setting of the W bit, unlike LDR and STR which always write-back when post-indexed.

The value of the base register, modified by the offset in a pre-indexed instruction, is used as the address for the transfer of the first word. The second word (if more than one is transferred) will go to or come from an address one word (4 bytes) higher than the first transfer, and the address will be incremented by one word for each subsequent transfer.

ADDRESS ALIGNMENT

The base address should normally be a word aligned quantity. The bottom 2 bits of the address will appear on **A[1:0]** and might be interpreted by the memory system.

USE OF R15

If Rn is R15, the value used will be the address of the instruction plus 8 bytes. Base write-back to R15 must not be specified.

DATA ABORTS

If the address is legal but the memory manager generates an abort, the data trap will be taken. The write-back of the modified base will take place, but all other processor state will be preserved. The coprocessor is partly responsible for ensuring that the data transfer can be restarted after the cause of the abort has been resolved, and must ensure that any subsequent actions it undertakes can be repeated when the instruction is retried.

INSTRUCTION CYCLE TIMES

Coprocessor data transfer instructions take $(n-1)S + 2N + bI$ incremental cycles to execute, where:

n The number of words transferred.

b The number of cycles spent in the coprocessor busy-wait loop.

S, N and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively.

ASSEMBLER SYNTAX

<LDC|STC>{cond}{L} p#,cd,<Address>

LDC	Load from memory to coprocessor
STC	Store from coprocessor to memory
{L}	When present perform long transfer (N=1), otherwise perform short transfer (N=0)
{cond}	Two character condition mnemonic. See Table 3-2..
p#	The unique number of the required coprocessor
cd	An expression evaluating to a valid coprocessor register number that is placed in the CRd field

<Address> can be:

- 1 An expression which generates an address:
The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated
- 2 A pre-indexed addressing specification:

[Rn]	offset of zero
[Rn,<#expression>]{!}	offset of <expression> bytes
- 3 A post-indexed addressing specification:

Rn,<#expression	offset of <expression> bytes
{!}	write back the base register (set the W bit) if ! is present
Rn	is an expression evaluating to a valid ARM7TDMI register number.

NOTE

If Rn is R15, the assembler will subtract 8 from the offset value to allow for ARM7TDMI pipelining.

EXAMPLES

LDC	p1,c2,table	; Load c2 of coproc 1 from address
		; table, using a PC relative address.
STCEQL	p2,c3,[R5,#24]!	; Conditionally store c3 of coproc 2
		; into an address 24 bytes up from R5,
		; write this address back to R5, and use
		; long transfer option (probably to store multiple words).

NOTE

Although the address offset is expressed in bytes, the instruction offset field is in words. The assembler will adjust the offset appropriately.

COPROCESSOR REGISTER TRANSFERS (MRC, MCR)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2.. The instruction encoding is shown in Figure 3-27.

This class of instruction is used to communicate information directly between ARM7TDMI and a coprocessor. An example of a coprocessor to ARM7TDMI register transfer (MRC) instruction would be a FIX of a floating point value held in a coprocessor, where the floating point number is converted into a 32 bit integer within the coprocessor, and the result is then transferred to ARM7TDMI register. A FLOAT of a 32 bit value in ARM7TDMI register into a floating point value within the coprocessor illustrates the use of ARM7TDMI register to coprocessor transfer (MCR).

An important use of this instruction is to communicate control information directly from the coprocessor into the ARM7TDMI CPSR flags. As an example, the result of a comparison of two floating point values within a coprocessor can be moved to the CPSR to control the subsequent flow of execution.

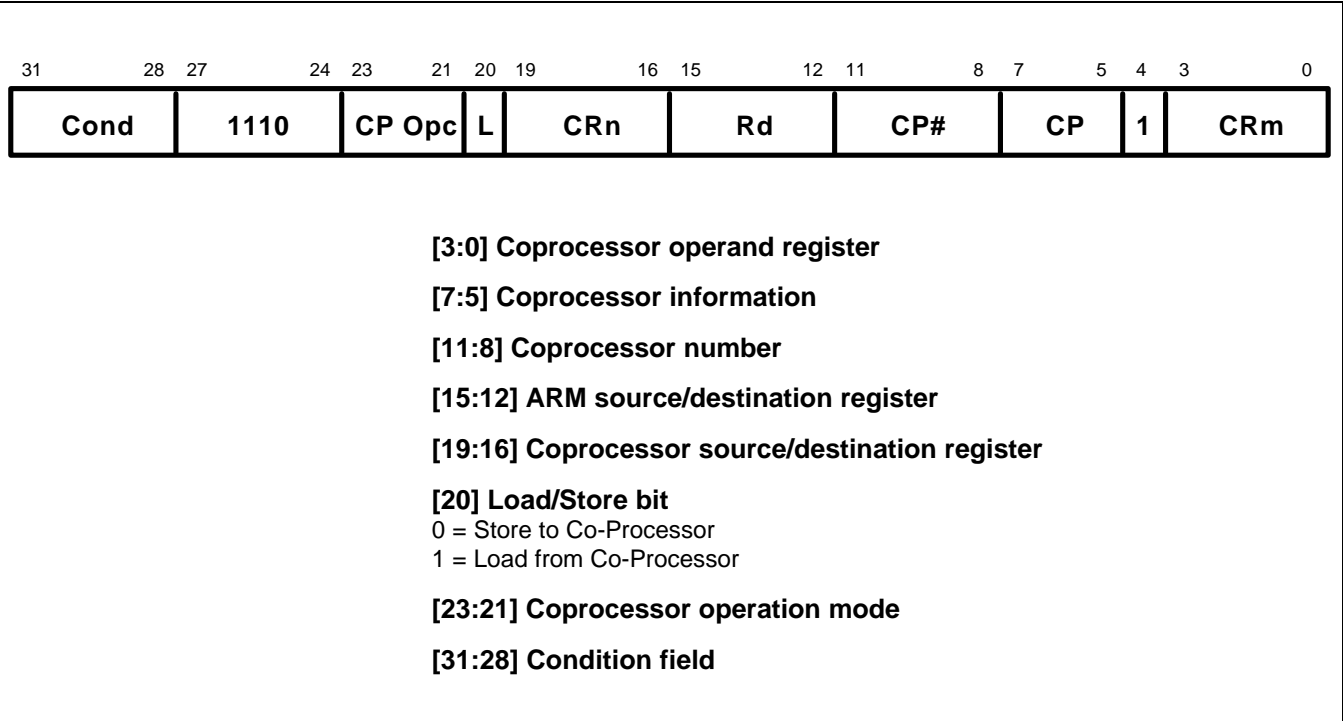


Figure 3-27. Coprocesspr Register Transfer Instructions

THE COPROCESSOR FIELDS

The CP# field is used, as for all coprocessor instructions, to specify which coprocessor is being called upon.

The CP Opc, CRn, CP and CRm fields are used only by the coprocessor, and the interpretation presented here is derived from convention only. Other interpretations are allowed where the coprocessor functionality is incompatible with this one. The conventional interpretation is that the CP Opc and CP fields specify the operation the coprocessor is required to perform, CRn is the coprocessor register which is the source or destination of the transferred information, and CRm is a second coprocessor register which may be involved in some way which depends on the particular operation specified.

TRANSFERS TO R15

When a coprocessor register transfer to ARM7TDMI has R15 as the destination, bits 31, 30, 29 and 28 of the transferred word are copied into the N, Z, C and V flags respectively. The other bits of the transferred word are ignored, and the PC and other CPSR bits are unaffected by the transfer.

TRANSFERS FROM R15

A coprocessor register transfer from ARM7TDMI with R15 as the source register will store the PC+12.

INSTRUCTION CYCLE TIMES

MRC instructions take $1S + (b+1)I + 1C$ incremental cycles to execute, where S, I and C are defined as sequential (S-cycle), internal (I-cycle), and coprocessor register transfer (C-cycle), respectively. MCR instructions take $1S + bI + 1C$ incremental cycles to execute, where b is the number of cycles spent in the coprocessor busy-wait loop.

ASSEMBLER SYNTAX

<MCR|MRC>{cond} p#,<expression1>,Rd,cn,cm{,<expression2>}

MRC	Move from coprocessor to ARM7TDMI register (L=1)
MCR	Move from ARM7TDMI register to coprocessor (L=0)
{cond}	Two character condition mnemonic. See Table 3-2
p#	The unique number of the required coprocessor
<expression1>	Evaluated to a constant and placed in the CP Opc field
Rd	An expression evaluating to a valid ARM7TDMI register number
cn and cm	Expressions evaluating to the valid coprocessor register numbers CRn and CRm respectively
<expression2>	Where present is evaluated to a constant and placed in the CP field

EXAMPLES

MRC	p2,5,R3,c5,c6	; Request coproc 2 to perform operation 5 ; on c5 and c6, and transfer the (single ; 32-bit word) result back to R3.
MCR	p6,0,R4,c5,c6	; Request coproc 6 to perform operation 0 ; on R4 and place the result in c6.
MRCEQ	p3,9,R3,c5,c6,2	; Conditionally request coproc 3 to ; perform operation 9 (type 2) on c5 and ; c6, and transfer the result back to R3.

UNDEFINED INSTRUCTION

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction format is shown in Figure 3-28.

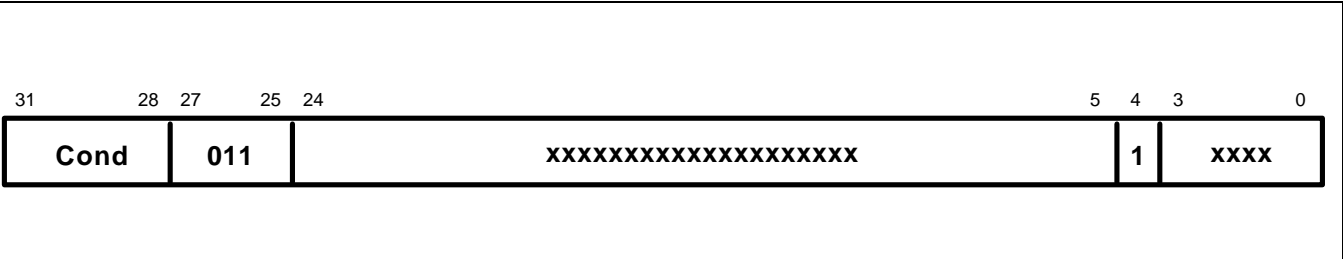


Figure 3-28. Undefined Instruction

If the condition is true, the undefined instruction trap will be taken.

Note that the undefined instruction mechanism involves offering this instruction to any coprocessors which may be present, and all coprocessors must refuse to accept it by driving **CPA** and **CPB** HIGH.

INSTRUCTION CYCLE TIMES

This instruction takes 2S + 1I + 1N cycles, where S, N and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle).

ASSEMBLER SYNTAX

The assembler has no mnemonics for generating this instruction. If it is adopted in the future for some specified use, suitable mnemonics will be added to the assembler. Until such time, this instruction must not be used.

INSTRUCTION SET EXAMPLES

The following examples show ways in which the basic ARM7TDMI instructions can combine to give efficient code. None of these methods saves a great deal of execution time (although they may save some), mostly they just save code.

USING THE CONDITIONAL INSTRUCTIONS

Using Conditionals for Logical OR

```

CMP      Rn,#p           ; If Rn=p OR Rm=q THEN GOTO Label.
BEQ      Label
CMP      Rm,#q
BEQ      Label

```

This can be replaced by

```

CMP      Rn,#p
CMPNE    Rm,#q           ; If condition not satisfied try other test.
BEQ      Label

```

Absolute Value

```

TEQ      Rn,#0           ; Test sign
RSBMI    Rn,Rn,#0       ; and 2's complement if necessary.

```

Multiplication by 4, 5 or 6 (Run Time)

```

MOV      Rc,Ra,LSL#2     ; Multiply by 4,
CMP      Rb,#5           ; Test value,
ADDCS    Rc,Rc,Ra        ; Complete multiply by 5,
ADDHI    Rc,Rc,Ra        ; Complete multiply by 6.

```

Combining Discrete and Range Tests

```

TEQ      Rc,#127         ; Discrete test,
CMPNE    Rc,#"-1        ; Range test
MOVLS    Rc,#"."         ; IF Rc<=" " OR Rc=ASCII(127)
                        ; THEN Rc:="."

```

Division and Remainder

A number of divide routines for specific applications are provided in source form as part of the ANSI C library provided with the ARM Cross Development Toolkit, available from your supplier. A short general purpose divide routine follows.

			; Enter with numbers in Ra and Rb.
	MOV	Rcnt,#1	; Bit to control the division.
Div1	CMP	Rb,#0x80000000	; Move Rb until greater than Ra.
	CMPCC	Rb,Ra	
	MOVCC	Rb,Rb,ASL#1	
	MOVCC	Rcnt,Rcnt,ASL#1	
	BCC	Div1	
	MOV	Rc,#0	
Div2	CMP	Ra,Rb	; Test for possible subtraction.
	SUBCS	Ra,Ra,Rb	; Subtract if ok,
	ADDCS	Rc,Rc,Rcnt	; Put relevant bit into result
	MOVS	Rcnt,Rcnt,LSR#1	; Shift control bit
	MOVNE	Rb,Rb,LSR#1	; Halve unless finished.
	BNE	Div2	; Divide result in Rc, remainder in Ra.

Overflow Eetection in the ARM7TDMI

1. Overflow in unsigned multiply with a 32-bit result

UMULL	Rd,Rt,Rm,Rn	; 3 to 6 cycles
TEQ	Rt,#0	; +1 cycle and a register
BNE	overflow	

2. Overflow in signed multiply with a 32-bit result

SMULL	Rd,Rt,Rm,Rn	; 3 to 6 cycles
TEQ	Rt,Rd ASR#31	; +1 cycle and a register
BNE	overflow	

3. Overflow in unsigned multiply accumulate with a 32 bit result

UMLAL	Rd,Rt,Rm,Rn	; 4 to 7 cycles
TEQ	Rt,#0	; +1 cycle and a register
BNE	overflow	

4. Overflow in signed multiply accumulate with a 32 bit result

SMLAL	Rd,Rt,Rm,Rn	; 4 to 7 cycles
TEQ	Rt,Rd, ASR#31	; +1 cycle and a register
BNE	overflow	

5. Overflow in unsigned multiply accumulate with a 64 bit result

UMULL	RI,Rh,Rm,Rn	; 3 to 6 cycles
ADDS	RI,RI,Ra1	; Lower accumulate
ADC	Rh,Rh,Ra2	; Upper accumulate
BCS	overflow	; 1 cycle and 2 registers

6. Overflow in signed multiply accumulate with a 64 bit result

SMULL	RI,Rh,Rm,Rn	; 3 to 6 cycles
ADDS	RI,RI,Ra1	; Lower accumulate
ADC	Rh,Rh,Ra2	; Upper accumulate
BVS	overflow	; 1 cycle and 2 registers

NOTE

Overflow checking is not applicable to unsigned and signed multiplies with a 64-bit result, since overflow does not occur in such calculations.

PSEUDO-RANDOM BINARY SEQUENCE GENERATOR

It is often necessary to generate (pseudo-) random numbers and the most efficient algorithms are based on shift generators with exclusive-OR feedback rather like a cyclic redundancy check generator. Unfortunately the sequence of a 32 bit generator needs more than one feedback tap to be maximal length (i.e. $2^{32}-1$ cycles before repetition), so this example uses a 33 bit register with taps at bits 33 and 20. The basic algorithm is newbit:=bit 33 eor bit 20, shift left the 33 bit number and put in newbit at the bottom; this operation is performed for all the newbits needed (i.e. 32 bits). The entire operation can be done in 5 S cycles:

		; Enter with seed in Ra (32 bits),
		; Rb (1 bit in Rb lsb), uses Rc.
TST	Rb,Rb,LSR#1	; Top bit into carry
MOVS	Rc,Ra,RRX	; 33 bit rotate right
ADC	Rb,Rb,Rb	; Carry into lsb of Rb
EOR	Rc,Rc,Ra,LSL#12	; (involved!)
EOR	Ra,Rc,Rc,LSR#20	; (similarly involved!) new seed in Ra, Rb as before

MULTIPLICATION BY CONSTANT USING THE BARREL SHIFTER**Multiplication by 2^n (1,2,4,8,16,32..)**

MOV Ra, Rb, LSL #n

Multiplication by 2^{n+1} (3,5,9,17..)

ADD Ra,Ra,Ra,LSL #n

Multiplication by 2^{n-1} (3,7,15..)

RSB Ra,Ra,Ra,LSL #n

Multiplication by 6

```

ADD    Ra,Ra,Ra,LSL #1    ; Multiply by 3
MOV    Ra,Ra,LSL#1        ; and then by 2

```

Multiply by 10 and add in extra number

```

ADD    Ra,Ra,Ra,LSL#2    ; Multiply by 5
ADD    Ra,Rc,Ra,LSL#1    ; Multiply by 2 and add in next digit

```

General recursive method for $R_b := R_a * C$, C a constant:

1. If C even, say $C = 2^n * D$, D odd:

```

D=1:    MOV  Rb,Ra,LSL #n
D<>1:   {Rb := Ra*D}
MOV     Rb,Rb,LSL #n

```

2. If $C \bmod 4 = 1$, say $C = 2^n * D + 1$, D odd, $n > 1$:

```

D=1:    ADD  Rb,Ra,Ra,LSL #n
D<>1:   {Rb := Ra*D}
ADD     Rb,Ra,Rb,LSL #n

```

3. If $C \bmod 4 = 3$, say $C = 2^n * D - 1$, D odd, $n > 1$:

```

D=1:    RSB  Rb,Ra,Ra,LSL #n
D<>1:   {Rb := Ra*D}
RSB     Rb,Ra,Rb,LSL #n

```

This is not quite optimal, but close. An example of its non-optimality is multiply by 45 which is done by:

```

RSB     Rb,Ra,Ra,LSL#2    ; Multiply by 3
RSB     Rb,Ra,Rb,LSL#2    ; Multiply by  $4*3-1 = 11$ 
ADD     Rb,Ra,Rb,LSL# 2   ; Multiply by  $4*11+1 = 45$ 

```

rather than by:

```

ADD     Rb,Ra,Ra,LSL#3    ; Multiply by 9
ADD     Rb,Rb,Rb,LSL#2    ; Multiply by  $5*9 = 45$ 

```

LOADING A WORD FROM AN UNKNOWN ALIGNMENT

		; Enter with address in Ra (32 bits) uses
		; Rb, Rc result in Rd. Note d must be less than c e.g. 0,1
BIC	Rb,Ra,#3	; Get word aligned address
LDMIA	Rb,{Rd,Rc}	; Get 64 bits containing answer
AND	Rb,Ra,#3	; Correction factor in bytes
MOVS	Rb,Rb,LSL#3	; ...now in bits and test if aligned
MOVNE	Rd,Rd,LSR Rb	; Produce bottom of result word (if not aligned)
RSBNE	Rb,Rb,#32	; Get other shift amount
ORRNE	Rd,Rd,Rc,LSL Rb	; Combine two halves to get result

NOTES

THUMB INSTRUCTION SET FORMAT

The thumb instruction sets are 16-bit versions of ARM instruction sets (32-bit format). The ARM instructions are reduced to 16-bit versions, Thumb instructions, at the cost of versatile functions of the ARM instruction sets. The thumb instructions are decompressed to the ARM instructions by the Thumb decompressor inside the ARM7TDMI core.

As the Thumb instructions are compressed ARM instructions, the Thumb instructions have the 16-bit format instructions and have some restrictions. The restrictions by 16-bit format is fully notified for using the Thumb instructions.

FORMAT SUMMARY

The THUMB instruction set formats are shown in the following figure.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
1	0	0	0	Op		Offset5					Rs		Rd			<i>Move shifted register</i>		
2	0	0	0	1	1	I	Op	Rn/offset3			Rs		Rd			<i>Add/subtract</i>		
3	0	0	1	Op		Rd			Offset8								<i>Move/compare/add /subtract immediate</i>	
4	0	1	0	0	0	0	Op			Rs		Rd			<i>ALU operations</i>			
5	0	1	0	0	0	1	Op		H1	H2	Rs/Hs		Rd/Hd			<i>Hi register operations /branch exchange</i>		
6	0	1	0	0	1	Rd			Word8								<i>PC-relative load</i>	
7	0	1	0	1	L	B	0	Ro			Rb		Rd			<i>Load/store with register offset</i>		
8	0	1	0	1	H	S	1	Ro			Rb		Rd			<i>Load/store sign-extended byte/halfword</i>		
9	0	1	1	B	L	Offset5					Rb		Rd			<i>Load/store with immediate offset</i>		
10	1	0	0	0	L	Offset5					Rb		Rd			<i>Load/store halfword</i>		
11	1	0	0	1	L	Rd			Word8								<i>SP-relative load/store</i>	
12	1	0	1	0	SP	Rd			Word8								<i>Load address</i>	
13	1	0	1	1	0	0	0	0	S	SWord7								<i>Add offset to stack pointer</i>
14	1	0	1	1	L	1	0	R	Rlist								<i>Push/pop registers</i>	
15	1	1	0	0	L	Rb			Rlist								<i>Multiple load/store</i>	
16	1	1	0	1	Cond					Soffset8								<i>Conditional branch</i>
17	1	1	0	1	1	1	1	1	Value8								<i>Software Interrupt</i>	
18	1	1	1	0	0	Offset11											<i>Unconditional branch</i>	
19	1	1	1	1	H	Offset											<i>Long branch with link</i>	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		

Figure 3-29. THUMB Instruction Set Formats

OPCODE SUMMARY

The following table summarizes the THUMB instruction set. For further information about a particular instruction please refer to the sections listed in the right-most column.

Table 3-7. THUMB Instruction Set Opcodes

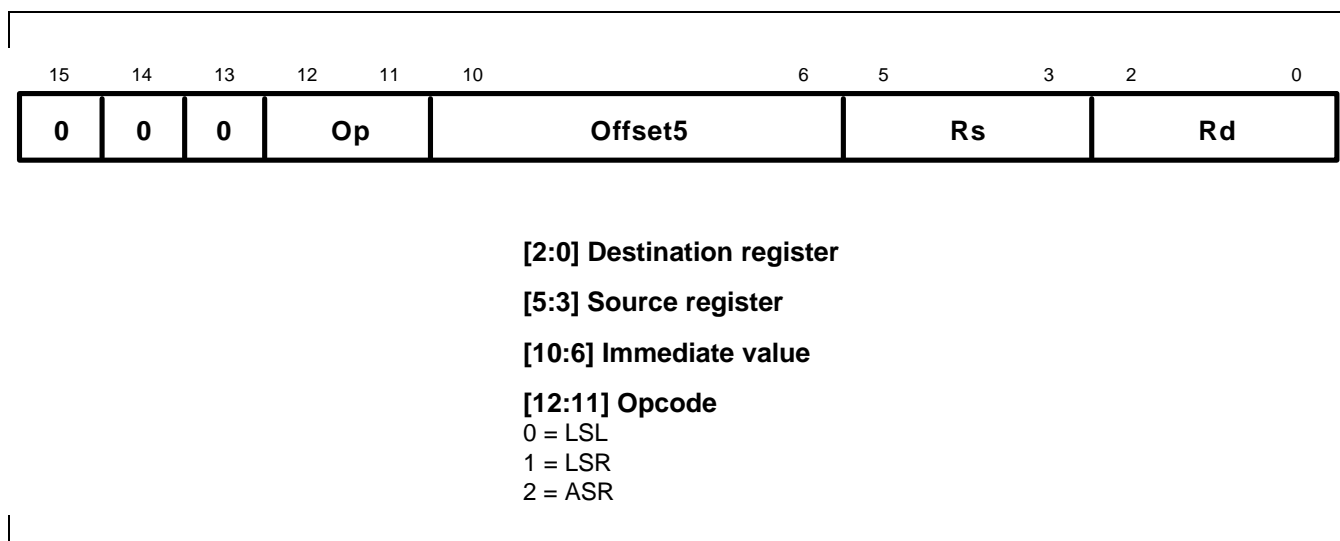
Mnemonic	Instruction	Lo-Register Operand	Hi-Register Operand	Condition Codes Set
ADC	Add with Carry	4	—	4
ADD	Add	4	4	4 ⁽¹⁾
AND	AND	4	—	4
ASR	Arithmetic Shift Right	4	—	4
B	Unconditional branch	4	—	—
Bxx	Conditional branch	4	—	—
BIC	Bit Clear	4	—	4
BL	Branch and Link	—	—	—
BX	Branch and Exchange	4	4	—
CMN	Compare Negative	4	—	4
CMP	Compare	4	4	4
EOR	EOR	4	—	4
LDMIA	Load multiple	4	—	—
LDR	Load word	4	—	—
LDRB	Load byte	4	—	—
LDRH	Load halfword	4	—	—
LSL	Logical Shift Left	4	—	4
LDSB	Load sign-extended byte	4	—	—
LDSH	Load sign-extended halfword	4	—	—
LSR	Logical Shift Right	4	—	4
MOV	Move register	4	4	4 ⁽²⁾
MUL	Multiply	4	—	4
MVN	Move Negative register	4	—	4
NEG	Negate	4	—	4
ORR	OR	4	—	4

Table 3-7. THUMB Instruction Set Opcodes (Continued)

Mnemonic	Instruction	Lo-Register Operand	Hi-Register Operand	Condition Codes Set
POP	Pop registers	4	—	—
PUSH	Push registers	4	—	—
ROR	Rotate Right	4	—	4
SBC	Subtract with Carry	4	—	4
STMIA	Store Multiple	4	—	—
STR	Store word	4	—	—
STRB	Store byte	4	—	—
STRH	Store halfword	4	—	—
SWI	Software Interrupt	—	—	—
SUB	Subtract	4	—	4
TST	Test bits	4	—	4

NOTES

1. The condition codes are unaffected by the format 5, 12 and 13 versions of this instruction.
2. The condition codes are unaffected by the format 5 version of this instruction.

FORMAT 1: MOVE SHIFTED REGISTER**Figure 3-30. Format 1****OPERATION**

These instructions move a shifted value between Lo registers. The THUMB assembler syntax is shown in Table 3-8.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-8. Summary of Format 1 Instructions

OP	THUMB assembler	ARM equivalent	Action
00	LSL Rd, Rs, #Offset5	MOVS Rd, Rs, LSL #Offset5	Shift Rs left by a 5-bit immediate value and store the result in Rd.
01	LSR Rd, Rs, #Offset5	MOVS Rd, Rs, LSR #Offset5	Perform logical shift right on Rs by a 5-bit immediate value and store the result in Rd.
10	ASR Rd, Rs, #Offset5	MOVS Rd, Rs, ASR #Offset5	Perform arithmetic shift right on Rs by a 5-bit immediate value and store the result in Rd.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-8. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

LSR	R2, R5, #27	; Logical shift right the contents
		; of R5 by 27 and store the result in R2.
		; Set condition codes on the result.

FORMAT 2: ADD/SUBTRACT

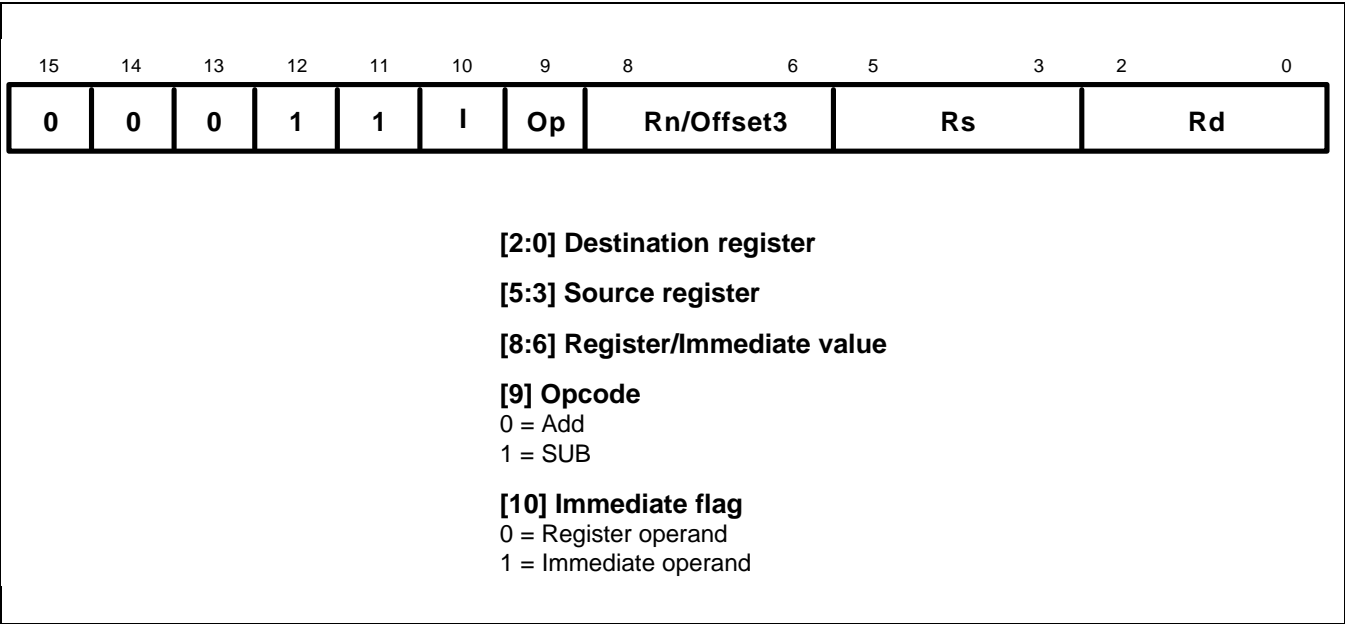


Figure 3-31. Format 2

OPERATION

These instructions allow the contents of a Lo register or a 3-bit immediate value to be added to or subtracted from a Lo register. The THUMB assembler syntax is shown in Table 3-9.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-9. Summary of Format 2 Instructions

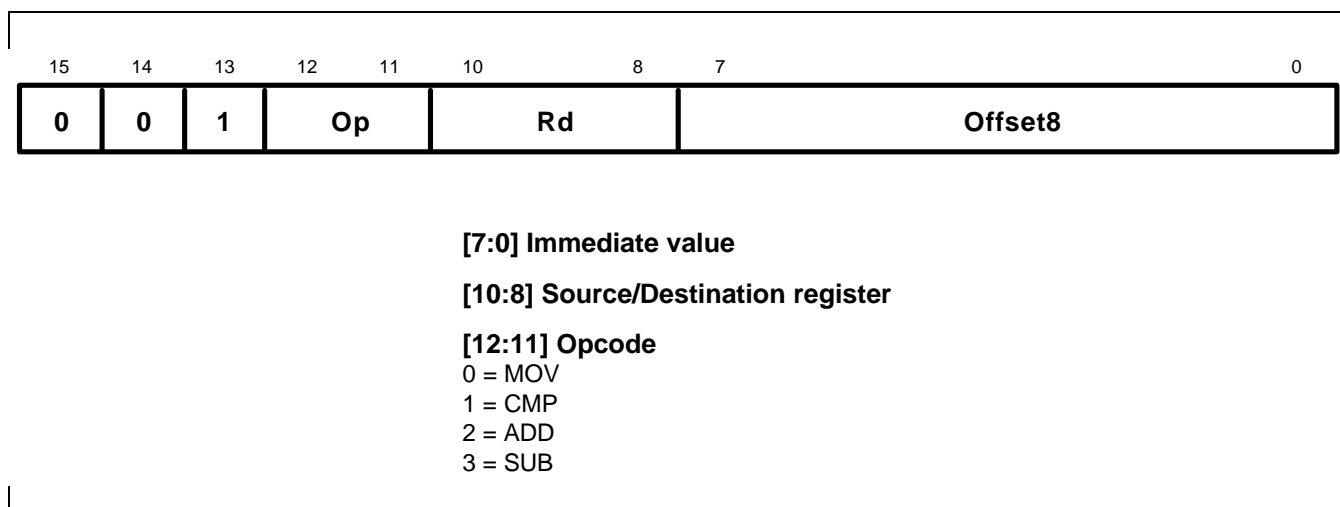
Op	I	THUMB assembler	ARM equivalent	Action
0	0	ADD Rd, Rs, Rn	ADDS Rd, Rs, Rn	Add contents of Rn to contents of Rs. Place result in Rd.
0	1	ADD Rd, Rs, #Offset3	ADDS Rd, Rs, #Offset3	Add 3-bit immediate value to contents of Rs. Place result in Rd.
1	0	SUB Rd, Rs, Rn	SUBS Rd, Rs, Rn	Subtract contents of Rn from contents of Rs. Place result in Rd.
1	1	SUB Rd, Rs, #Offset3	SUBS Rd, Rs, #Offset3	Subtract 3-bit immediate value from contents of Rs. Place result in Rd.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-9. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

ADD	R0, R3, R4	; R0 := R3 + R4 and set condition codes on the result.
SUB	R6, R2, #6	; R6 := R2 – 6 and set condition codes.

FORMAT 3: MOVE/COMPARE/ADD/SUBTRACT IMMEDIATE**Figure 3-32. Format 3****OPERATIONS**

The instructions in this group perform operations between a Lo register and an 8-bit immediate value. The THUMB assembler syntax is shown in Table 3-10.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-10. Summary of Format 3 Instructions

Op	THUMB assembler	ARM equivalent	Action
00	MOV Rd, #Offset8	MOVS Rd, #Offset8	Move 8-bit immediate value into Rd.
01	CMP Rd, #Offset8	CMP Rd, #Offset8	Compare contents of Rd with 8-bit immediate value.
10	ADD Rd, #Offset8	ADDS Rd, Rd, #Offset8	Add 8-bit immediate value to contents of Rd and place the result in Rd.
11	SUB Rd, #Offset8	SUBS Rd, Rd, #Offset8	Subtract 8-bit immediate value from contents of Rd and place the result in Rd.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-10. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

MOV	R0, #128	; R0 := 128 and set condition codes
CMP	R2, #62	; Set condition codes on R2 – 62
ADD	R1, #255	; R1 := R1 + 255 and set condition codes
SUB	R6, #145	; R6 := R6 – 145 and set condition codes

FORMAT 4: ALU OPERATIONS

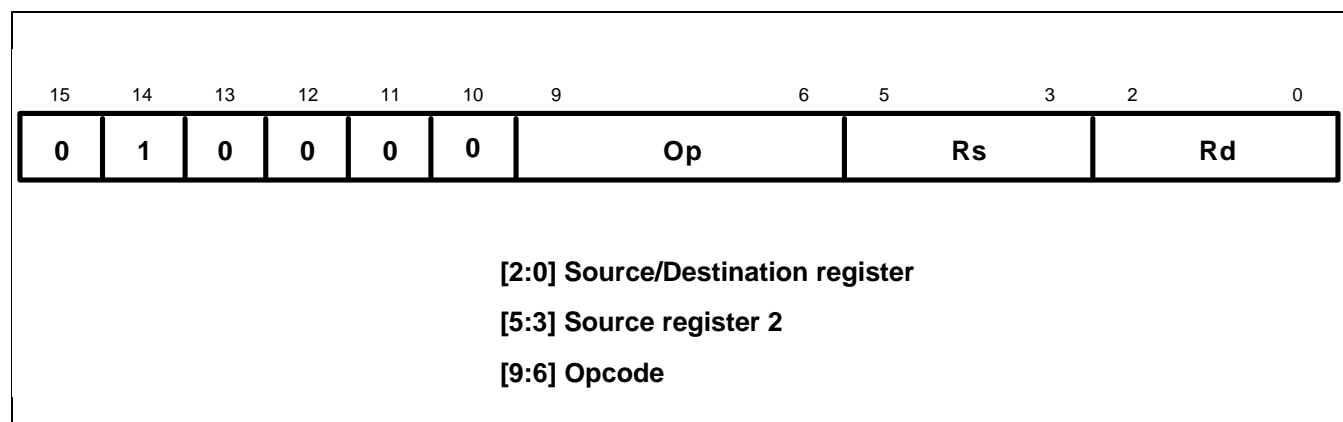


Figure 3-33. Format 4

OPERATION

The following instructions perform ALU operations on a Lo register pair.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-11. Summary of Format 4 Instructions

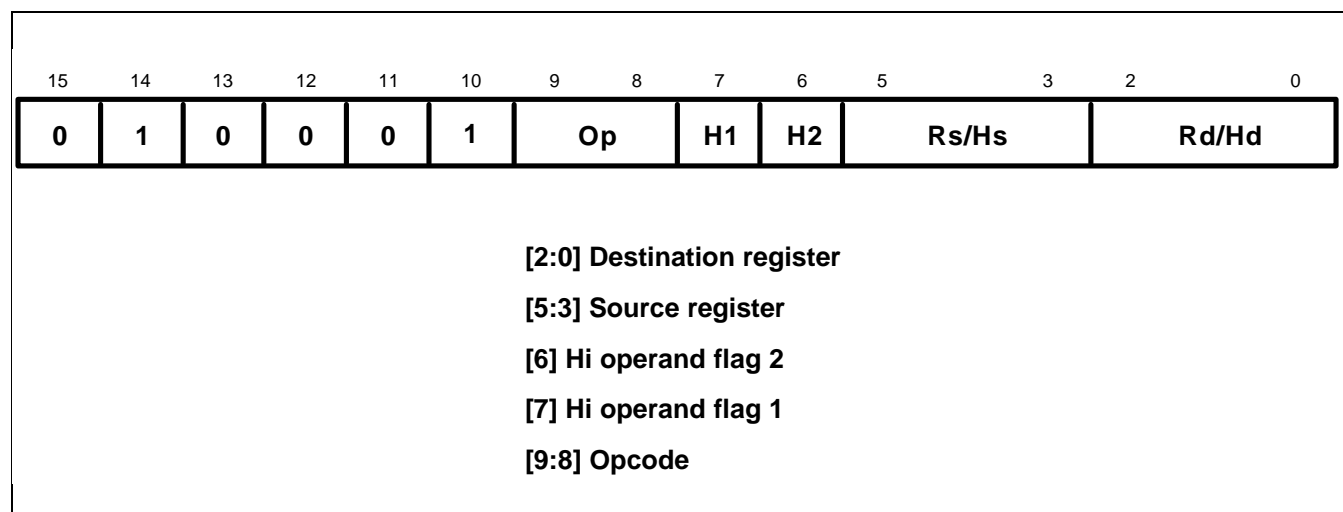
OP	THUMB assembler	ARM equivalent	Action
0000	AND Rd, Rs	ANDS Rd, Rd, Rs	Rd := Rd AND Rs
0001	EOR Rd, Rs	EORS Rd, Rd, Rs	Rd := Rd EOR Rs
0010	LSL Rd, Rs	MOVS Rd, Rd, LSL Rs	Rd := Rd << Rs
0011	LSR Rd, Rs	MOVS Rd, Rd, LSR Rs	Rd := Rd >> Rs
0100	ASR Rd, Rs	MOVS Rd, Rd, ASR Rs	Rd := Rd ASR Rs
0101	ADC Rd, Rs	ADCS Rd, Rd, Rs	Rd := Rd + Rs + C-bit
0110	SBC Rd, Rs	SBCS Rd, Rd, Rs	Rd := Rd – Rs – NOT C-bit
0111	ROR Rd, Rs	MOVS Rd, Rd, ROR Rs	Rd := Rd ROR Rs
1000	TST Rd, Rs	TST Rd, Rs	Set condition codes on Rd AND Rs
1001	NEG Rd, Rs	RSBS Rd, Rs, #0	Rd = – Rs
1010	CMP Rd, Rs	CMP Rd, Rs	Set condition codes on Rd – Rs
1011	CMN Rd, Rs	CMN Rd, Rs	Set condition codes on Rd + Rs
1100	ORR Rd, Rs	ORRS Rd, Rd, Rs	Rd := Rd OR Rs
1101	MUL Rd, Rs	MULS Rd, Rs, Rd	Rd := Rs * Rd
1110	BIC Rd, Rs	BICS Rd, Rd, Rs	Rd := Rd AND NOT Rs
1111	MVN Rd, Rs	MVNS Rd, Rs	Rd := NOT Rs

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-11. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

EOR	R3, R4	; R3 := R3 EOR R4 and set condition codes
ROR	R1, R0	; Rotate Right R1 by the value in R0, store ; the result in R1 and set condition codes
NEG	R5, R3	; Subtract the contents of R3 from zero, ; store the result in R5. Set condition codes ie R5 = – R3
CMP	R2, R6	; Set the condition codes on the result of R2 – R6
MUL	R0, R7	; R0 := R7 * R0 and set condition codes

FORMAT 5: HI-REGISTER OPERATIONS/BRANCH EXCHANGE**Figure 3-34. Format 5****OPERATION**

There are four sets of instructions in this group. The first three allow ADD, CMP and MOV operations to be performed between Lo and Hi registers, or a pair of Hi registers. The fourth, BX, allows a Branch to be performed which may also be used to switch processor state. The THUMB assembler syntax is shown in Table 3-12.

NOTE

In this group only CMP (Op = 01) sets the CPSR condition codes.

The action of H1= 0, H2 = 0 for Op = 00 (ADD), Op = 01 (CMP) and Op = 10 (MOV) is undefined, and should not be used.

Table 3-12. Summary of Format 5 Instructions

Op	H1	H2	THUMB assembler	ARM equivalent	Action
00	0	1	ADD Rd, Hs	ADD Rd, Rd, Hs	Add a register in the range 8-15 to a register in the range 0-7.
00	1	0	ADD Hd, Rs	ADD Hd, Hd, Rs	Add a register in the range 0-7 to a register in the range 8-15.
00	1	1	ADD Hd, Hs	ADD Hd, Hd, Hs	Add two registers in the range 8-15
01	0	1	CMP Rd, Hs	CMP Rd, Hs	Compare a register in the range 0-7 with a register in the range 8-15. Set the condition code flags on the result.
01	1	0	CMP Hd, Rs	CMP Hd, Rs	Compare a register in the range 8-15 with a register in the range 0-7. Set the condition code flags on the result.

Table 3-12. Summary of Format 5 Instructions (Continued)

Op	H1	H2	THUMB assembler	ARM equivalent	Action
01	1	1	CMP Hd, Hs	CMP Hd, Hs	Compare two registers in the range 8-15. Set the condition code flags on the result.
10	0	1	MOV Rd, Hs	MOV Rd, Hs	Move a value from a register in the range 8-15 to a register in the range 0-7.
10	1	0	MOV Hd, Rs	MOV Hd, Rs	Move a value from a register in the range 0-7 to a register in the range 8-15.
10	1	1	MOV Hd, Hs	MOV Hd, Hs	Move a value between two registers in the range 8-15.
11	0	0	BX Rs	BX Rs	Perform branch (plus optional state change) to address in a register in the range 0-7.
11	0	1	BX Hs	BX Hs	Perform branch (plus optional state change) to address in a register in the range 8-15.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-12. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

THE BX INSTRUCTION

BX performs a Branch to a routine whose start address is specified in a Lo or Hi register.

Bit 0 of the address determines the processor state on entry to the routine:

- Bit 0 = 0 Causes the processor to enter ARM state.
- Bit 0 = 1 Causes the processor to enter THUMB state.

NOTE

The action of H1 = 1 for this instruction is undefined, and should not be used.

EXAMPLES**Hi-Register Operations**

ADD	PC, R5	; PC := PC + R5 but don't set the condition codes.
CMP	R4, R12	; Set the condition codes on the result of R4 - R12.
MOV	R15, R14	; Move R14 (LR) into R15 (PC)
		; but don't set the condition codes,
		; eg. return from subroutine.

Branch and Exchange

		; Switch from THUMB to ARM state.
ADR	R1,outofTHUMB	; Load address of outofTHUMB into R1.
MOV	R11,R1	
BX	R11	; Transfer the contents of R11 into the PC.
		; Bit 0 of R11 determines whether
		; ARM or THUMB state is entered, ie. ARM state here.
...		
ALIGN		
CODE32		
outofTHUMB		
		; Now processing ARM instructions...

USING R15 AS AN OPERAND

If R15 is used as an operand, the value will be the address of the instruction + 4 with bit 0 cleared. Executing a BX PC in THUMB state from a non-word aligned address will result in unpredictable execution.

FORMAT 6: PC-RELATIVE LOAD

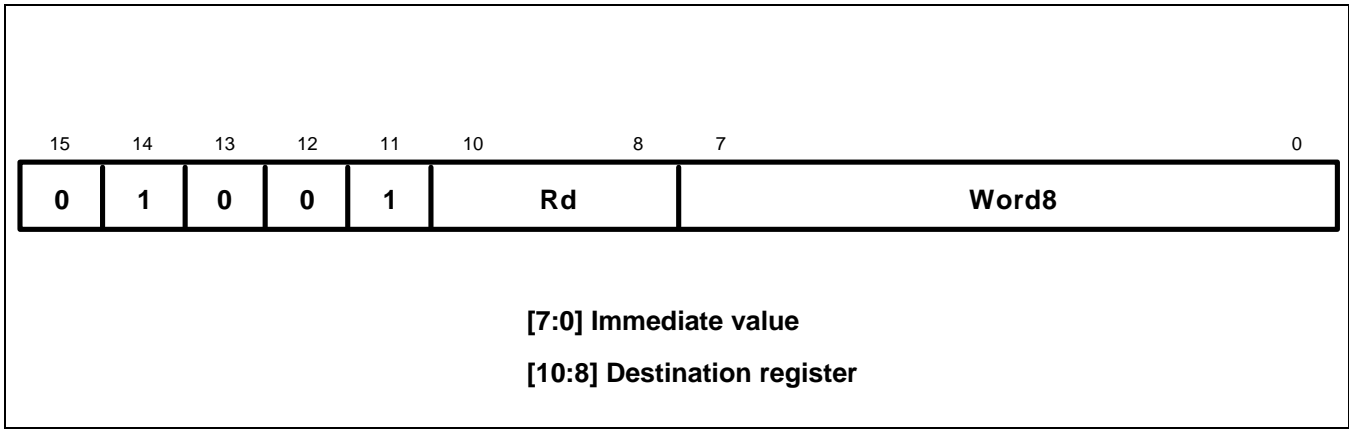


Figure 3-35. Format 6

OPERATION

This instruction loads a word from an address specified as a 10-bit immediate offset from the PC. The THUMB assembler syntax is shown below.

Table 3-13. Summary of PC-Relative Load Instruction

THUMB assembler	ARM equivalent	Action
LDR Rd, [PC, #Imm]	LDR Rd, [R15, #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the PC. Load the word from the resulting address into Rd.

NOTE: The value specified by #Imm is a full 10-bit address, but must always be word-aligned (ie with bits 1:0 set to 0), since the assembler places #Imm >> 2 in field Word 8. The value of the PC will be 4 bytes greater than the address of this instruction, but bit 1 of the PC is forced to 0 to ensure it is word aligned.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

LDR R3,[PC,#844]	;	Load into R3 the word found at the
	;	address formed by adding 844 to PC.
	;	bit[1] of PC is forced to zero.
	;	Note that the THUMB opcode will contain
	;	211 as the Word8 value.

FORMAT 7: LOAD/STORE WITH REGISTER OFFSET

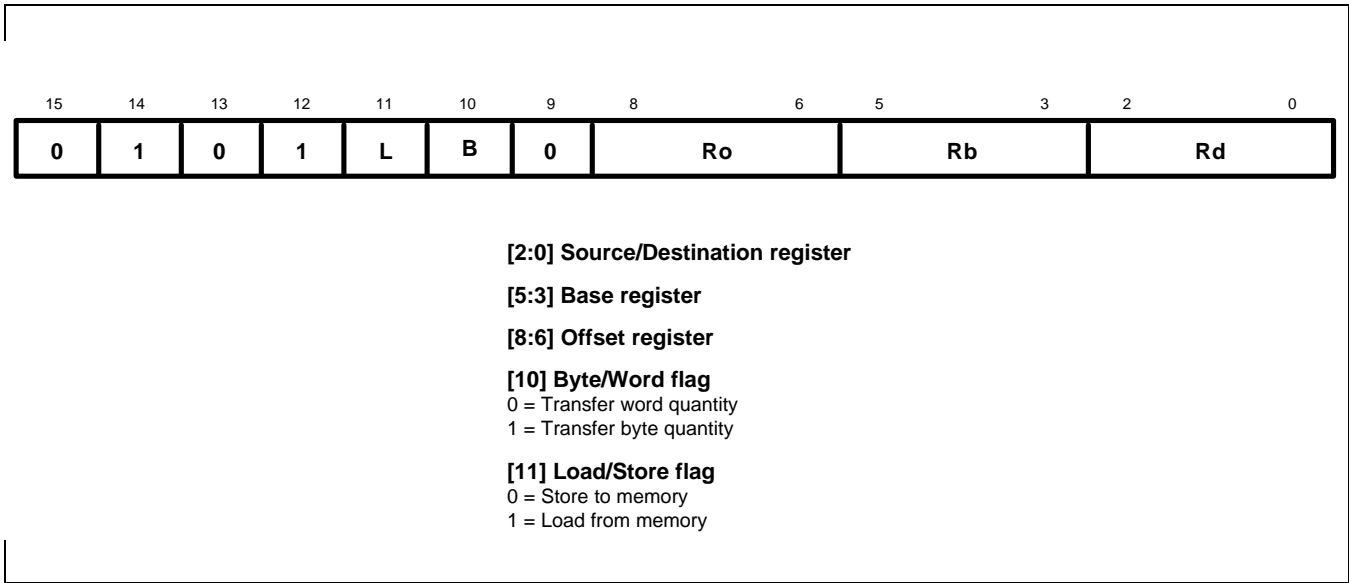


Figure 3-36. Format 7

OPERATION

These instructions transfer byte or word values between registers and memory. Memory addresses are pre-indexed using an offset register in the range 0-7. The THUMB assembler syntax is shown in Table 3-14.

Table 3-14. Summary of Format 7 Instructions

L	B	THUMB assembler	ARM equivalent	Action
0	0	STR Rd, [Rb, Ro]	STR Rd, [Rb, Ro]	Pre-indexed word store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the contents of Rd at the address.
0	1	STRB Rd, [Rb, Ro]	STRB Rd, [Rb, Ro]	Pre-indexed byte store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the byte value in Rd at the resulting address.
1	0	LDR Rd, [Rb, Ro]	LDR Rd, [Rb, Ro]	Pre-indexed word load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the contents of the address into Rd.

Table 3-14. Summary of Format 7 Instructions (Continued)

L	B	THUMB assembler	ARM equivalent	Action
1	1	LDRB Rd, [Rb, Ro]	LDRB Rd, [Rb, Ro]	Pre-indexed byte load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the byte value at the resulting address.

INSTRUCTION CYCLE TIMES

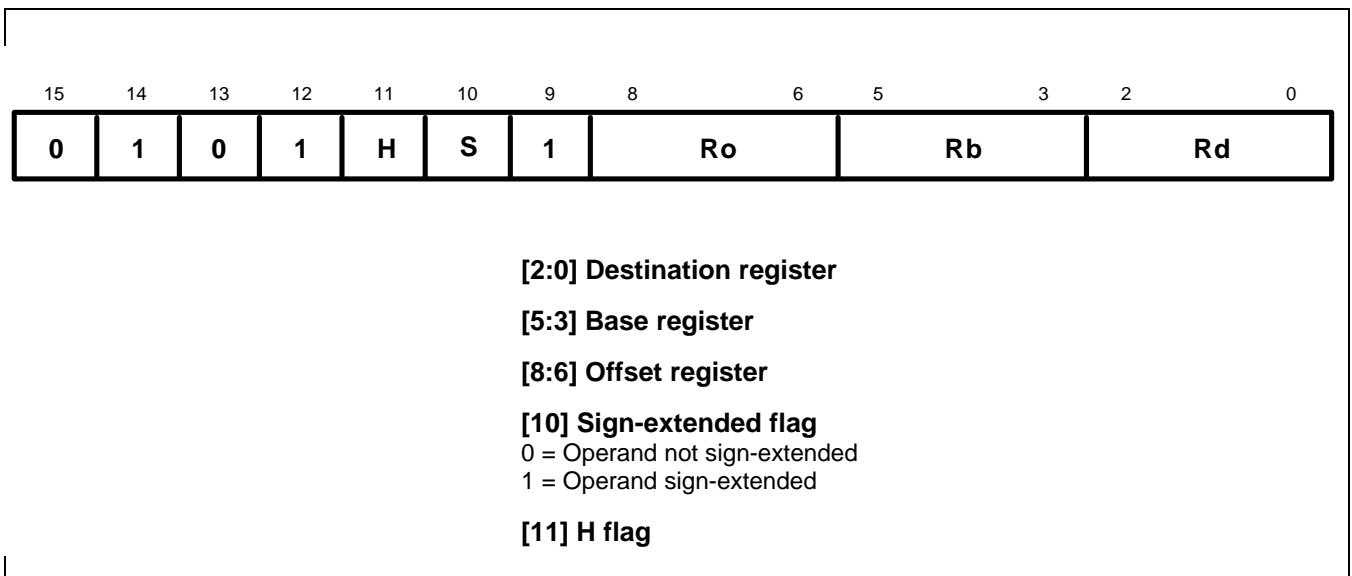
All instructions in this format have an equivalent ARM instruction as shown in Table 3-14. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```

STR      R3, [R2,R6]      ; Store word in R3 at the address
                        ; formed by adding R6 to R2.
LDRB     R2, [R0,R7]      ; Load into R2 the byte found at
                        ; the address formed by adding R7 to R0.

```

FORMAT 8: LOAD/STORE SIGN-EXTENDED BYTE/HALFWORD**Figure 3-37. Format 8****OPERATION**

These instructions load optionally sign-extended bytes or halfwords, and store halfwords. The THUMB assembler syntax is shown below.

Table 3-15. Summary of format 8 instructions

S	H	THUMB assembler	ARM equivalent	Action
0	0	STRH Rd, [Rb, Ro]	STRH Rd, [Rb, Ro]	Store halfword: Add Ro to base address in Rb. Store bits 0-15 of Rd at the resulting address.
0	1	LDRH Rd, [Rb, Ro]	LDRH Rd, [Rb, Ro]	Load halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to 0.
1	0	LDSB Rd, [Rb, Ro]	LDRSB Rd, [Rb, Ro]	Load sign-extended byte: Add Ro to base address in Rb. Load bits 0-7 of Rd from the resulting address, and set bits 8-31 of Rd to bit 7.
1	1	LDSH Rd, [Rb, Ro]	LDRSH Rd, [Rb, Ro]	Load sign-extended halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to bit 15.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-15. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

STRH	R4, [R3, R0]	; Store the lower 16 bits of R4 at the ; address formed by adding R0 to R3.
LDSB	R2, [R7, R1]	; Load into R2 the sign extended byte ; found at the address formed by adding R1 to R7.
LDSH	R3, [R4, R2]	; Load into R3 the sign extended halfword ; found at the address formed by adding R2 to R4.

FORMAT 9: LOAD/STORE WITH IMMEDIATE OFFSET

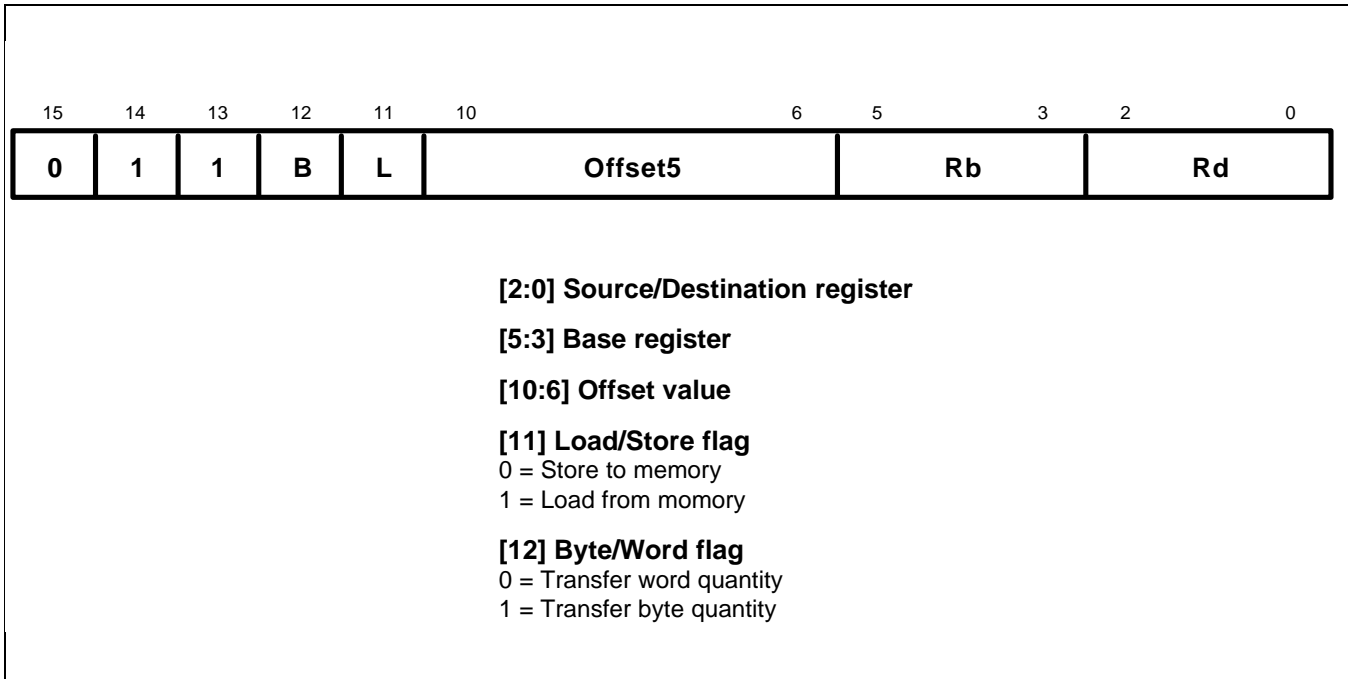


Figure 3-38. Format 9

OPERATION

These instructions transfer byte or word values between registers and memory using an immediate 5 or 7-bit offset. The THUMB assembler syntax is shown in Table 3-16.

Table 3-16. Summary of Format 9 Instructions

L	B	THUMB assembler	ARM equivalent	Action
0	0	STR Rd, [Rb, #Imm]	STR Rd, [Rb, #Imm]	Calculate the target address by adding together the value in Rb and Imm. Store the contents of Rd at the address.
1	0	LDR Rd, [Rb, #Imm]	LDR Rd, [Rb, #Imm]	Calculate the source address by adding together the value in Rb and Imm. Load Rd from the address.
0	1	STRB Rd, [Rb, #Imm]	STRB Rd, [Rb, #Imm]	Calculate the target address by adding together the value in Rb and Imm. Store the byte value in Rd at the address.
1	1	LDRB Rd, [Rb, #Imm]	LDRB Rd, [Rb, #Imm]	Calculate source address by adding together the value in Rb and Imm. Load the byte value at the address into Rd.

NOTE: For word accesses (B = 0), the value specified by #Imm is a full 7-bit address, but must be word-aligned (ie with bits 1:0 set to 0), since the assembler places #Imm >> 2 in the Offset5 field.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-16. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

LDR	R2, [R5,#116]	; Load into R2 the word found at the
		; address formed by adding 116 to R5.
		; Note that the THUMB opcode will
		; contain 29 as the Offset5 value.
STRB	R1, [R0,#13]	; Store the lower 8 bits of R1 at the
		; address formed by adding 13 to R0.
		; Note that the THUMB opcode will
		; contain 13 as the Offset5 value.

FORMAT 10: LOAD/STORE HALFWORD

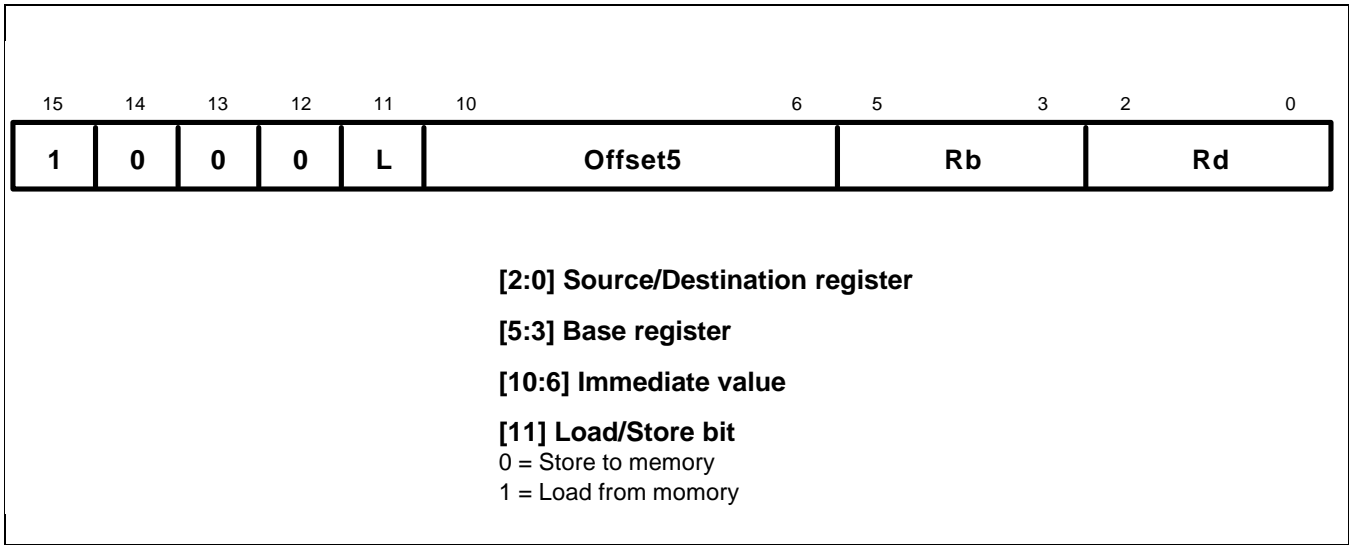


Figure 3-39. Format 10

OPERATION

These instructions transfer halfword values between a Lo register and memory. Addresses are pre-indexed, using a 6-bit immediate value. The THUMB assembler syntax is shown in Table 3-17.

Table 3-17. Halfword Data Transfer Instructions

L	THUMB assembler	ARM equivalent	Action
0	STRH Rd, [Rb, #Imm]	STRH Rd, [Rb, #Imm]	Add #Imm to base address in Rb and store bits 0–15 of Rd at the resulting address.
1	LDRH Rd, [Rb, #Imm]	LDRH Rd, [Rb, #Imm]	Add #Imm to base address in Rb. Load bits 0-15 from the resulting address into Rd and set bits 16-31 to zero.

NOTE: #Imm is a full 6-bit address but must be halfword-aligned (ie with bit 0 set to 0) since the assembler places #Imm >> 1 in the Offset5 field.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-17. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

STRH	R6, [R1, #56]	; Store the lower 16 bits of R4 at the address formed by ; adding 56 R1. Note that the THUMB opcode will contain ; 28 as the Offset5 value.
LDRH	R4, [R7, #4]	; Load into R4 the halfword found at the address formed by ; adding 4 to R7. Note that the THUMB opcode will contain ; 2 as the Offset5 value.

FORMAT 11: SP-RELATIVE LOAD/STORE

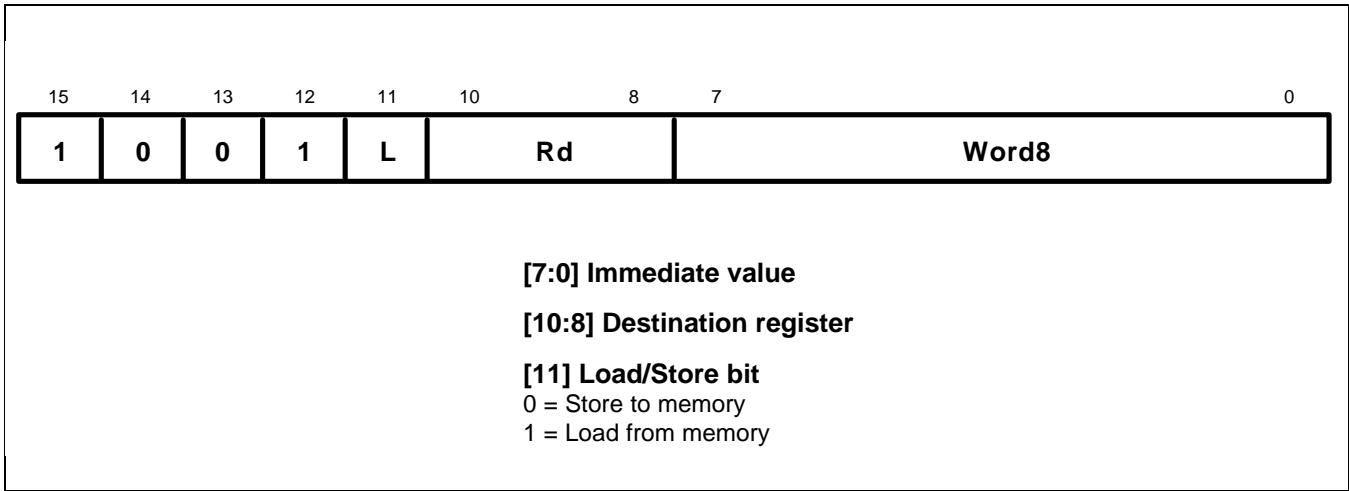


Figure 3-40. Format 11

OPERATION

The instructions in this group perform an SP-relative load or store. The THUMB assembler syntax is shown in the following table.

Table 3-18. SP-Relative Load/Store Instructions

L	THUMB assembler	ARM equivalent	Action
0	STR Rd, [SP, #Imm]	STR Rd, [R13 #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Store the contents of Rd at the resulting address.
1	LDR Rd, [SP, #Imm]	LDR Rd, [R13 #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Load the word from the resulting address into Rd.

NOTE: The offset supplied in #Imm is a full 10-bit address, but must always be word-aligned (ie bits 1:0 set to 0), since the assembler places #Imm >> 2 in the Word8 field.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-18. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

STR	R4, [SP,#492]	;	Store the contents of R4 at the address
		;	formed by adding 492 to SP (R13).
		;	Note that the THUMB opcode will contain
		;	123 as the Word8 value.

FORMAT 12: LOAD ADDRESS

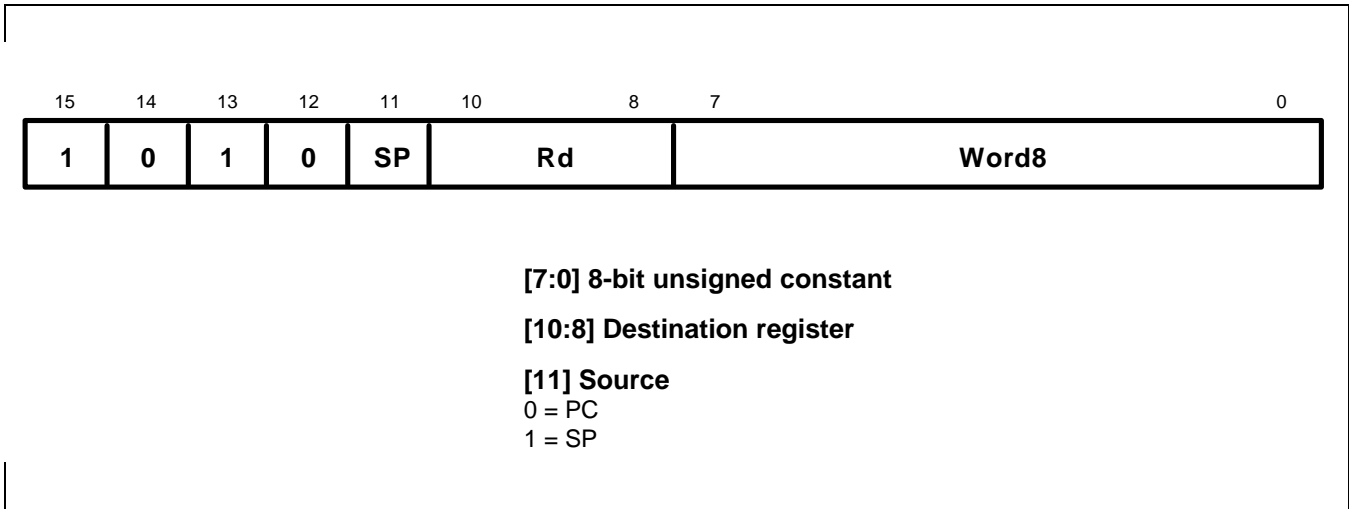


Figure 3-41. Format 12

OPERATION

These instructions calculate an address by adding an 10-bit constant to either the PC or the SP, and load the resulting address into a register. The THUMB assembler syntax is shown in the following table.

Table 3-19. Load Address

SP	THUMB assembler	ARM equivalent	Action
0	ADD Rd, PC, #Imm	ADD Rd, R15, #Imm	Add #Imm to the current value of the program counter (PC) and load the result into Rd.
1	ADD Rd, SP, #Imm	ADD Rd, R13, #Imm	Add #Imm to the current value of the stack pointer (SP) and load the result into Rd.

NOTE: The value specified by #Imm is a full 10-bit value, but this must be word-aligned (ie with bits 1:0 set to 0) since the assembler places #Imm >> 2 in field Word 8.

Where the PC is used as the source register (SP = 0), bit 1 of the PC is always read as 0. The value of the PC will be 4 bytes greater than the address of the instruction before bit 1 is forced to 0.

The CPSR condition codes are unaffected by these instructions.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-19. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

ADD	R2, PC, #572	; R2 := PC + 572, but don't set the ; condition codes. bit[1] of PC is forced to zero. ; Note that the THUMB opcode will ; contain 143 as the Word8 value.
ADD	R6, SP, #212	; R6 := SP (R13) + 212, but don't ; set the condition codes. ; Note that the THUMB opcode will ; contain 53 as the Word 8 value.

FORMAT 13: ADD OFFSET TO STACK POINTER

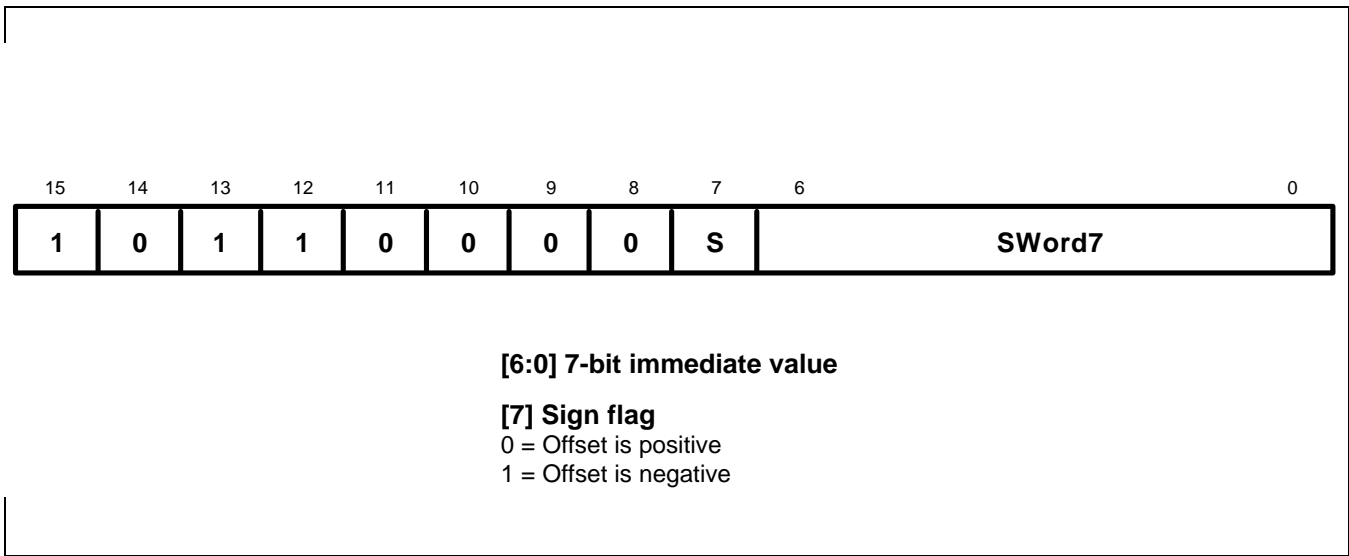


Figure 3-42. Format 13

OPERATION

This instruction adds a 9-bit signed constant to the stack pointer. The following table shows the THUMB assembler syntax.

Table 3-20. The ADD SP Instruction

S	THUMB assembler	ARM equivalent	Action
0	ADD SP, #Imm	ADD R13, R13, #Imm	Add #Imm to the stack pointer (SP).
1	ADD SP, #-Imm	SUB R13, R13, #Imm	Add #-Imm to the stack pointer (SP).

NOTE: The offset specified by #Imm can be up to -/+ 508, but must be word-aligned (ie with bits 1:0 set to 0) since the assembler converts #Imm to an 8-bit sign + magnitude number before placing it in field SWord7. The condition codes are not set by this instruction.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-20. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

ADD	SP, #268	; SP (R13) := SP + 268, but don't set the condition codes. ; Note that the THUMB opcode will ; contain 67 as the Word7 value and S=0.
ADD	SP, #-104	; SP (R13) := SP - 104, but don't set the condition codes. ; Note that the THUMB opcode will contain ; 26 as the Word7 value and S=1.

FORMAT 14: PUSH/POP REGISTERS

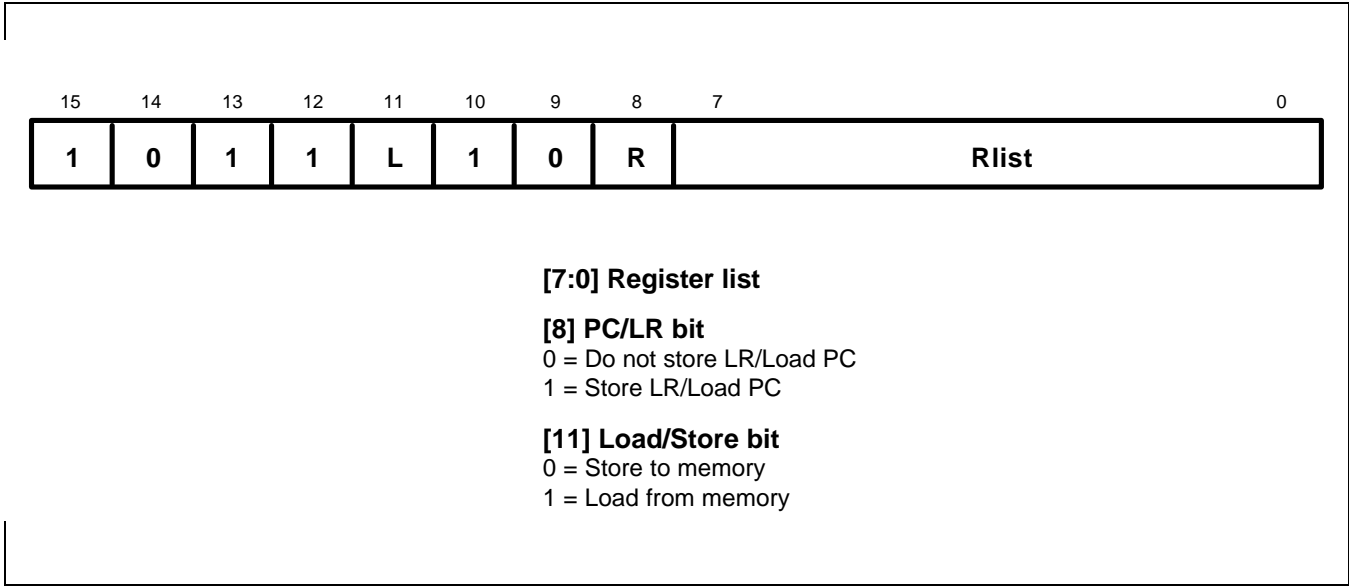


Figure 3-43. Format 14

OPERATION

The instructions in this group allow registers 0-7 and optionally LR to be pushed onto the stack, and registers 0-7 and optionally PC to be popped off the stack. The THUMB assembler syntax is shown in Table 3-21.

NOTE

The stack is always assumed to be Full Descending.

Table 3-21. PUSH and POP Instructions

L	R	THUMB assembler	ARM equivalent	Action
0	0	PUSH { Rlist }	STMDB R13!, { Rlist }	Push the registers specified by Rlist onto the stack. Update the stack pointer.
0	1	PUSH { Rlist, LR }	STMDB R13!, { Rlist, R14 }	Push the Link Register and the registers specified by Rlist (if any) onto the stack. Update the stack pointer.
1	0	POP { Rlist }	LDMIA R13!, { Rlist }	Pop values off the stack into the registers specified by Rlist. Update the stack pointer.
1	1	POP { Rlist, PC }	LDMIA R13!, {Rlist, R15}	Pop values off the stack and load into the registers specified by Rlist. Pop the PC off the stack. Update the stack pointer.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-21. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

PUSH	{R0-R4,LR}	;	Store R0,R1,R2,R3,R4 and R14 (LR) at
		;	the stack pointed to by R13 (SP) and update R13.
		;	Useful at start of a sub-routine to
		;	save workspace and return address.
POP	{R2,R6,PC}	;	Load R2,R6 and R15 (PC) from the stack
		;	pointed to by R13 (SP) and update R13.
		;	Useful to restore workspace and return from sub-routine.

FORMAT 15: MULTIPLE LOAD/STORE

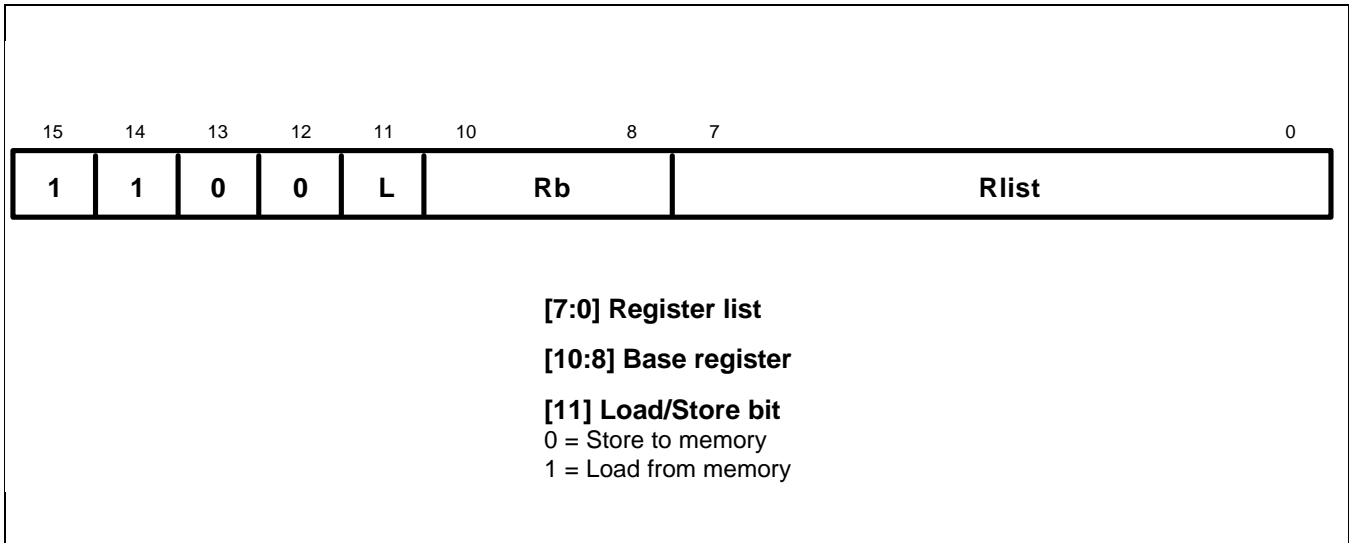


Figure 3-44. Format 15

OPERATION

These instructions allow multiple loading and storing of Lo registers. The THUMB assembler syntax is shown in the following table.

Table 3-22. The Multiple Load/Store Instructions

L	THUMB assembler	ARM equivalent	Action
0	STMIA Rb!, { Rlist }	STMIA Rb!, { Rlist }	Store the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.
1	LDMIA Rb!, { Rlist }	LDMIA Rb!, { Rlist }	Load the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-22. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

STMIAR0!, {R3-R7}

; Store the contents of registers R3-R7

; starting at the address specified in

; R0, incrementing the addresses for each word.

; Write back the updated value of R0.

FORMAT 16: CONDITIONAL BRANCH

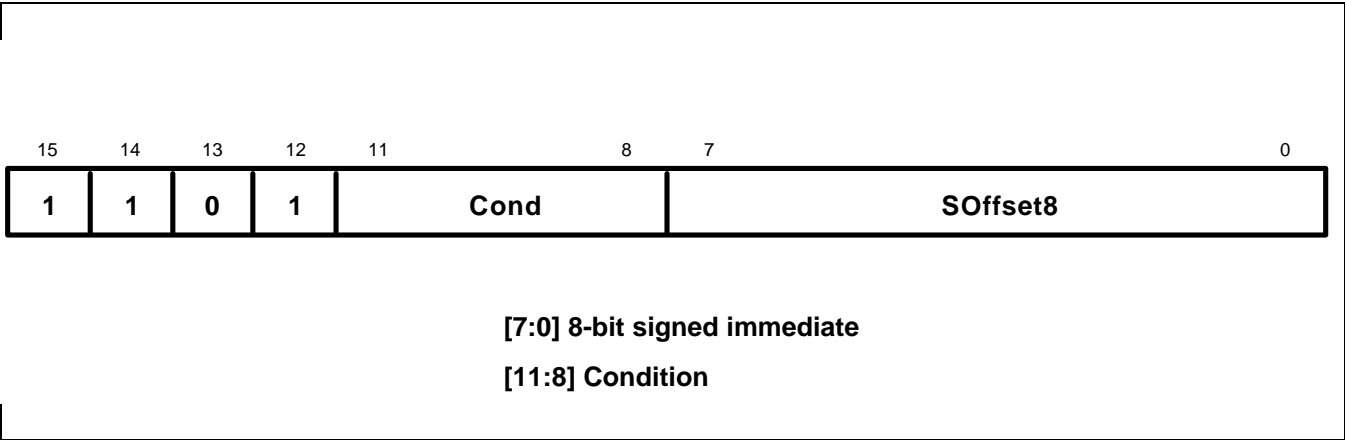


Figure 3-45. Format 16

OPERATION

The instructions in this group all perform a conditional Branch depending on the state of the CPSR condition codes. The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction.

The THUMB assembler syntax is shown in the following table.

Table 3-23. The Conditional Branch Instructions

Cond	THUMB assembler	ARM equivalent	Action
0000	BEQ label	BEQ label	Branch if Z set (equal)
0001	BNE label	BNE label	Branch if Z clear (not equal)
0010	BCS label	BCS label	Branch if C set (unsigned higher or same)
0011	BCC label	BCC label	Branch if C clear (unsigned lower)
0100	BMI label	BMI label	Branch if N set (negative)
0101	BPL label	BPL label	Branch if N clear (positive or zero)
0110	BVS label	BVS label	Branch if V set (overflow)
0111	BVC label	BVC label	Branch if V clear (no overflow)
1000	BHI label	BHI label	Branch if C set and Z clear (unsigned higher)
1001	BLS label	BLS label	Branch if C clear or Z set (unsigned lower or same)
1010	BGE label	BGE label	Branch if N set and V set, or N clear and V clear (greater or equal)

Table 3-23. The Conditional Branch Instructions (Continued)

Cond	THUMB assembler	ARM equivalent	Action
1011	BLT label	BLT label	Branch if N set and V clear, or N clear and V set (less than)
1100	BGT label	BGT label	Branch if Z clear, and either N set and V set or N clear and V clear (greater than)
1101	BLE label	BLE label	Branch if Z set, or N set and V clear, or N clear and V set (less than or equal)

NOTES

- 1. While label specifies a full 9-bit two’s complement address, this must always be halfword-aligned (ie with bit 0 set to 0) since the assembler actually places label >> 1 in field SOffset8.
- 2. Cond = 1110 is undefined, and should not be used.
Cond = 1111 creates the SWI instruction: see .

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-23. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

CMP R0, #45
BGT over
...
...
over
...
...

; Branch to 'over' if R0 > 45.
; Note that the THUMB opcode will contain
; the number of halfwords to offset.

; Must be halfword aligned.

FORMAT 17: SOFTWARE INTERRUPT

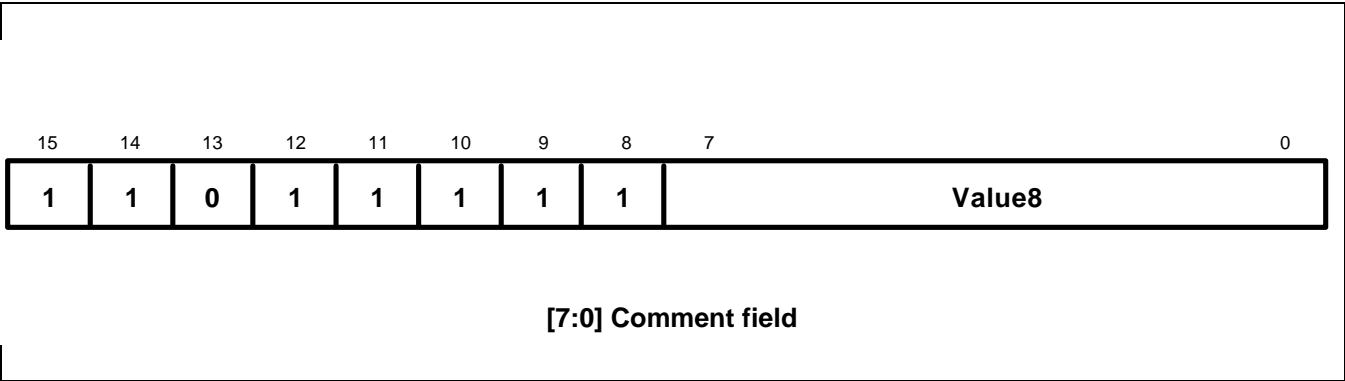


Figure 3-46. Format 17

OPERATION

The SWI instruction performs a software interrupt. On taking the SWI, the processor switches into ARM state and enters Supervisor (SVC) mode.

The THUMB assembler syntax for this instruction is shown below.

Table 3-24. The SWI Instruction

THUMB Assembler	ARM Equivalent	Action
SWI Value 8	SWI Value 8	Perform Software Interrupt: Move the address of the next instruction into LR, move CPSR to SPSR, load the SWI vector address (0x8) into the PC. Switch to ARM state and enter SVC mode.

NOTE: Value8 is used solely by the SWI handler; it is ignored by the processor.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-24. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

SWI 18

; Take the software interrupt exception.
; Enter Supervisor mode with 18 as the
; requested SWI number.

FORMAT 18: UNCONDITIONAL BRANCH

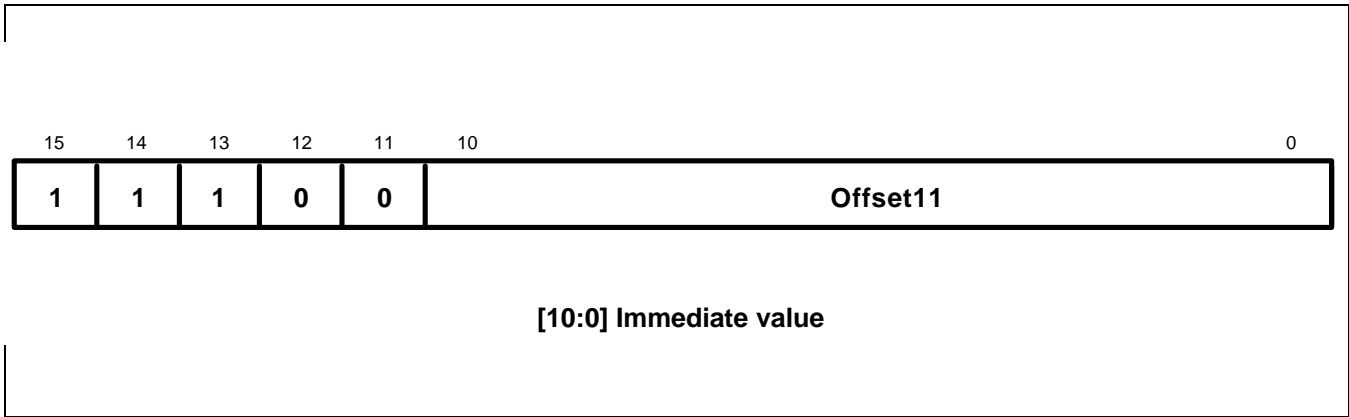


Figure 3-47. Format 18

OPERATION

This instruction performs a PC-relative Branch. The THUMB assembler syntax is shown below. The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction.

Table 3-25. Summary of Branch Instruction

THUMB Assembler	ARM Equivalent	Action
B label	BAL label (halfword offset)	Branch PC relative +/- Offset11 << 1, where label is PC +/- 2048 bytes.

NOTE: The address specified by label is a full 12-bit two's complement address, but must always be halfword aligned (ie bit 0 set to 0), since the assembler places label >> 1 in the Offset11 field.

EXAMPLES

here	B here	; Branch onto itself. Assembles to 0xE7FE.
		; (Note effect of PC offset).
	B jimmy	; Branch to 'jimmy'.
	...	; Note that the THUMB opcode will contain the number of
		; halfwords to offset.
jimmy	...	; Must be halfword aligned.

FORMAT 19: LONG BRANCH WITH LINK

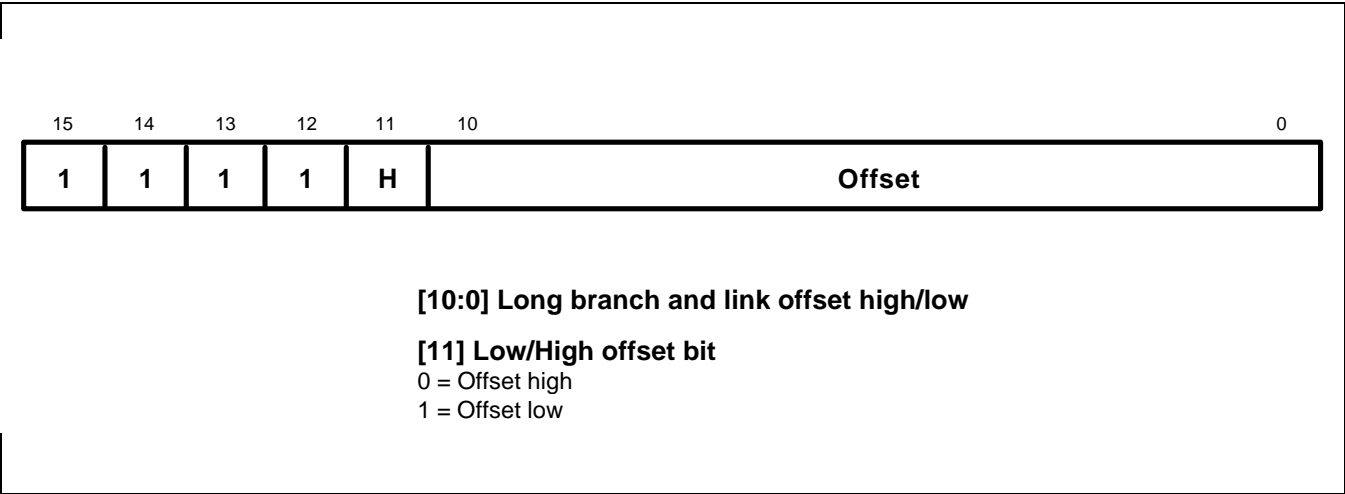


Figure 3-48. Format 19

OPERATION

This format specifies a long branch with link.

The assembler splits the 23-bit two's complement half-word offset specified by the label into two 11-bit halves, ignoring bit 0 (which must be 0), and creates two THUMB instructions.

Instruction 1 (H = 0)

In the first instruction the Offset field contains the upper 11 bits of the target address. This is shifted left by 12 bits and added to the current PC address. The resulting address is placed in LR.

Instruction 2 (H =1)

In the second instruction the Offset field contains an 11-bit representation lower half of the target address. This is shifted left by 1 bit and added to LR. LR, which now contains the full 23-bit address, is placed in PC, the address of the instruction following the BL is placed in LR and bit 0 of LR is set.

The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction

INSTRUCTION CYCLE TIMES

This instruction format does not have an equivalent ARM instruction.

Table 3-26. The BL Instruction

H	THUMB assembler	ARM equivalent	Action
0	BL label	none	LR := PC + OffsetHigh << 12
1			temp := next instruction address PC := LR + OffsetLow << 1 LR := temp 1

EXAMPLES

```
BL faraway           ; Unconditionally Branch to 'faraway'
next ...             ; and place following instruction
                     ; address, ie 'next', in R14,the Link
                     ; register and set bit 0 of LR high.
                     ; Note that the THUMB opcodes will
faraway ...          ; contain the number of halfwords to offset.
                     ; Must be Half-word aligned.
```

INSTRUCTION SET EXAMPLES

The following examples show ways in which the THUMB instructions may be used to generate small and efficient code. Each example also shows the ARM equivalent so these may be compared.

MULTIPLICATION BY A CONSTANT USING SHIFTS AND ADDS

The following shows code to multiply by various constants using 1, 2 or 3 Thumb instructions alongside the ARM equivalents. For other constants it is generally better to use the built-in MUL instruction rather than using a sequence of 4 or more instructions.

Thumb

ARM

1. Multiplication by 2^n (1,2,4,8,...)

LSL	Ra, Rb, LSL #n	; MOV Ra, Rb, LSL #n
-----	----------------	----------------------

2. Multiplication by 2^{n+1} (3,5,9,17,...)

LSL	Rt, Rb, #n	; ADD Ra, Rb, Rb, LSL #n
ADD	Ra, Rt, Rb	

3. Multiplication by 2^{n-1} (3,7,15,...)

LSL	Rt, Rb, #n	; RSB Ra, Rb, Rb, LSL #n
SUB	Ra, Rt, Rb	

4. Multiplication by -2^n (-2, -4, -8, ...)

LSL	Ra, Rb, #n	; MOV Ra, Rb, LSL #n
MVN	Ra, Ra	; RSB Ra, Ra, #0

5. Multiplication by -2^{n-1} (-3, -7, -15, ...)

LSL	Rt, Rb, #n	; SUB Ra, Rb, Rb, LSL #n
SUB	Ra, Rb, Rt	

Multiplication by any $C = \{2^{n+1}, 2^{n-1}, -2^n \text{ or } -2^{n-1}\} * 2^n$

Effectively this is any of the multiplications in 2 to 5 followed by a final shift. This allows the following additional constants to be multiplied. 6, 10, 12, 14, 18, 20, 24, 28, 30, 34, 36, 40, 48, 56, 60, 62

(2..5)		; (2..5)
LSL	Ra, Ra, #n	; MOV Ra, Ra, LSL #n

GENERAL PURPOSE SIGNED DIVIDE

This example shows a general purpose signed divide and remainder routine in both Thumb and ARM code.

Thumb code

```

;signed_divide                                ; Signed divide of R1 by R0: returns quotient in R0,
                                                ; remainder in R1

;Get abs value of R0 into R3
    ASR     R2, R0, #31                      ; Get 0 or -1 in R2 depending on sign of R0
    EOR     R0, R2                          ; EOR with -1 (0xFFFFFFFF) if negative
    SUB     R3, R0, R2                      ; and ADD 1 (SUB -1) to get abs value

;SUB always sets flag so go & report division by 0 if necessary
    BEQ     divide_by_zero

;Get abs value of R1 by xoring with 0xFFFFFFFF and adding 1 if negative
    ASR     R0, R1, #31                      ; Get 0 or -1 in R3 depending on sign of R1
    EOR     R1, R0                          ; EOR with -1 (0xFFFFFFFF) if negative
    SUB     R1, R0                          ; and ADD 1 (SUB -1) to get abs value

;Save signs (0 or -1 in R0 & R2) for later use in determining ; sign of quotient & remainder.
    PUSH    {R0, R2}

;Justification, shift 1 bit at a time until divisor (R0 value) ; is just <= than dividend (R1 value). To do this shift
;dividend ; right by 1 and stop as soon as shifted value becomes >.
    LSR     R0, R1, #1
    MOV     R2, R3
    B       %FT0
just_l    LSL     R2, #1
0         CMP     R2, R0
         BLS     just_l

    MOV     R0, #0                          ; Set accumulator to 0
    B       %FT0                          ; Branch into division loop

div_l    LSR     R2, #1
0         CMP     R1, R2                      ; Test subtract
         BCC     %FT0
         SUB     R1, R2                      ; If successful do a real subtract
0         ADC     R0, R0                      ; Shift result and add 1 if subtract succeeded

         CMP     R2, R3                      ; Terminate when R2 == R3 (ie we have just
         BNE     div_l                      ; tested subtracting the 'ones' value).

```

;Now fixup the signs of the quotient (R0) and remainder (R1)

```

    POP        {R2, R3}          ; Get dividend/divisor signs back
    EOR        R3, R2            ; Result sign
    EOR        R0, R3            ; Negate if result sign = - 1
    SUB        R0, R3
    EOR        R1, R2            ; Negate remainder if dividend sign = - 1
    SUB        R1, R2
    MOV        pc, lr

```

ARM Code

signed_divide ; Effectively zero a4 as top bit will be shifted out later

```

    ANDS       a4, a1, #&80000000
    RSBMI      a1, a1, #0
    EORS       ip, a4, a2, ASR #32

```

;ip bit 31 = sign of result

;ip bit 30 = sign of a2

```

    RSBCS      a2, a2, #0

```

;Central part is identical code to udiv (without MOV a4, #0 which comes for free as part of signed entry sequence)

```

    MOVS       a3, a1
    BEQ        divide_by_zero

```

just_l ; Justification stage shifts 1 bit at a time

```

    CMP        a3, a2, LSR #1
    MOVLS      a3, a3, LSL #1      ; NB: LSL #1 is always OK if LS succeeds
    BLO        s_loop

```

div_l

```

    CMP        a2, a3
    ADC        a4, a4, a4
    SUBCS      a2, a2, a3
    TEQ        a3, a1
    MOVNE      a3, a3, LSR #1
    BNE        s_loop2
    MOV        a1, a4
    MOVS       ip, ip, ASL #1
    RSBCS      a1, a1, #0
    RSBMI      a2, a2, #0
    MOV        pc, lr

```

DIVISION BY A CONSTANT

Division by a constant can often be performed by a short fixed sequence of shifts, adds and subtracts.

Here is an example of a divide by 10 routine based on the algorithm in the ARM Cookbook in both Thumb and ARM code.

Thumb Code

```

udiv10                                ; Take argument in a1 returns quotient in a1,
                                       ; remainder in a2
    MOV    a2, a1
    LSR    a3, a1, #2
    SUB    a1, a3
    LSR    a3, a1, #4
    ADD    a1, a3
    LSR    a3, a1, #8
    ADD    a1, a3
    LSR    a3, a1, #16
    ADD    a1, a3
    LSR    a1, #3
    ASL    a3, a1, #2
    ADD    a3, a1
    ASL    a3, #1
    SUB    a2, a3
    CMP    a2, #10
    BLT    %FT0
    ADD    a1, #1
    SUB    a2, #10
0
    MOV    pc, lr

```

ARM Code

```

udiv10                                ; Take argument in a1 returns quotient in a1,
                                       ; remainder in a2
    SUB    a2, a1, #10
    SUB    a1, a1, a1, lsr #2
    ADD    a1, a1, a1, lsr #4
    ADD    a1, a1, a1, lsr #8
    ADD    a1, a1, a1, lsr #16
    MOV    a1, a1, lsr #3
    ADD    a3, a1, a1, asl #2
    SUBS   a2, a2, a3, asl #1
    ADDPL  a1, a1, #1
    ADDMI  a2, a2, #10
    MOV    pc, lr

```

NOTES

4 System Manager

OVERVIEW

The KS32C6200 System Manager has the following functions:

- Arbitrates bus access requests from several master blocks, based on a fixed priority.
- Provides the required memory control signals for external memory accesses. For example, if a master block such as DMA or the CPU generates an address that corresponds to a DRAM bank, the System Manager's DRAM controller generates the required DRAM access signals (nRAS, nCAS, and so on).
- Supports big-endian mode for most graphic device drivers (see Figure 4-5).
- Compensates for differences in bus width for data flowing between the external data bus and the internal data bus.

SYSTEM MANAGER REGISTERS

The KS32C6200 microcontroller has the SFRs, Special Function Registers, to keep the system control information of system manager, cache, DMA, UART and so on. The SFRs have the SMR, System Manager Register files, to configure the external memory maps such as DRAM, SRAM, ROM and extra-I/O control.

By utilizing the SMR, you can specify the memory type, external bus width, access cycles, required control signal timings (nRAS, nCAS and so on), memory bank location and each memory bank size which has a very configurable address space. The SMR provides (or accepts) the control signals, addresses, and data that are required by external devices during normal system operation. There are eleven registers to control one ROM bank, two SRAM banks, two DRAM banks, four extra-I/O banks and DRAM Refresh.

The KS32C6200 provides up to 32 M bytes of address space and each bank provides up to 4 M half-word memory space because each bank has 22 address pins and 16-bit data width.

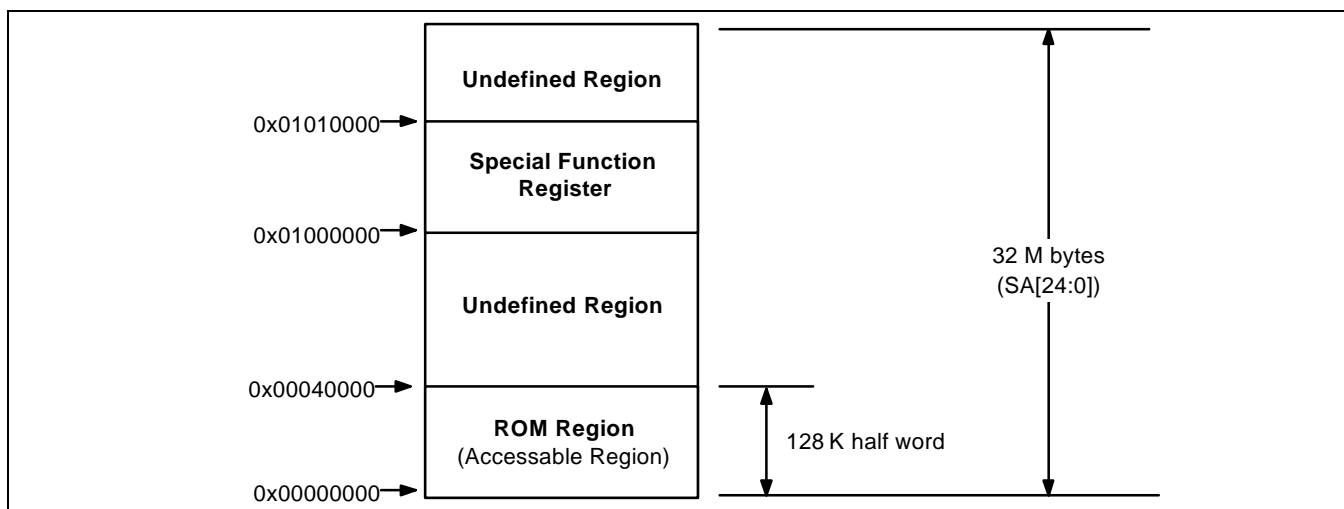


Figure 4-1. KS32C6200 Memory Map (Default Map after Reset)

The KS32C6200 provides 32-MByte memory space and an internal 25-bit system address bus. You can use any address area from 0000000h to 1FEFFFFh by 64-Kbyte address steps. Each bank can be located anywhere in the 32-MByte address space, except the upper 64-Kbyte area where SFRs (Special Function Registers) can be located. To use the full 32-MByte memory space, we recommend for you to allocate the SFRs to the upper 64-kbyte address area, 1FF0000h–1FFFFFFh.

The configurable memory allocation in the KS32C6200 is designed to provide you with convenience. By manipulating the SMRs, you can easily allocate the memory area anywhere you wish and use the consecutively connected memory space without changing the H/W. You can also easily change the physical DRAM memory size by manipulating the SMR.

For example, if you want to change the size of memory space from 1 M half word to 4 M half word. You can enlarge the memory space just by changing the next pointer of the DRAM bank.

NOTE

The last 64 Kbyte area can not be allocated as memory banks except SFR. Because the last 64 Kbyte bank starts its address 1FFxxxxh, the next pointer of the last bank should be "200xxxxh". Actually, the next point is 9 bits, so the value of the next pointer is 000xxxxh. If you need to utilize the full 32 M bytes of memory space, it is recommended that you allocate the SFRs to the last 64-Kbyte area, 1F0000h–1FFFFFFh, and to use the remaining areas for other banks.

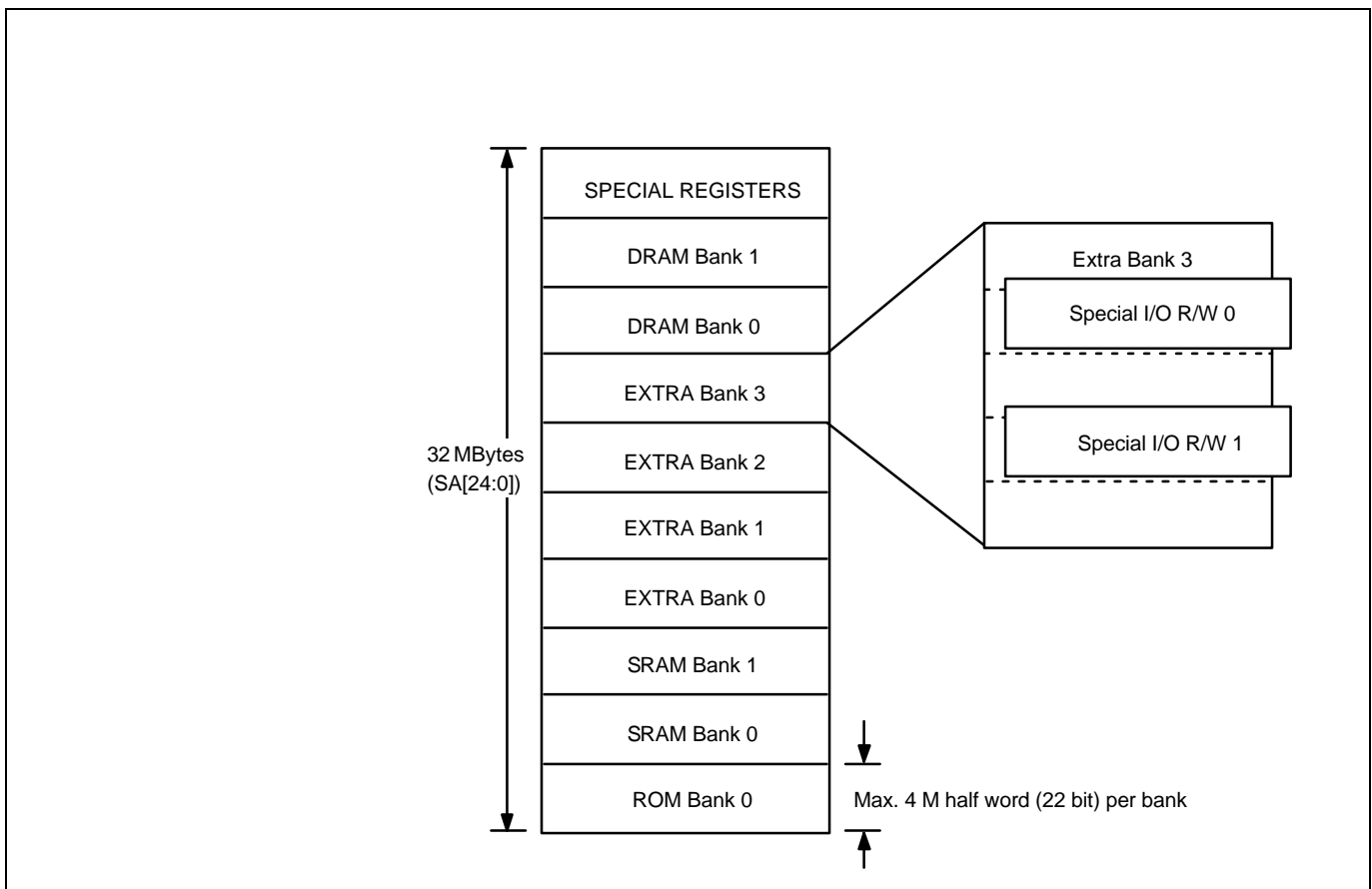


Figure 4-2. System Memory Map

SYSTEM REGISTER ADDRESS CONFIGURATION REGISTER (SYSCFG)

The SMRs (System Manager Registers) have the SYSCFG (System Register Address Configuration Register), which determines the start address (base point) of SFR (Special Function Register) files. The SYSCFG contains the start address of SFR. If the reset value of SYSCFG is 1001h, the SYSCFG is mapped to the virtual address 01000000h.

Register	Offset Address	R/W	Description	Reset Value
SYSCFG	0x0000	R/W	Special function register to determine the start address	0x1001

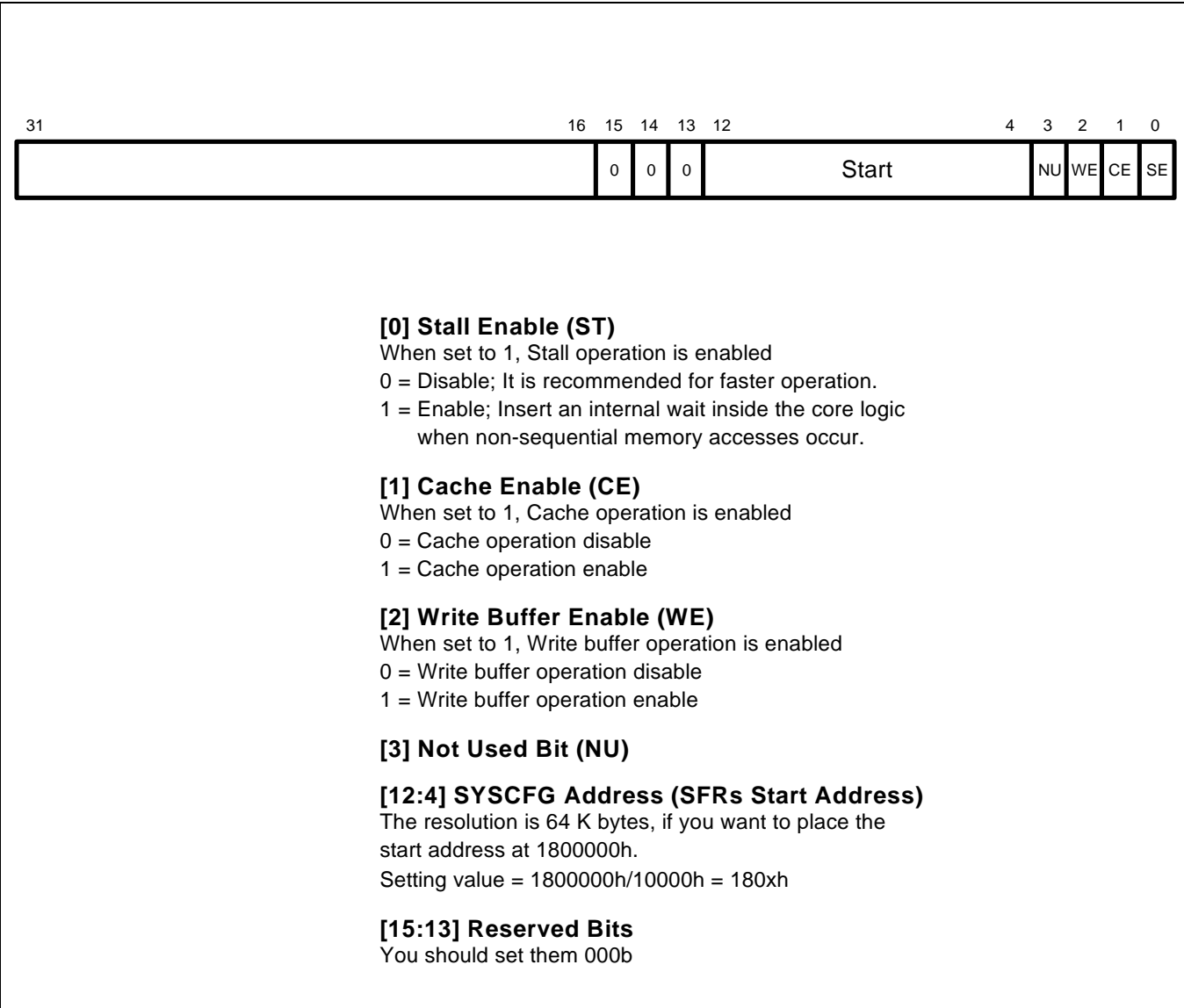


Figure 4-3. System Register Address Configuration Register (SYSCFG)

Start Address

The SYSCFG[12:4] bits indicate the start address of SFRs. Since the SYSCFG is located at the bottom of the SFR file, the SYSCFG's location is the same as the start address of the SFRs.

You can allocate the SFRs to the arbitrary location by manipulating the SYSCFG. We recommend that you do not change the SYSCFG during operation once you have configured it after a system reset. The SYSCFG should not be overlapped with any other bank.

If the start address of the SYSCFG is changed, the other control registers in the SFRs will have the new address which is the sum of its offset address and the new address of SYSCFG. For example, after a system reset, the initial address of SYSCFG is 1000000h and the ROM control register has the initial address '1003000h, because the ROM control register has the offset value '3000h', and the initial address is the sum of 1000000h and 3000h. If the SYSCFG address is changed to 1800000h, the address of ROM control register will be 1803000h.

Cache Disable/Enable

You can disable/enable the cache memory in the KS32C6200. You can enable the cache memory by setting the CE bit to logic 'one'. You can also use the non-cacheble area to maintain the data coherency of a specific memory area. Because the cache memory does not have the auto-flushing mode, you should be careful about the data coherency when you reenale the cache memory. You also have to check whether or not the DMA changes the memory data. The DMA accessible memory area must be non-cacheble to keep data coherency.

To keep the data coherency between the cache and the external memory, the KS32C6200 uses the write-through policy. To compensate for performance degradation, there is an internal four-depth write buffer. Please refer to Chapter 5 for more detail information.

Write Buffer Disable/Enable

The KS32C6200 has four 'Write Buffer Registers' to enhance its memory writing performance. When the Write Buffer mode is enabled, the CPU writes data into the Write Buffer first, instead of an external memory which requires the longer memory write cycles. Write Buffer has 4 registers and each register includes a 32-bit data field, a 25-bit address field and a 2-bit status field.

Stall Disable/Enable

When the stall option is enabled, the MCU core logic inserts a wait, which occurs by non-sequential memory access. The MCU core has a larger time margin when memory access is executed. When the stall option is disabled, the MCU core logic does not insert a wait signal. The operation time of MCU core logic is faster than when the stall option is enabled.

ROM CONTROL REGISTER

The KS32C6200 supports one ROM bank for program memory. The ROM bank has configurable features such as access timing, access size and page mode support. The ROMCON (ROM Control Register) in SMR supplies the control modes such as normal mode access, page mode access and wait cycles of each mode and external ROM bank.

The initial address of ROMCON is 01003000h which is the sum of the initial address of SYSCFG (01000000h) and ROM control register offset address (00003000h). The register address is reconfigurable. You can change the ROM control register by changing the content of SYSCFG.

Register	Offset Address	R/W	Description	Reset Value
ROMCON	0x3000	R/W	ROM control register	02003002h

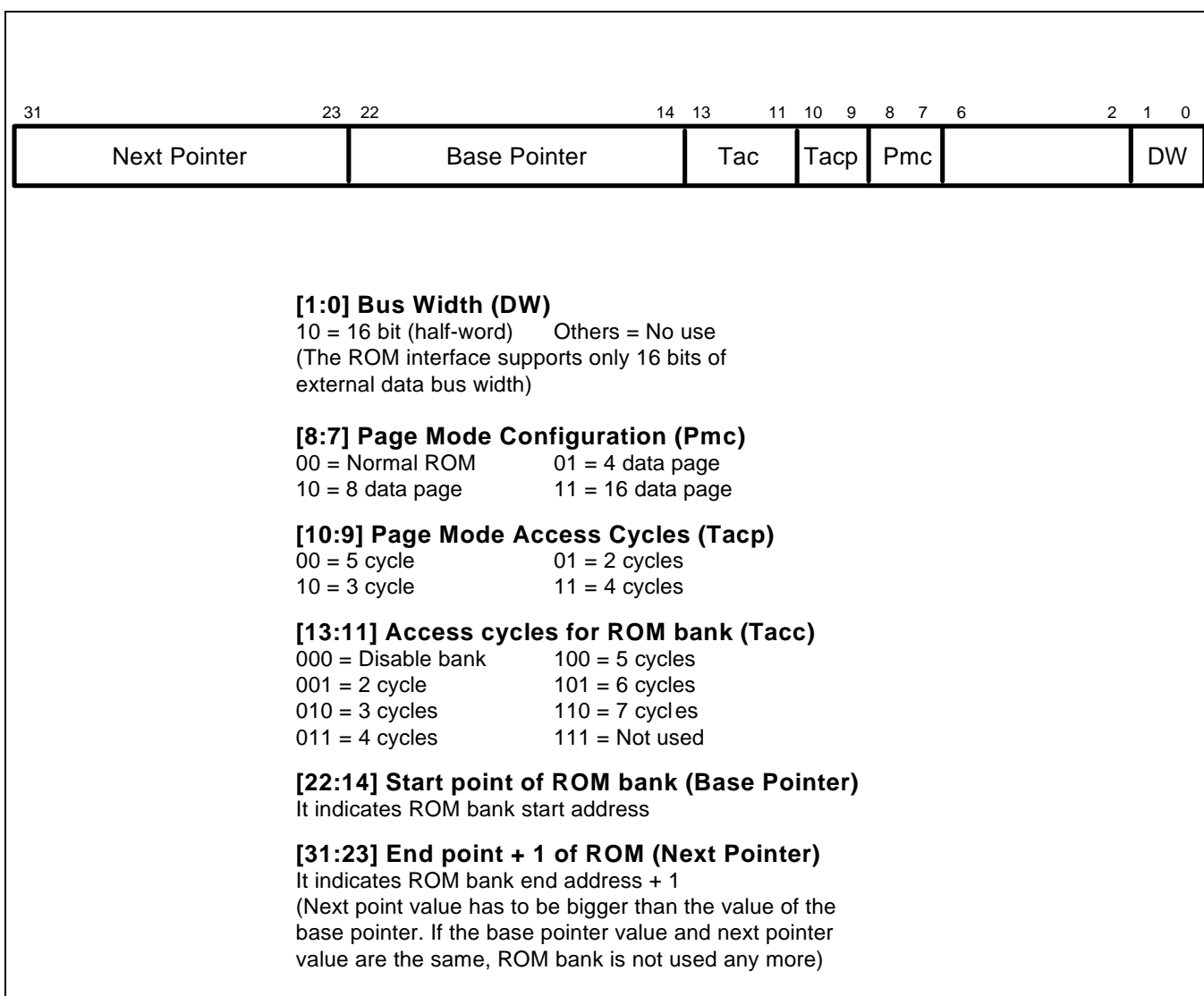


Figure 4-4. ROM Control Register (ROMCON)

Page Mode ROM Access (Burst Mode Access)

KS32C6200 ROM can interface with simple ROM and page mode ROM. You can enable/disable burst mode and can define the readable number of burst data by using ROMCON[8:7]. ROM has two different access cycles for simple ROM and page mode ROM. When page mode is selected, the first data access time is different from the access time of the following data if the data is in the same page.

T_{acc} , access cycles for ROM bank, is defined as the access cycle after the active bank changes to the ROM bank. This cycle time is also used for simple ROM access mode. When CPU reads consecutive data within the same page, the page mode ROM supplies a data read cycle shorter than reading the different page. The T_{acp} bit in ROM control register defines the consecutive data read cycles in page mode ROM.

Writing on the ROM bank

In the KS32C6200, you can write data into the ROM bank area. Though the internal program of ROM is not changed physically, you may need a writing feature if SRAM or flash ROM is installed into the ROM bank.

ROM Bank Space

You can configure memory space in the KS32C6200. You can manipulate the memory bank size and bank location by using the ROMCON (ROM Control Register). The ROMCON has two 9-bit address pointers. One is the base pointer and the other is the next pointer. These two pointers denote the beginning and ending addresses of the ROM bank. The values of these pointers are compared with the address[24:16] to make a bank-select signal. The size of the ROM bank area can be increased/decreased by 64 K bytes. The value of the next pointer should be the sum of ROM bank end address and one.

For example, after a system reset, the start address of ROM bank is 00000000h and the end address is 00FFFFFFh. The value of the next pointer is 0100h ((0FFFFFFh+1h)/64-Kbytes). If the values of the next pointer and the base pointer of the ROM bank are the same, the ROM bank will be disabled.

Initialization

After a system reset, the initial value of ROMCON is set to 80003002h. In the system initialization, the external bus width is 16-bits (half-word) and the normal ROM mode is enabled. The longest value of the page mode access cycles is selected.

ROM Programming

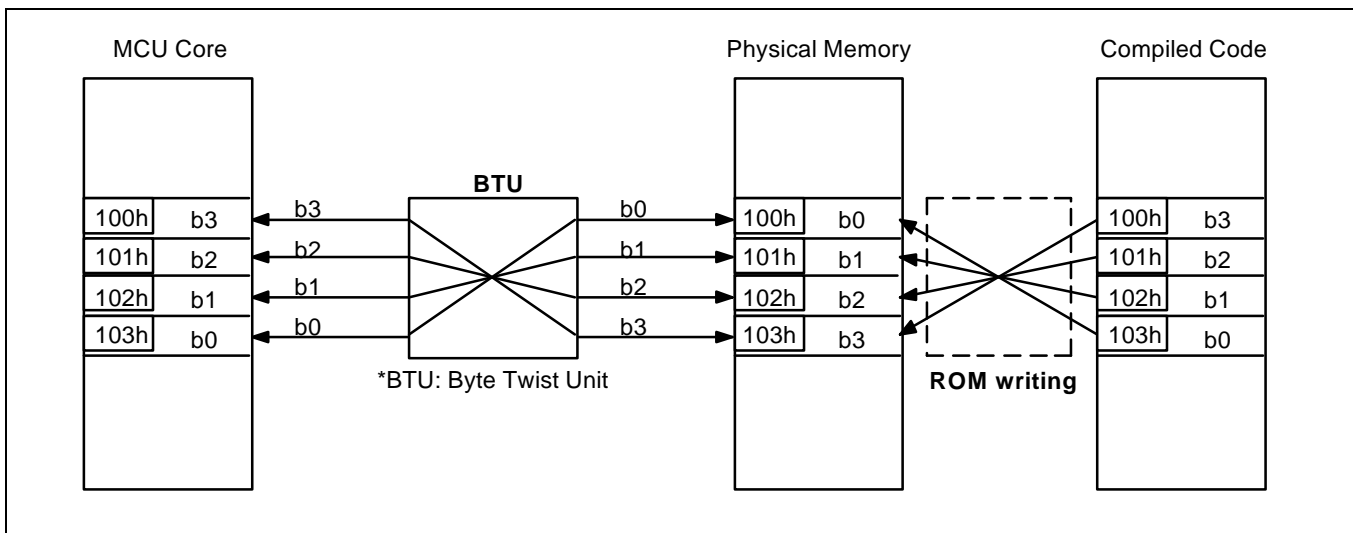


Figure 4-5. The Byte Swap Operation of BTU and Data Positions in Memory

Big-Endian Format / Little-Endian Format

In the Big-Endian format, the most significant byte of a word is stored at the lowest numbered byte and the least significant byte at the highest numbered byte. The byte '0' of the memory system is connected to data lines from 31 to 24.

In the Little-Endian format, the lowest numbered byte in a word is considered as the least significant byte, and the highest numbered byte as the most significant. The byte '0' of the memory system is connected to data lines from 7 to 0.

Big-Endian Supporting Core and Little-Engine Supporting Physical Memory

The ARM7TDMI core in KS32C6200 is a little-endian base core supporting the big-endian format. To support the big-endian format in ARM7TDMI, the KS32C6200 adopts BTU, Bus Twist Unit. The main function of BTU is to exchange bytes in a word as shown in Figure 4-5. For example, when MCU core accesses the data '100h', the BTU alters '100h' to '103h', and the CPU will get the data, 103h, in the physical memory.

To put big-endian format data into memory, a compiled code has to put into memory ROM by exchanging bytes in a word. In Figure 4-5, double swappings (BTU and compiled code swapping) enable Big-Endian format with no problem. The reason that the KS32C6200 uses the double swappings is due to internal hardware implementation issue.

Byte Swapping in a Word

Byte swapping is executed by using the sample C. The sample changes the byte sequence in a word.

Unsigned Long Swap (Unsigned long Data) // Make the Sequence of Bytes Reverse in a word

```
{
    return      ( ( (0xff000000 & data)>>24)+
                  ( (0x00ff0000 & data)>>8)+
                  ( (0x0000ff00 & data)<<8)+
                  ( (0x000000ff & data)<<24) );
}
```

ROM Writing

BTU changes the byte sequence in a word. Program codes are byte-swapped. To write program to ROM, do as follows;

1. Compile the program in big-endian format
2. Byte-swap the compiled code (recommend to make a program that executes byte-swap of a binary file)
3. Write the code to ROM

Little-Endian Format Code versus Byte-Swapped Big-Endian Format Code

If character strings do not exist in a program, the Little-Endian format code may be the same as the byte-swapped big-endian format code. Because the bytes in strings are not affected by the endian format, the two codes are different in strings. The byte-swapped big-endian format code has to be used in the KS32C6200. If Little-Endian format code is used, the strings will not be displayed correctly (byte-swapped strings may be displayed).

Interfacing External Peripherals

Peripheral address is also byte-swapped. If you want to access address 00b in memory, you have to access address 11b in the program. This is due to the word swapping of BTU. The relation between physical address and the address used by instructions is as follows.

Table 4-1. The Relation Between Physical Address and Address in Instructions

Physical address	Byte Wide Access (Address Used in Instructions)	Half Word Wide Access (Address Used in Instructions)
00b	11b	10b
01b	10b	–
10b	01b	00b
11b	00b	–

SRAM CONTROL REGISTERS

KS32C6200 has two banks of SRAM. Each bank can set up its SRAM access configuration. SRAMCON0 and SRAMCON1 (SRAM Control Registers) specify not only the features of SRAM but also two special I/Os (special I/O 0, special I/O 1) in the external bank 3.

The initial address of SRAMCON0,1 are 01003004h and 01003008h. The real address of each SRAM control registers is the sum of 'SYSCFG address' and 'offset address' of each SRAM control register. The register address is reconfigurable, so you can change the SRAM control register address using the SYSCFG.

Register	Offset Address	R/W	Description	Reset Value
SRAMCON0	0x3004	R/W	SRAM control register 0	0x00000000
SRAMCON1	0x3008	R/w	SRAM control register 1	0x00000000

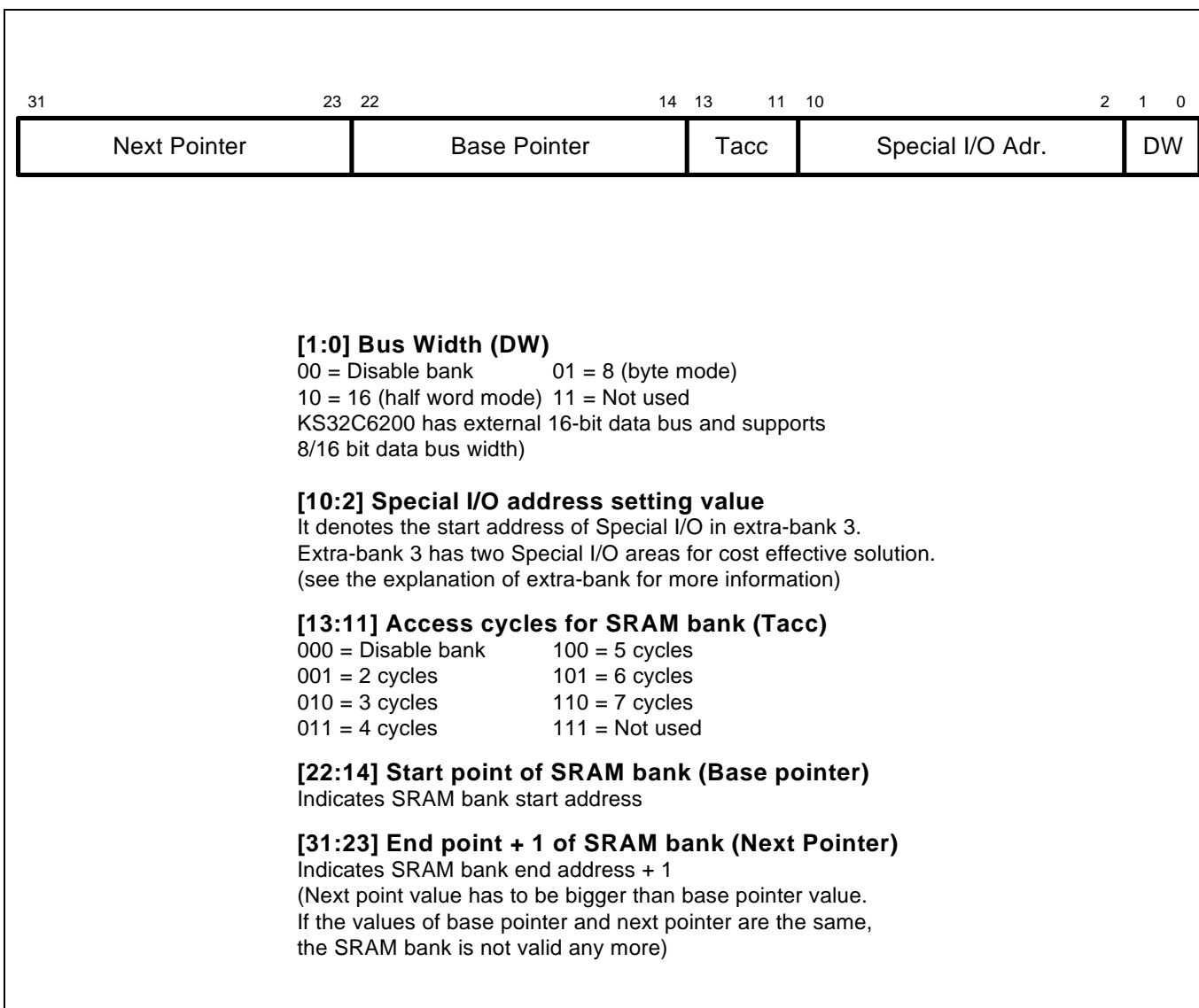


Figure 4-6. SRAM Control Registers (SRAMCON0, SRAMCON1)

SRAM Bank Space

KS32C6200 provides two SRAM banks and each bank can be configured differently. You can program SRAM access cycles, memory bank size and location by using the SRAMCON0,1. The SRAM control register has two 9-bit address pointers (base pointer and next pointer). These two pointers denote the start address and end address of the SRAM bank. These 9 bits are compared with the address [24:16] to make the bank-select signal. The size of SRAM bank area can be increased/decreased by 64 K bytes. The value of the next pointer should be the sum of the end address of SRAM bank and one.

Initialization

When the system is initialized, the value of both SRAM control registers is 0000000h and it specifies that SRAM banks are disabled because the next pointer and the base pointer have the same values.

Special I/O Address

The extra-bank 3 of KS32C6200 has two special I/O areas to make simple control signal for the external latch. Two SRAM control registers have 9 bits dedicated for these special I/O areas in the extra-bank 3.

extra-bank 3 provides four special control signals, nIORD0,1 and nLOWR0,1. When you read/write data from/to external latch devices, the four special control signals do not need additional address decoder logics. These signals are available only at extra-bank 3. When MCU accesses any special I/O area specified by SRAM control registers, the extra-bank generates I/O read/write signals to the corresponding area. Figure 4-20 shows the diagram of special I/O read/write interface logic.

Address Bus Generation

The address bus of KS32C6200 is quite different from the general MCUs'. Although general MCU does not use the A0 pin in 16-bit data bus width, the KS32C6200 always uses the A0 pin regardless of data bus width. When an 8-bit data bus is selected, the resolution of address bus is a byte and when a 16-bit is selected, the resolution of address bus is a half-word.

Data Bus Width	External Address Pins (ADDR[21:0])	Accessible Memory size
8-bits	A21-A0 (internal)	4 M bytes
16-bits	A22-A1 (internal)	4 M-byte half-word

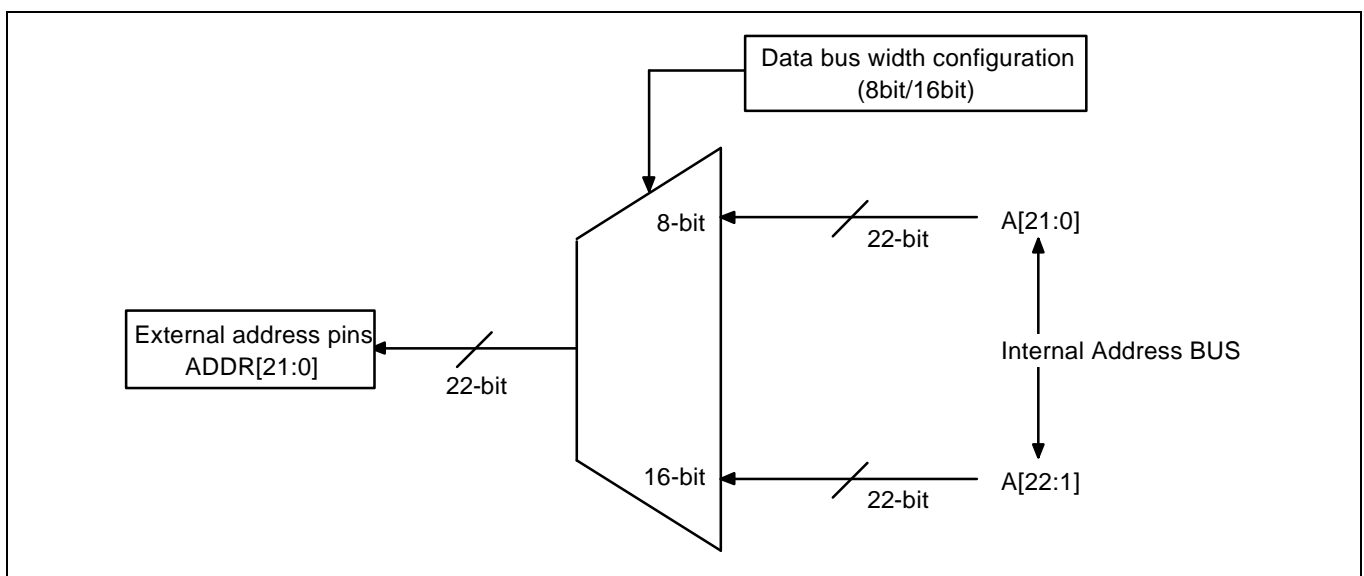


Figure 4-7. External Address Bus Generation (ADDR[21:0])

SRAM Bank Configuration

The nWBE (not Write Byte Enable) signal is for SRAM, the extra I/O or external memories such as flash memory. When CPU writes one-byte data to an external RAM with 16-bit data bus, only one 8-bit data must be written. However, the other 8-bit data should not be written. The nWBE[0] is for low-byte write operation and nWBE[1] is for upper-byte write operation.

The DRAM has different writing methods from SRAM and other external memories. The DRAM module has two CAS signals to separate data bus by a byte unit order. A RAS signal is used for bank selection and a CAS signal for byte selection.

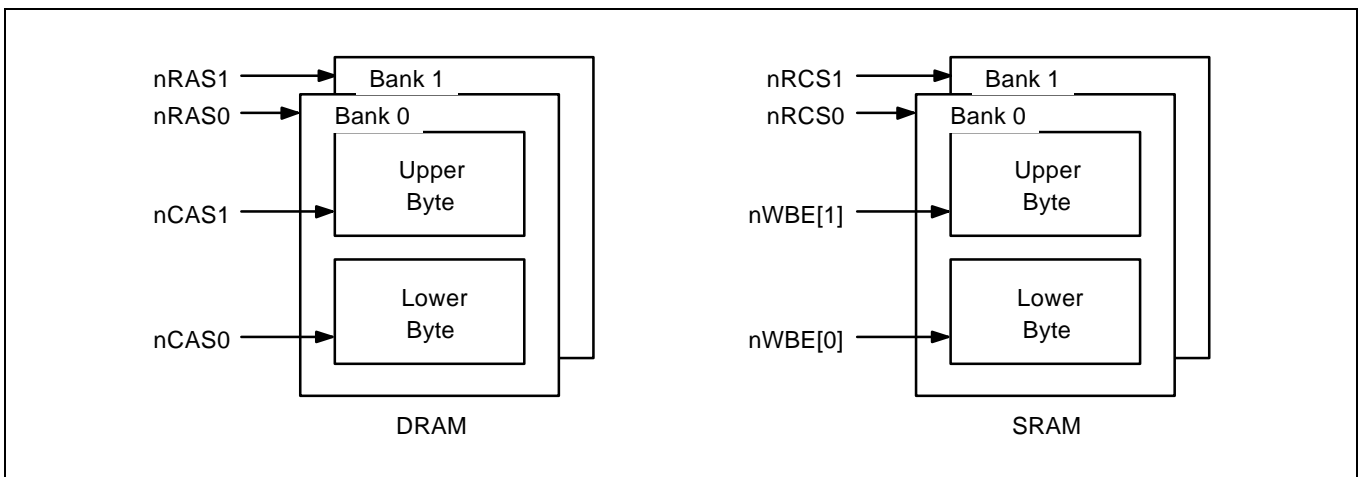


Figure 4-8. DRAM and SRAM Bank Configuration

DRAM CONTROL REGISTERS

KS32C62000 has two banks of DRAM and each bank can control DRAM access timing as other memory banks do. The DRAM interface has two DRAMCON0,1 (DRAM Control Registers) and one REFCON (DRAM Refresh Control Register). The initial addresses of each DRAM control register are 01003010h and 01003020h. The Refresh control register address is 01003024h. You can change the address of the DRAM control register by changing the value of SYSCFG.

Register	Offset Address	R/W	Description	Reset Value
DRAMCON0	0x0000301c	R/W	DRAM 0 control register	0x00000000
DRAMCON1	0x00003020	R/W	DRAM 1 control register	0x00000000

KS32C6200 provides a fully programmable configuration for DRAM interface. You can easily modify the configuration setting value such as external data bus width, number of access cycles for fast page or EDO, access cycles for each DRAM bank and row address strobe (nRAS) pre-charge timing by changing the value of the corresponding DRAM control register. The Refresh Control register controls the DRAM refresh operation. KS32C6200 supports CAS before the RAS (CBR) refresh mode and Self-Refresh mode.

KS32C6200 can generate row and column address, and supports symmetric/asymmetric address DRAM by changing the number of address line from 8 to 11. It can support various sizes of DRAM by varying column address size. If the number of a column address or a row address is larger than 11, the accessible DRAM memory size is smaller than the original size of the DRAM. For example, if a 16 M bit DRAM with 4 M x 4 (row address=12-bit and column address=10-bit) is connected to KS32C6200, the maximum accessible size of the memory is 8 M bit (11-bit x10-bit) and the other 8 M bit will be obsolete.

EDO Mode DRAM Access

Even if you specify DRAM as EDO mode, KS32C6200 gives the same timing diagram compared with normal fast page mode. However, KS32C6200 CPU fetches data (when read) a half-clock later than normal fast page mode, because EDO mode can make data valid even if CAS goes to high when RAS is low. CPU can have enough time to access and latch the data. Eventually, EDO mode can reduce memory access time.

DRAM Bank Space

KS32C6200 provides two DRAM banks and each bank can be configured differently. You can program the DRAM access cycles, memory bank size and bank location by using two DRAM control registers, DRAMCON0,1. DRAM control register has two 9-bit address pointers, Base and Next pointer. These two pointers denote the start and end address of DRAM bank. These 9-bits are compared with the address [24:16] to make the bank select signal. The size of the DRAM bank area can be increased/decreased by 64 K bytes. The value of the next pointer should be the sum of the end address of the bank and one.

Initialization

When the system is initialized, the start address and end address of two DRAM bank are 00000000h. It specifies that the DRAM bank is disabled because the respective values of the next pointer and the base pointer are the same.

DRAM Bank Configuration

The DRAM has different write methods from SRAM or other external memories. DRAM module has two CAS signals to separate data bus by byte order. RAS signal is used for bank selection and CAS signal is used for byte selection. Figure 4-9 shows the DRAM bank configuration.

Ex) Setting for 60 ns EDO DRAM (KM416V1204)

Conditions	Setting Value for DRAMCON
Memory map: 1000000h–11ffffh DRAM: 10 bits (row)x 10 bits (column) x 16 bits (data), 60 ns, EDO MCLK: 33 MHz	0x9040101a

31	23	22	14	13	12	11	10	8	7	6	5	4	3	2	1	0
Next Pointer			Base Pointer				Trp	Trc	Tcs	Tcp	Tpgm	EDO	CAN	DW		

[1:0] Bus Width (DW)
00 = Disable Bank 01 = 8(Byte)
10 = 16 (Half word) 11 = Not used

[3:2] Column Address Number (CAN)
00 = 8 bits 01 = 9 bits
10 = 10 bits 11 = 11 bits

[4] EDO ERAM or Ordinary DRAM (EDO)
0 = Ordinary 1 = EDO DRAM

[6:5] CAS strobe time (@ Page mode) (TPGM)
00 = 1 cycle 01 = 2 cycles
10 = 3 cycles 11 = 4 cycles

[7] CAS pre-charge (Tcp)
0 = 1 cycle 1 = 2 cycles

[10:8] CAS strobe time (@ Single mode) (Tcs)
000 = 1 cycle 100 = 5 cycles
001 = 2 cycles 101 = Not used
010 = 3 cycles 110 = Not used
011 = 4 cycles 111 = Not used

[11] RAS to CAS delay (Trc)
0 = 1 cycle 1 = 2 cycles

[13:12] RAS pre-charge time (Trp)
00 = 1 cycle 01 = 2 cycles
10 = 3 cycles 11 = 4 cycles

[22:14] Base point of DRAM x (Base Pointer)
DRAM bank start address

[31:23] End point + 1 of DRAM x (Next Pointer)
DRAM bank end address + 1

Figure 4-9. DRAM Control Registers (DRAMCON 0,1)

KS32C6200 provides the CAS Before RAS (CBR) refresh and Self-refresh Mode. The refresh control register (REFCON) determines refresh mode, refresh timings, refresh intervals as well as external bus enable.

Register	Offset Address	R/W	Description	Reset Value
REFCON	0x00003024	R/W	DRAM refresh control	0x00000001

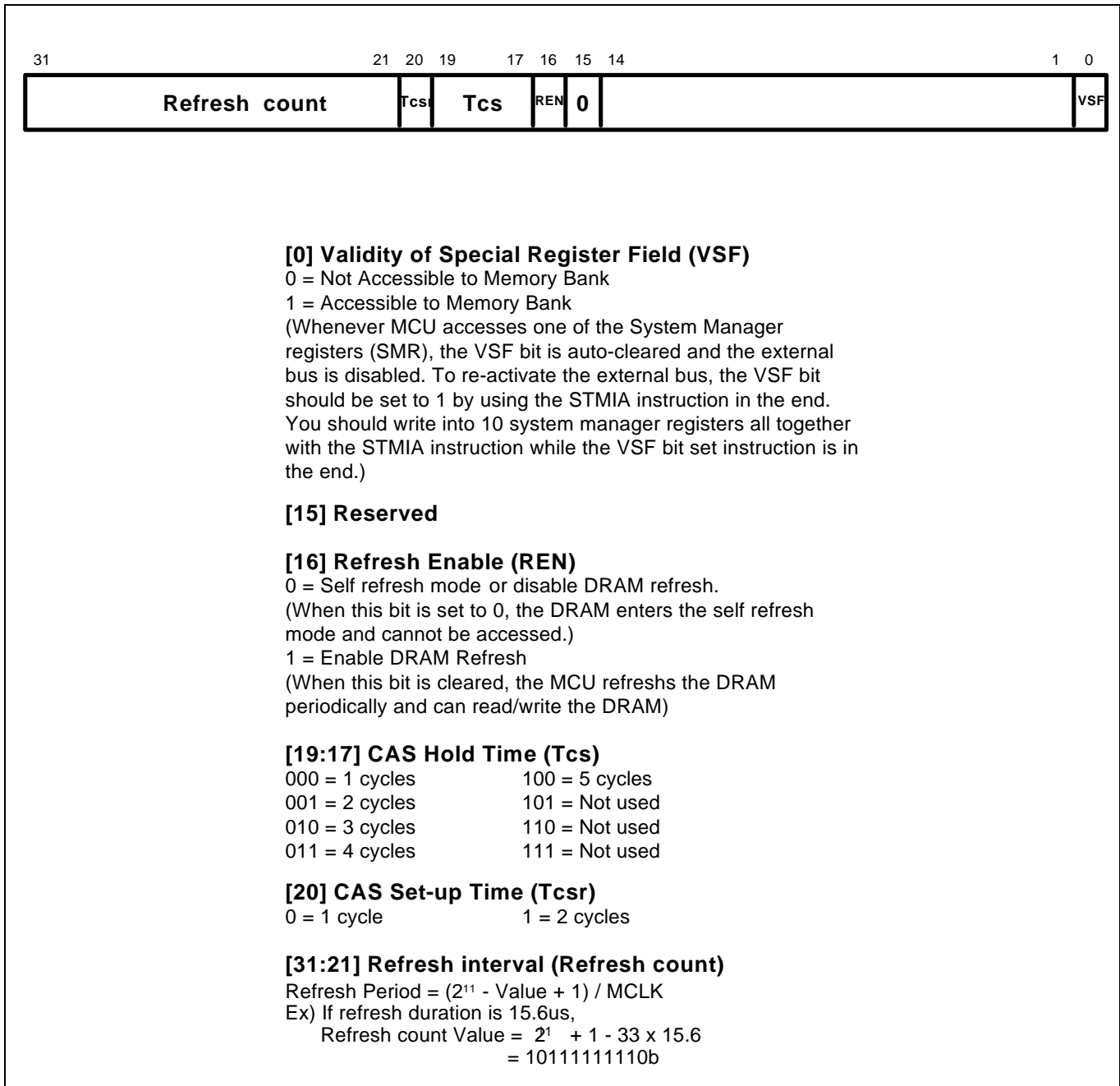


Figure 4-10. DRAM Refresh Control Register (REFCON)

DRAM SELF-REFRESH MODE

DRAM requires a refresh operation periodically to keep data correct, and JEDEC defines a couple of refresh modes. The self-refresh mode, which is defined in JEDEC specification, enables the DRAM to refresh memory cells internally without periodical external refresh control signals, unless another refresh mode happens or power fails.

The operation of self-refresh is similar to that of CBR (CAS before RAS) refresh. Once after CPU generates CBR mode signals and keeps the CBR mode state for more than 100 us, DRAMs recognize refresh mode as self-refresh mode instead of CBR refresh.

self-Refresh mode by Hardware

When a nRESET, system reset pin, is low, the system manager block generates self-refresh mode signals. For example, whenever the KS32C6200 is initialized, it activates self-refresh mode. This hardware refresh feature enables the system to avoid DRAM data loss if a system back-up circuitry supplies power to DRAM continuously while main power is disconnected.

When the main power of the system is disconnected, the KS32C6200 will be disabled. Meanwhile, if DRAM has power back-up circuitry, it still requires periodical refresh signals from the KS32C6200. Therefore, if KS32C6200 does not make DRAM self-refresh mode, it loses valid DRAM data in a short time.

For this reason, when the main power of the system is disconnected and nRESET goes to the low signal, the system manager block of KS32C6200 makes self-refresh signals. You can make memory back-up systems easily by utilizing this feature, if only DRAM is used for system memory.

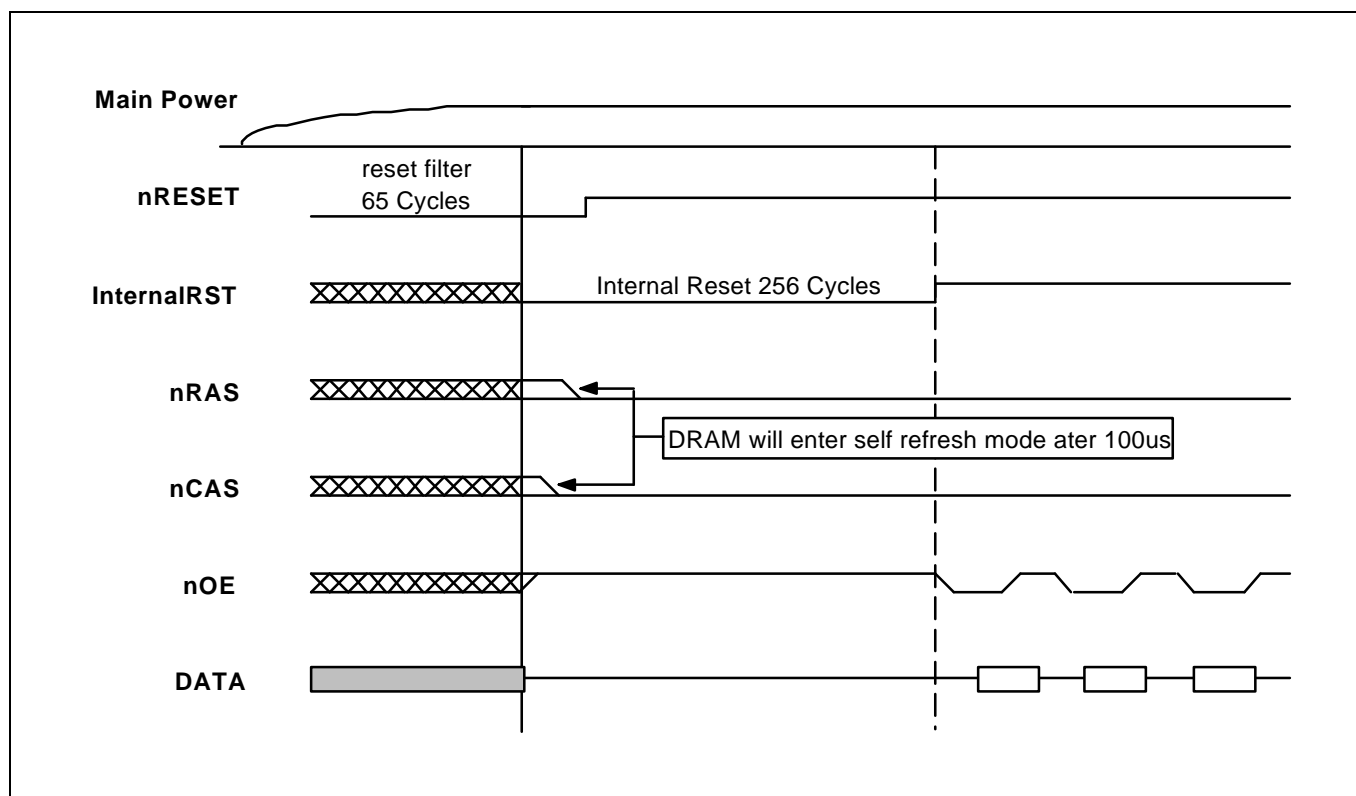


Figure 4-11. self-Refresh Mode Entry Process by nRESET (Power-On)

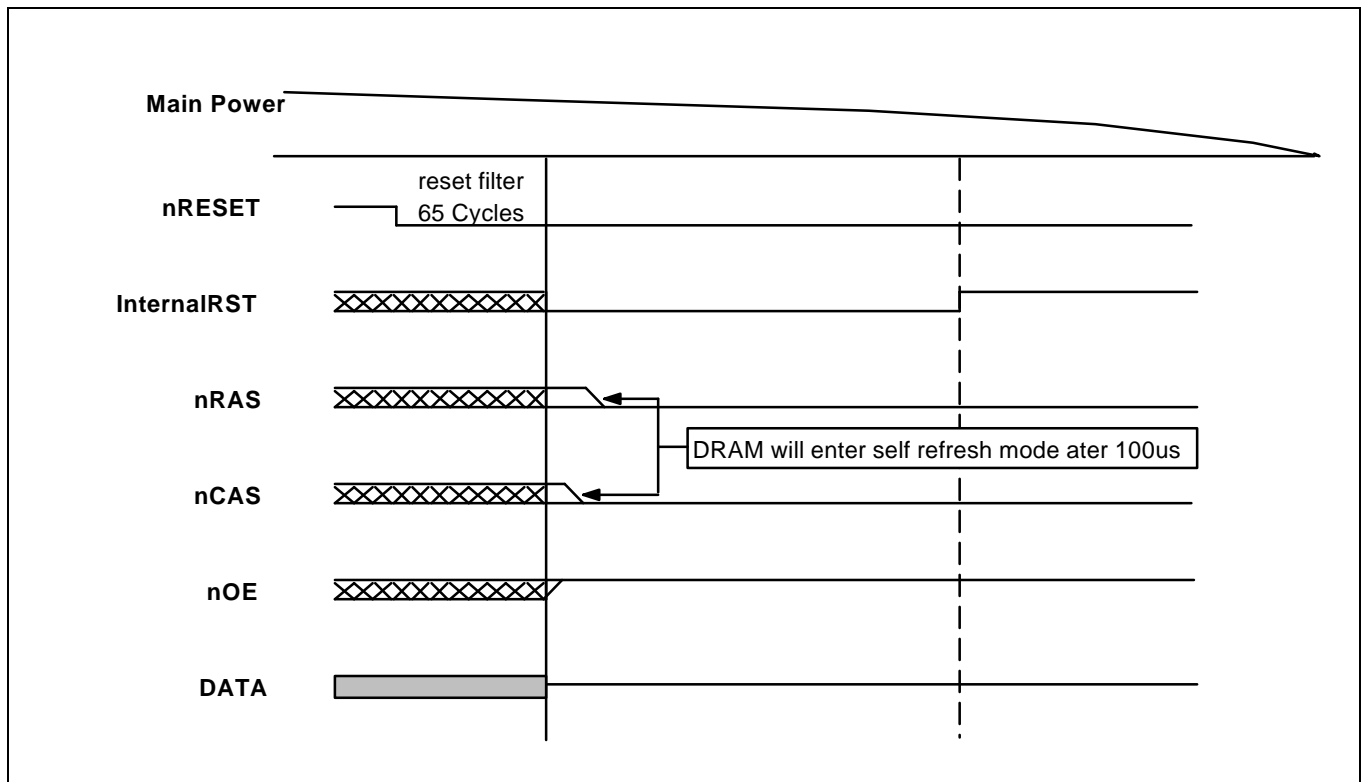


Figure 4-12. Self-Refresh Mode Entry Process by nRESET (Power-Off)

Self-Refresh mode by software

After a system reset, KS32C6200 goes into the DRAM self-refresh mode. By setting the REN bit of DRAM refresh control register to "1", you can make the system manager block work as a normal DRAM access mode.

To enable the self-refresh mode during a normal DRAM access mode, you need to change the REN bit to "0". System manager detects the value of the REN bit, changing the value from 1 to 0, and then activates the self-refresh mode. If you want change the mode from self-refresh mode to normal DRAM access mode, just write "1" to REN bit once again.

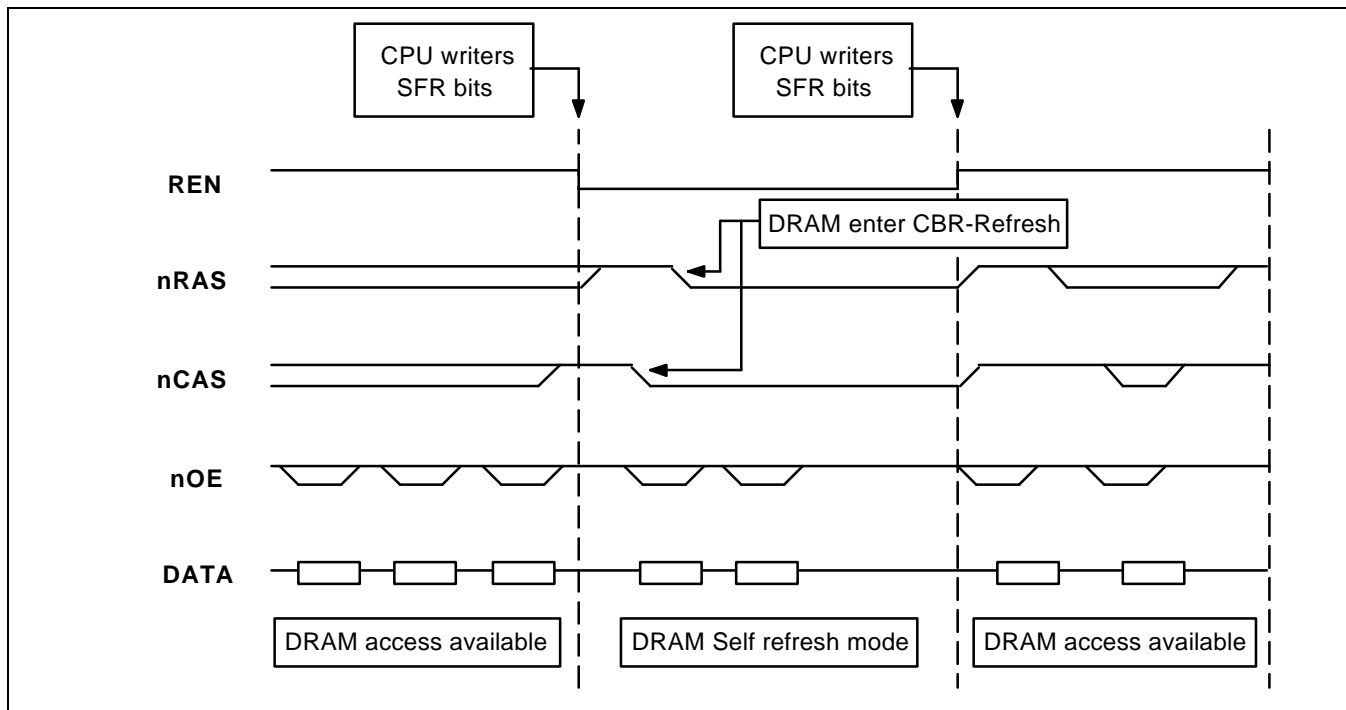


Figure 4-13. Self-Refresh Mode Entry Process by Software

NOTE: When DRAM does not recognize self-refresh mode after a system initialization

Even though KS32C6200 activates self-refresh mode when the system power is connected, DRAM may not recognize self-refresh mode correctly, because of unstable state of control signals during a system initialization—most of DRAMs recognize the self-refresh mode very well when power is switched on. If the DRAM does not go into the self-refresh mode and nRAS, nCAS are low level, the DRAM outputs data with the assertion of OE signal. In this case, KS32C6200 may fetch corrupted data from external memories because DRAM always outputs data with the assertion of OE signal.

KS32C6200 has a watch-dog timer to cope with a system malfunction problem. When KS32C6200 is initialized, the watch-dog timer is enabled and makes the external system reset signal unless MCU disables the watch-dog timer in the middle of an operation. Therefore, it is recommended that you put the code, which disables the watch-dog timer, into the boot ROM area. If "power on initial" does not work correctly and KS32C6200 fetches corrupted data, the watch-dog timer will make a system reset signal, causing the system reset of KS32C6200 once again. The second watch-dog reset will cause DRAM to enter the self-refresh mode since system power and other states are stable.

If KS32C6200 accesses DRAM to read or write data during the self-refresh mode, the nRAS and nCAS signals do not work at all, because the DRAM accessing during the self-refresh mode causes corrupted data to be read or written.

Memory access is forbidden when the SMR is changed.

The external bus is disabled when MCU accesses SMRs to change system memory configurations. It is intended to prevent the system malfunction, caused by memory address space overlapped during the new configuration. To reactivate external bus operation, the VSF bit in the refresh control register needs to be set to logic 'one' by writing SMRs with STMIA instruction. While STMIA instruction writes 10 registers of SMRs, the refresh control register must be written at the last step with VSF bit = "0" so that the external bus can be reactivated right after the System Manager Register has a new configuration.

It is not recommended for you to change the SFRs after a system initialization. If the SFRs is changed, especially memory related areas, you have to flush the cache memory for data coherency.

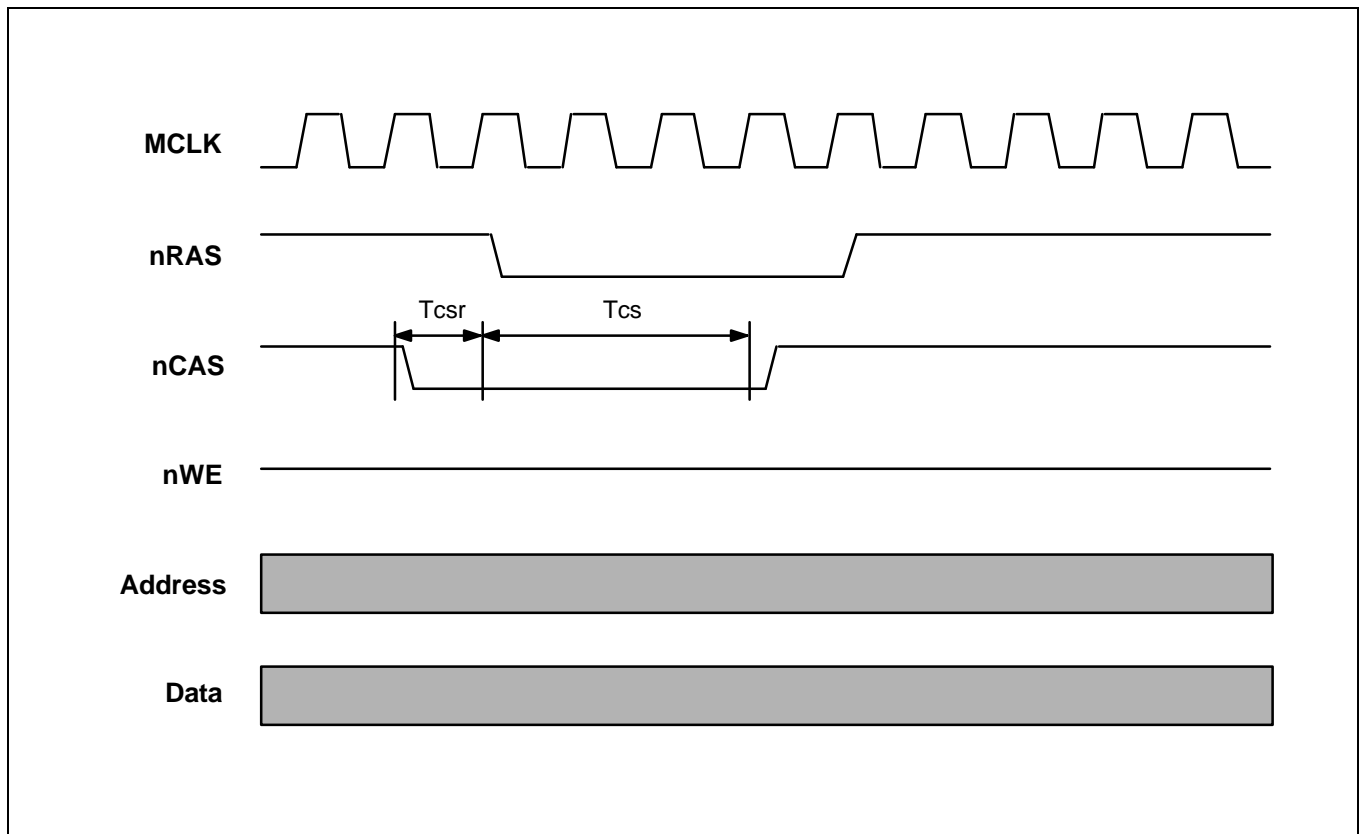


Figure 4-14. DRAM Refresh Timing

EXTRA-BANK ACCESS CONTROL REGISTERS

The KS32C6200 provides four extra-banks and four EXTCONn (extra-bank Control registers) which control timing, bank size and bus width. Extra-bank 3 has special features compared with other extra-banks. It has two special dedicated addresses (refer to two SRAM control registers) to provide the low cost external I/O control solution. Extra-bank 3 has special signals such as nIORD0/1 and nIOWR0/1. When you read/write data from/to external latch devices in extra-bank 3, the extra-address decoding ICs are not required any more. Basically, they have the same timing diagram as extra-bank 3.

The initial address of each I/O control register is the sum of its own offset address with the initial SYSCFG register address, 01000000h.

Register	Offset Address	R/W	Description	Reset Value
EXTCON0	0x300c	R/W	extra-bank 0 control register	0x00000000
EXTCON1	0x3010	R/W	extra-bank 1 control register	0x00000000
EXTCON2	0x3014	R/W	extra-bank 2 control register	0x00000000
EXTCON3	0x3018	R/W	extra-bank 3 control register	0x00000000

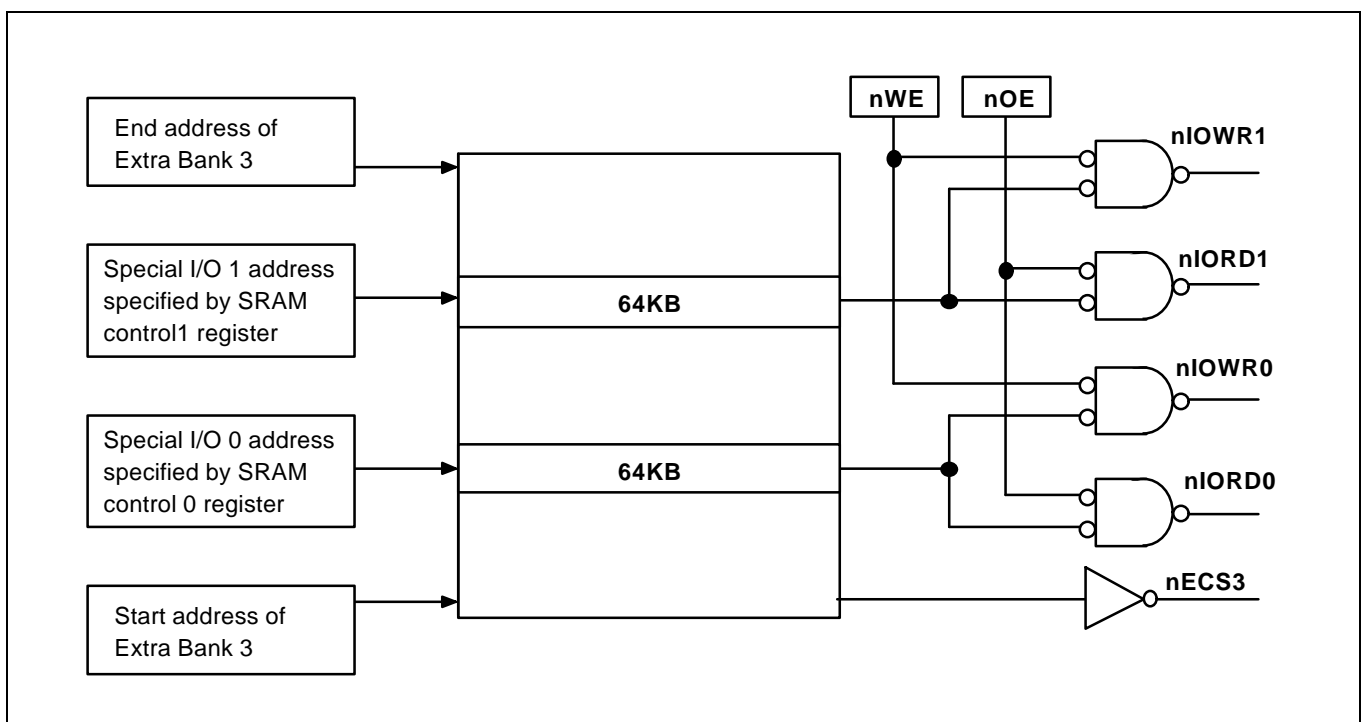


Figure 4-15. Special I/O Address Map

Fetching data from read cycle

When fetching data, the point of data reading is the last down edge of MCLK within the nECS active region. Users may be curious about the figure 4-23, nOE's deasserting before the point of data reading. If nOE has to be deasserted after the point of data reading, use 'tcoh'=0 which defines the time between nOE's deasserting and nECS's deasserting. Setting 'tcoh' as 0, nOE is deasserted after the point of data reading as you want.

Special I/O Address

Two SRAM control registers have dedicated 9 bits each for the extra-bank 3, for providing the low cost system solution. Bank 3 has special signals, nIORD0/1, nIOWR0/1. When you read/write data from/to external latch devices, these signals prevent extra-address decoding ICs. These signals are only available at extra-bank 3. When CPU access any of the special I/O address area (64-Kbytes, 16-bit offset address) specified by SRAM Control registers, the extra-bank generates I/O Read and Write signals for the corresponding address area. Fig 4-23, 24 shows the timing diagram of special I/O read/write cycles.

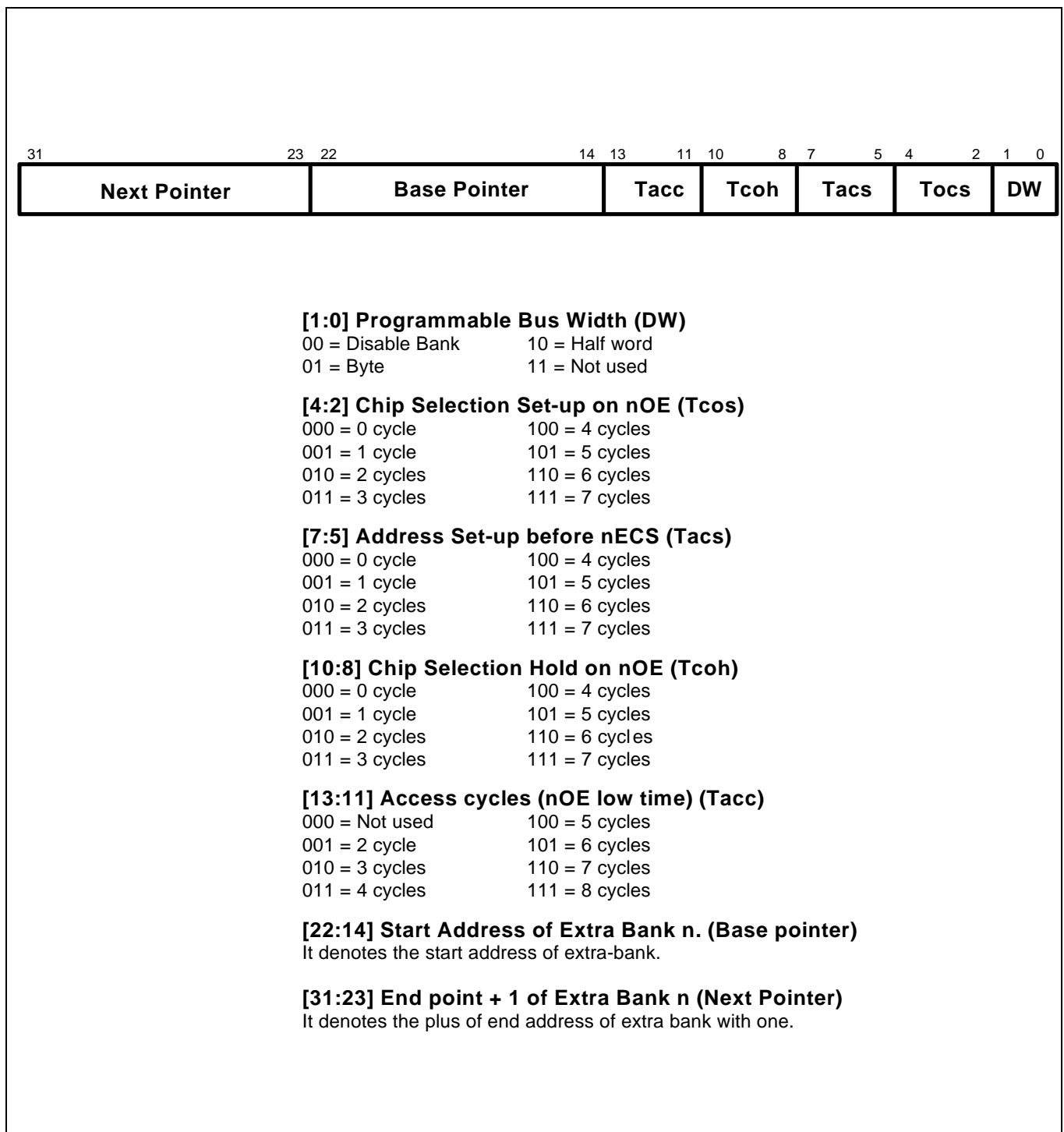


Figure 4-16. Extra-bank Control Registers (Extcntr0,1,2,3)

MEMORY MAPPING FOR EXTERNAL MEMORY AND I/O

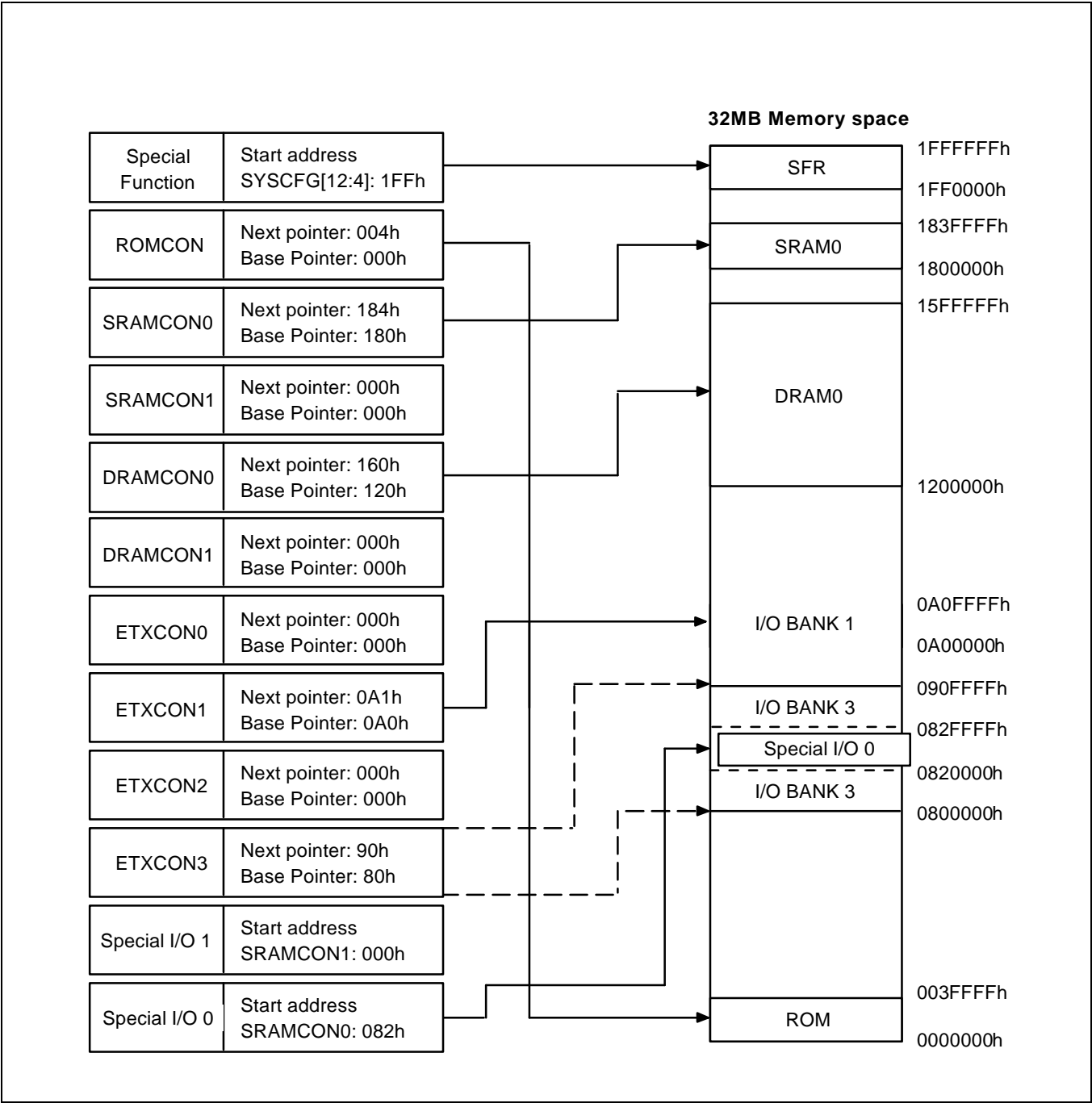


Figure 4-17. An Example of System Manager Register Settings

TIMING DIAGRAM

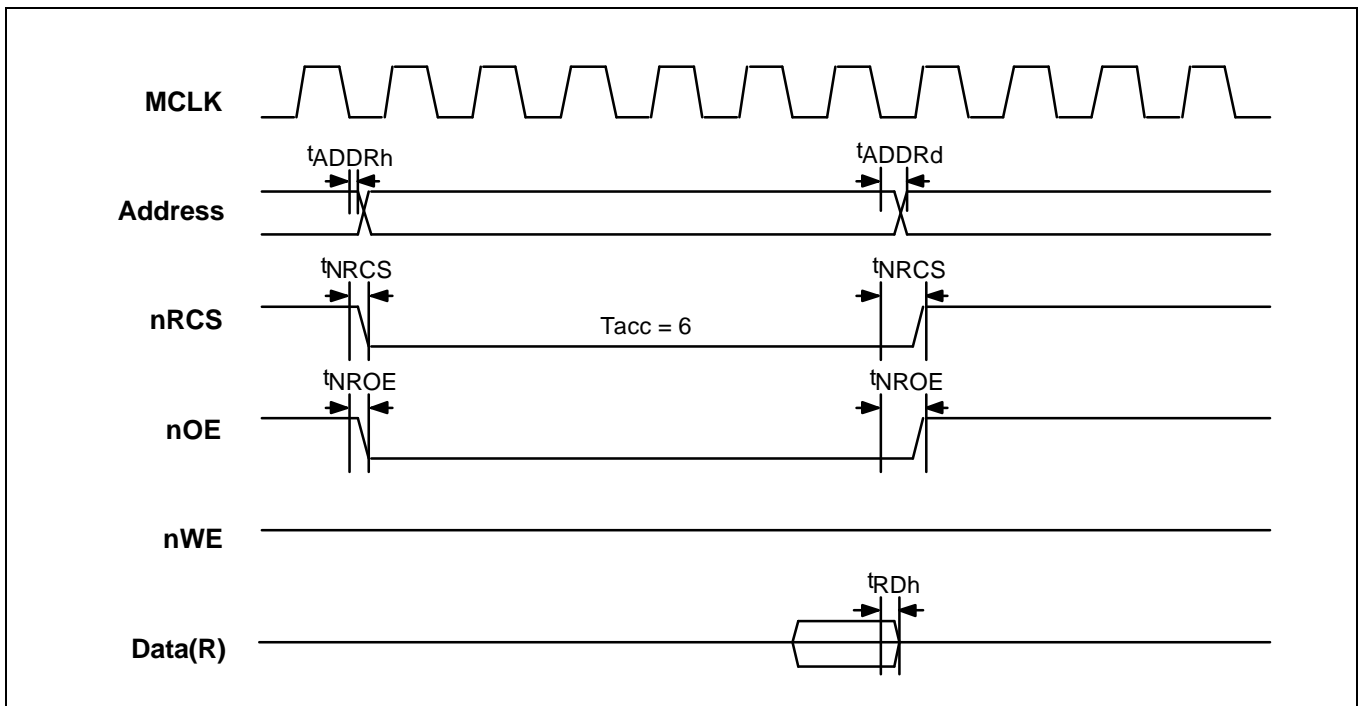


Figure 4-18. Simple ROM Access Timing

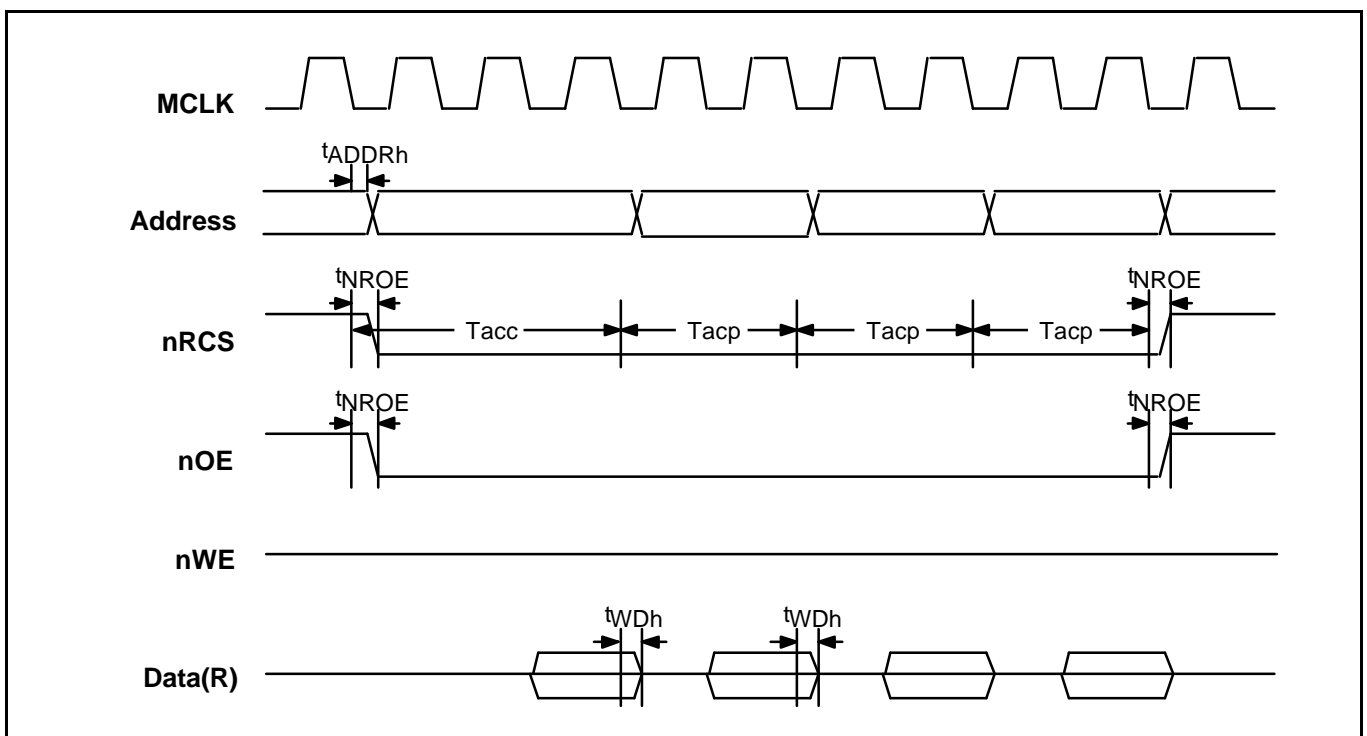


Figure 4-19. Page Mode ROM Access Timing

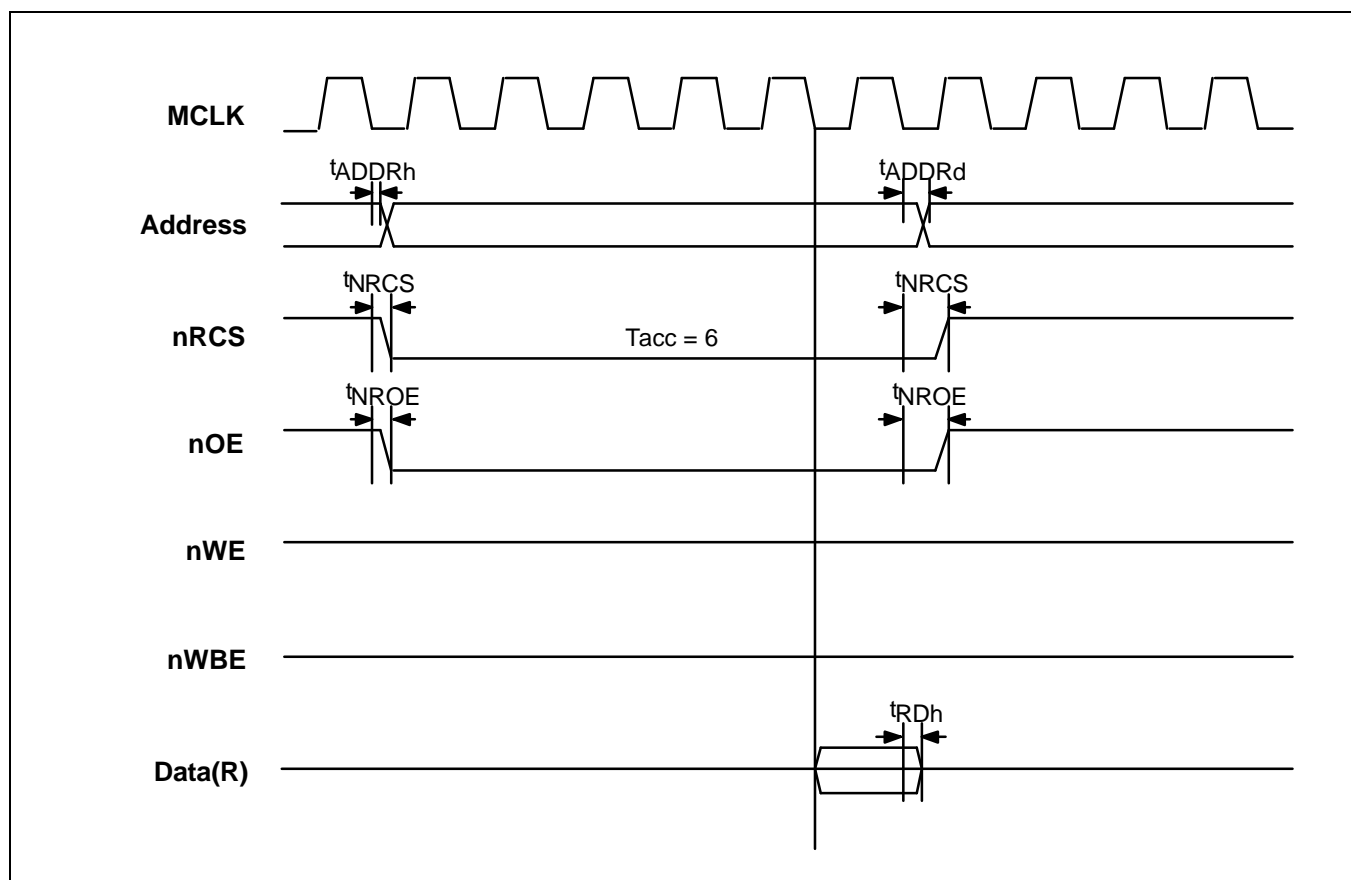


Figure 4-20. SRAM Read Timing

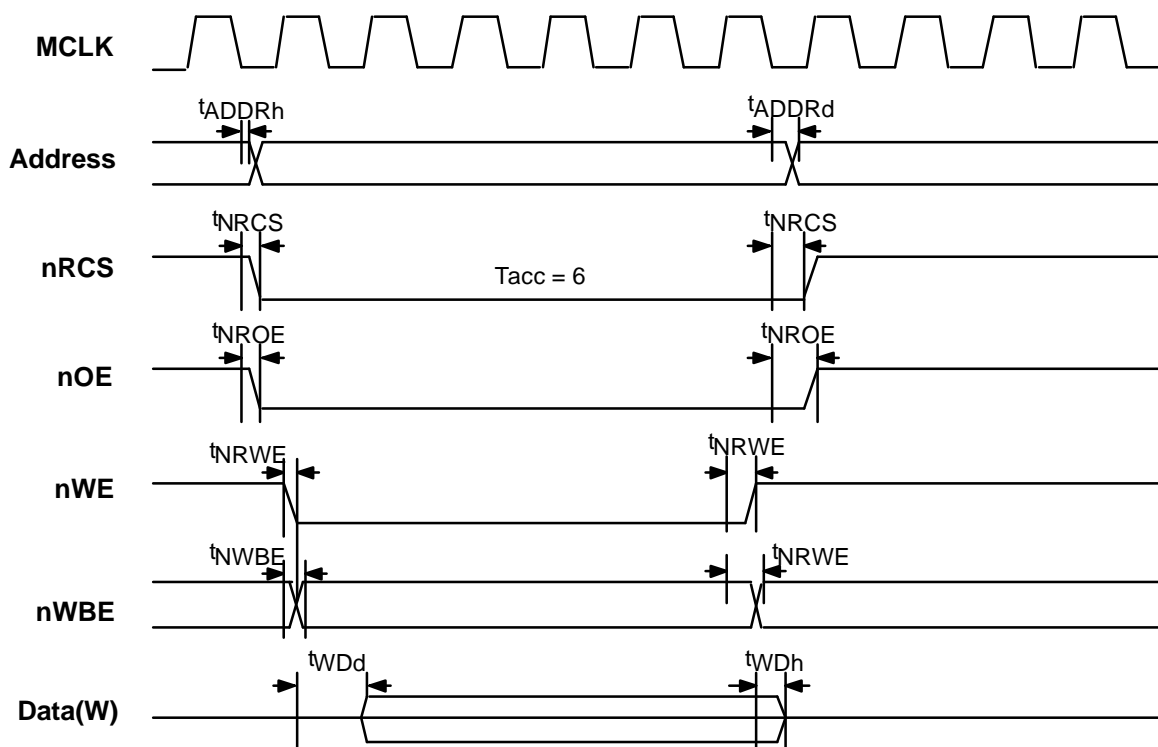


Figure 4-21. SRAM Write Timing

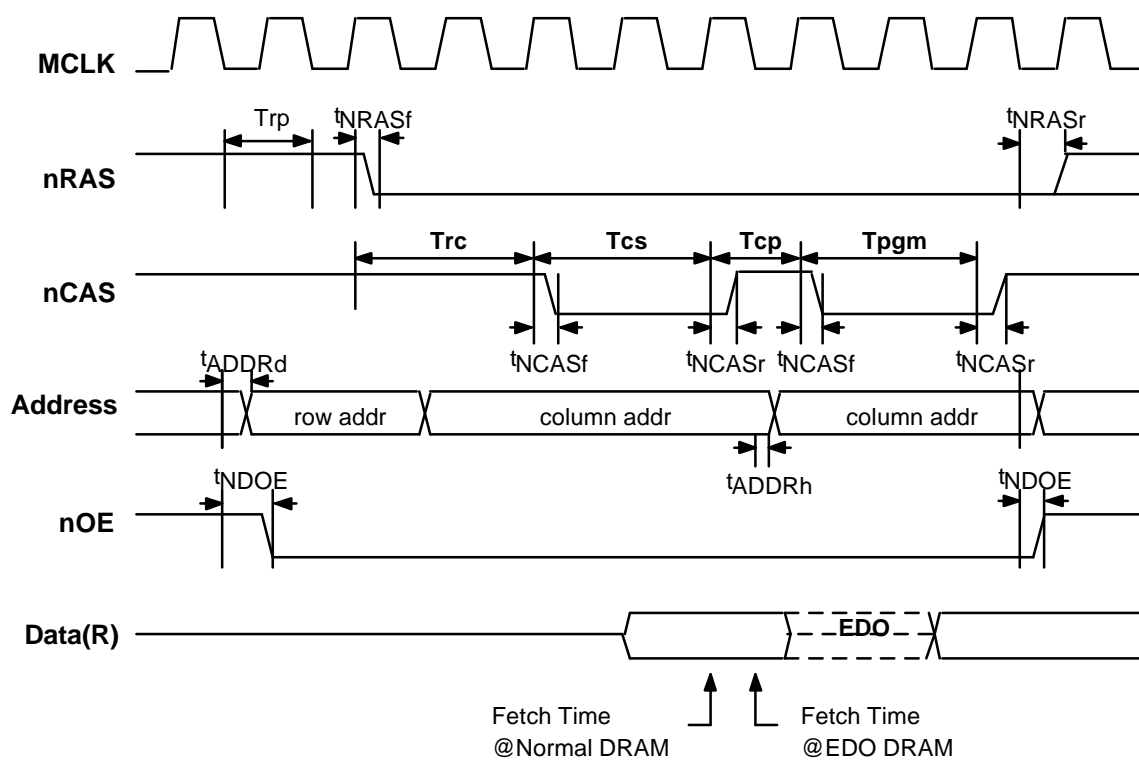


Figure 4-22. DRAM Bank Read Timing (Page Mode)

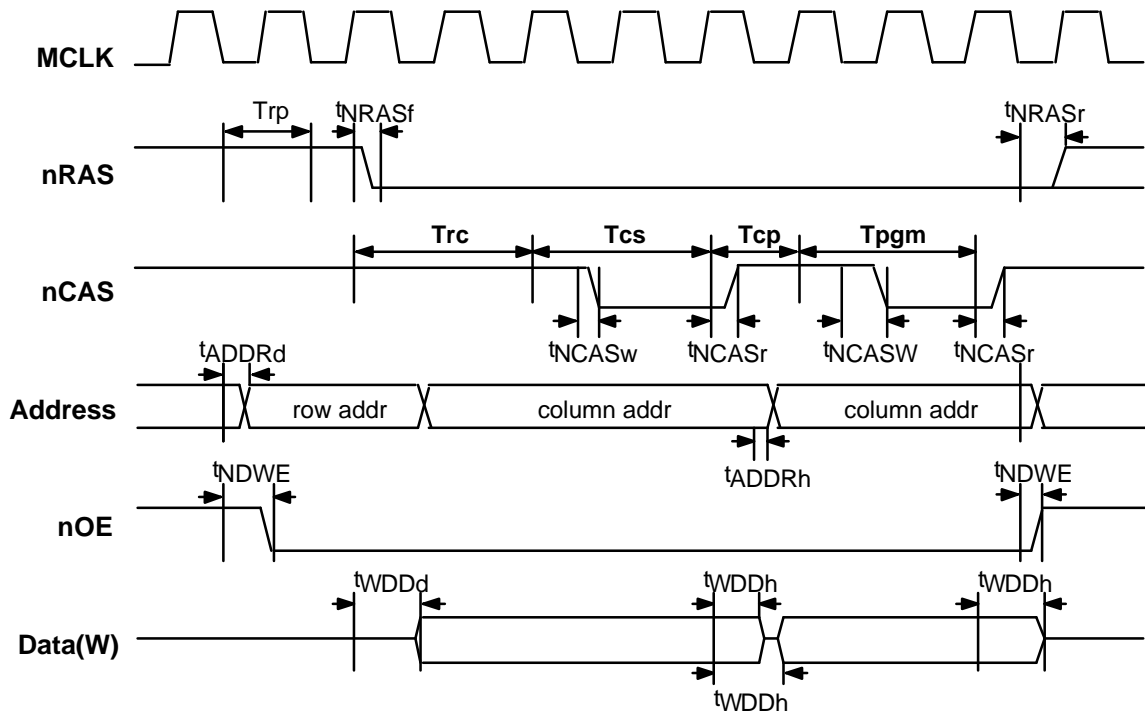


Figure 4-23. DRAM Bank Write Timing (Page Mode)

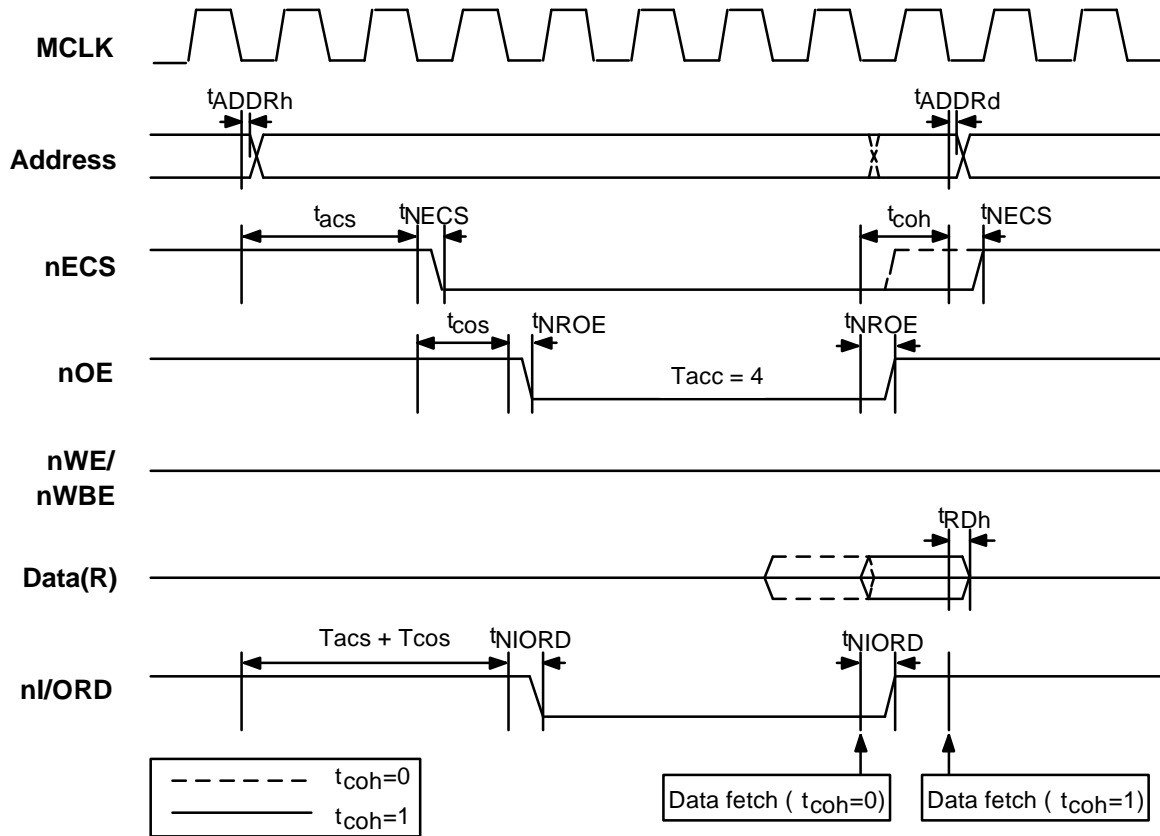


Figure 4-24. Extra-I/O Read Timing ($t_{OH}=1$, $t_{ACC}=4$, $t_{COS}=1$, $t_{ACS}=2$)

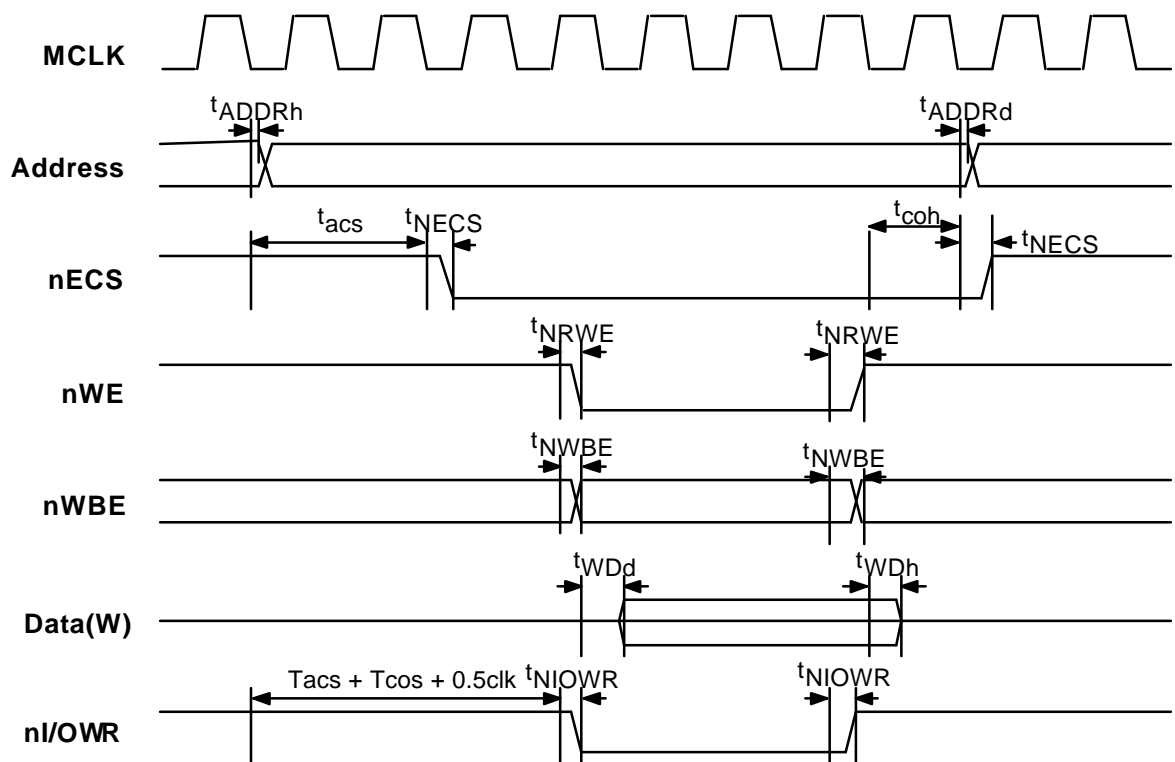


Figure 4-25. Extra-I/O Write Timing

NOTES

5

Unified (Instruction/Data) Cache

OVERVIEW

KS32C6200 CPU has a 2-Kbyte internal unified (instruction/data) cache, which adopts associative two-way set architecture with four-word (16 bytes) line size. It has a write-through policy to keep data coherency. When cache miss occurs, four words of memory are fetched sequentially from external memory. It has an LRU (Least Recently Used) algorithm to raise the hit ratio.

RISC CPU uses the instruction and data cache to improve performance. Without cache, bottleneck, which occurs during the instruction and data fetches from the external memory, may seriously degrade performance. The unified cache deals with instruction and data without distinguishing them.

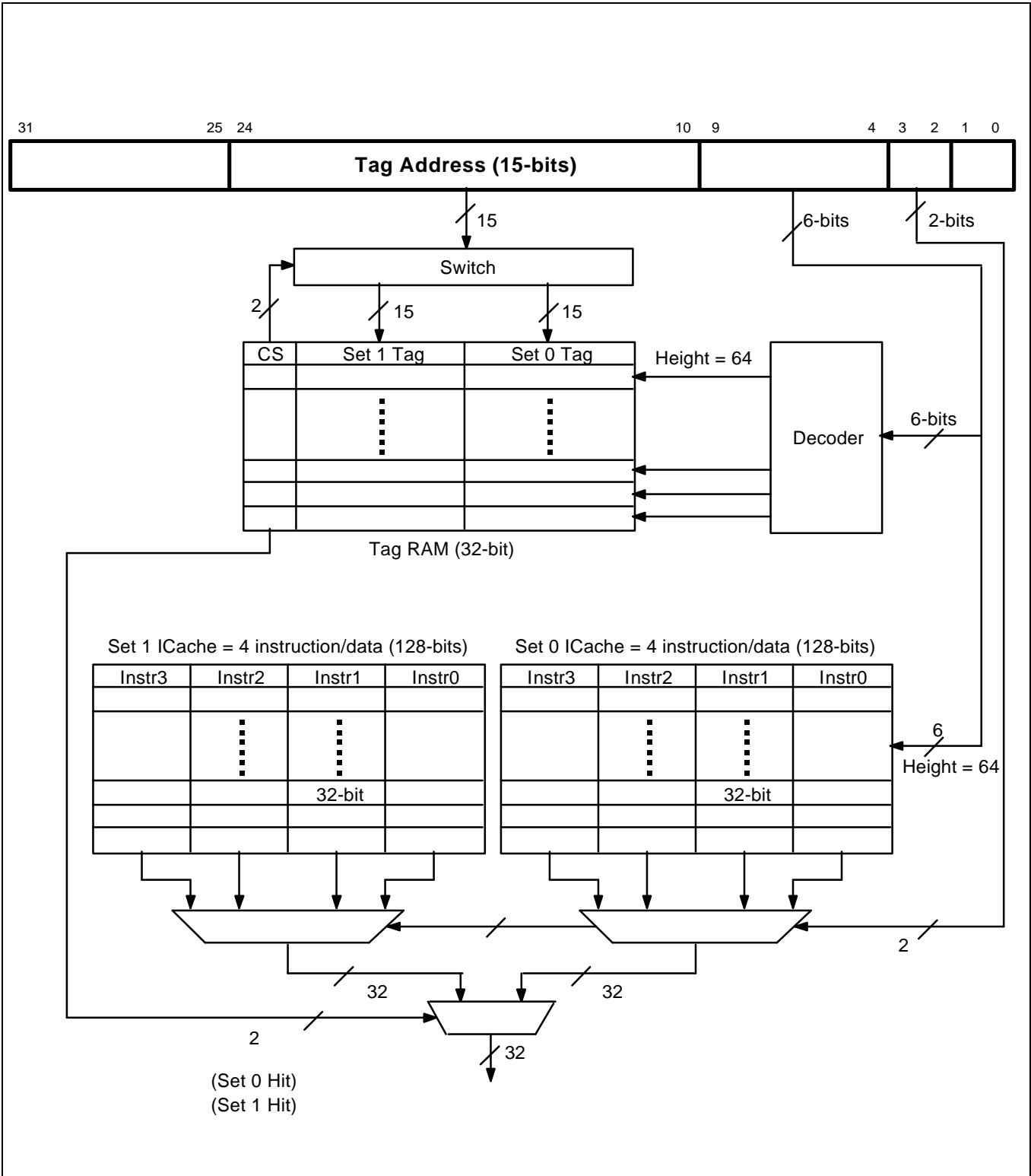


Figure 5-1. Cache Memory Configuration

CACHE OPERATION

Cache Organization

KS32C6200 cache has a two-Kbyte cache memory and one small Tag RAM. The Tag RAM has a 2-bit CS (Cache Status) and two set Tag memories for set 0 and 1. Each Tag set has a 15-bit address field [24:10] which is stored in the cache memory. The 2-bit CS indicates the validity of cached data of the corresponding cache memory line. It is also used for the cache replacing algorithm and for selecting the data coming from Set 0 and 1. Cache memory has two sets, Set 0 and Set 1. Each set has 64-lines and each line has four words of memory space (128-bits).

Cache Replace Operation

After a system is initialized, the value of CS is set to "00", signifying that the contents of set 0 and set 1 cache memory are invalid. When a cache fill occurs, the value of CS is changed to "01" at the specified line, which signifies that only set 0 is valid. When the subsequent cache fill occurs, the value of CS will be "11" at the specified line, which represents that contents of both set 0 and set 1 are valid. When the contents of the two set are valid and the content replacement is required due to the cache miss, the value of CS is changed to "10" at the specified line, signifying that the content of set 0 is replaced. When the value of CS is "10" and another contents replacement is required due to the cache miss, the content of set 1 will be replaced by changing the value of CS to "11".

Conclusively, at normal steady state, the value of CS will be changed from 11 to 10 (10 to 11), which is information for the implementation of a 2-bit pseudo LRU (Least Recently Used) replacement policy.

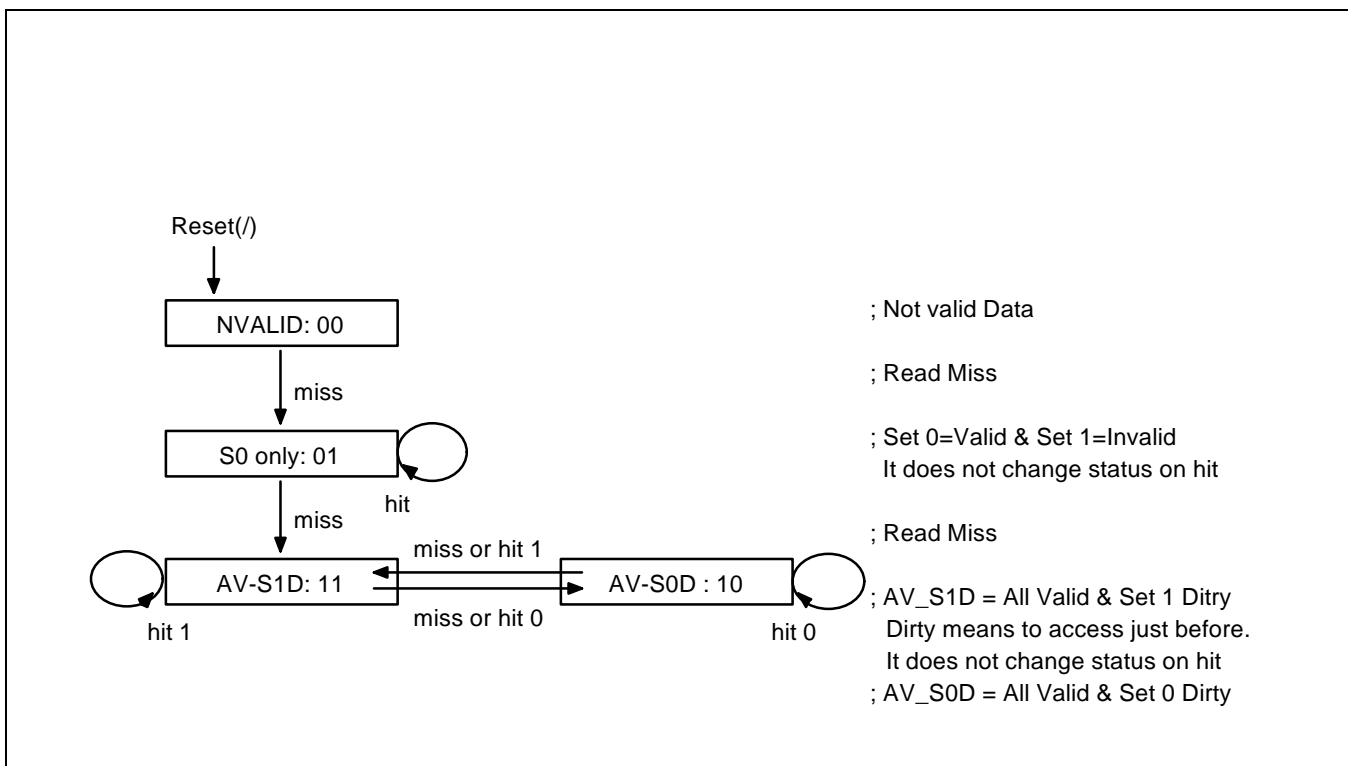


Figure 5-2. CS-Bit Status Diagram

Cache Disable Operation

The KS32C6200 cache provides programmable entire-cache-enable/disable mode. You can enable cache by setting the value of CE in SYSCFG to 1, and disable it by clearing SYSCFG[1] to zero. When the cache disable mode is specified, instructions and data are always fetched from external memory. The KS32C6200 can also provide non-cacheable areas in cache-enable mode for some particular memory access operations, such as the DMA operation. The two non-cacheable areas are specified by four special registers to be introduced later.

You have to be cautious about data coherency when the cache memory is enabled again, because the cache memory does not have auto flush mode. You also have to be cautious whether or not DMA changes memory data. The DMA accessible memory area should be non-cacheable to keep the data coherency. To keep data coherency between cache and external memory, KS32C6200 uses the write-through method.

Write Buffer Operation

KS32C6200 has four Write Buffer Registers to enhance memory writing performance. When Write Buffer mode is enabled, the CPU writes data into the write buffer instead of an external memory when the external bus is already occupied by another bus master like that of DMA. The Write buffer has 4 registers and each register includes a 32-bit data field, a 25-bit address field and a 2-bit status field. The system manager executes all operations of the write buffer

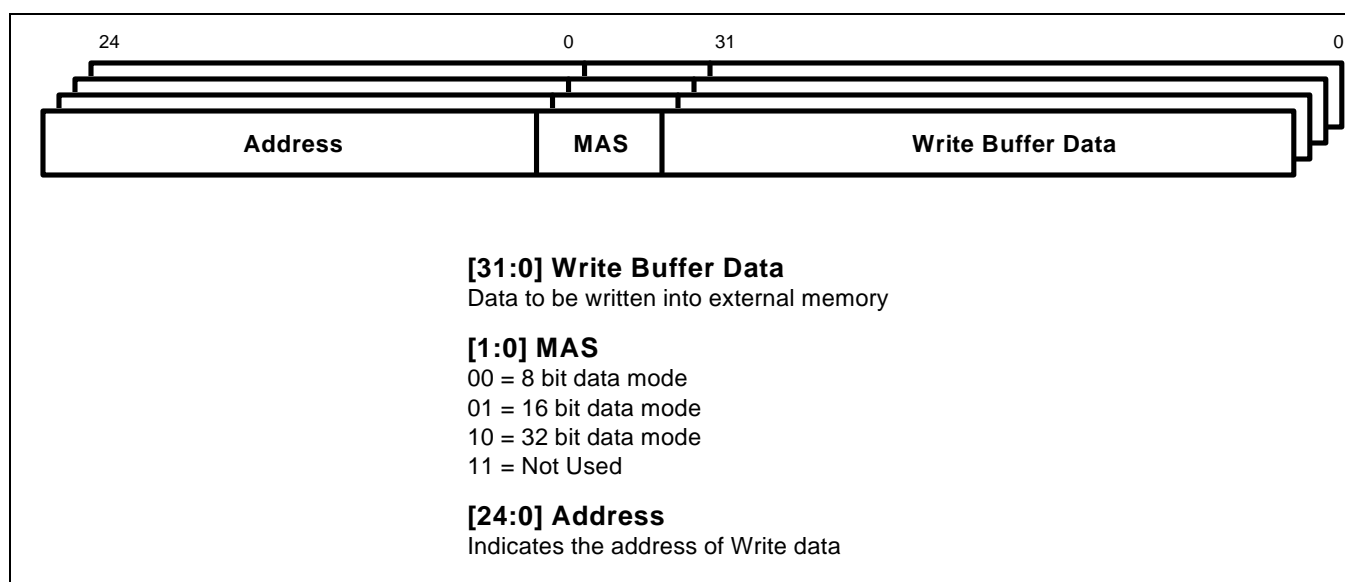


Figure 5-3. Write Buffer Configuration

Cache Flushing

A cache flushing is needed to enable the cache operation again. When the cache is disabled, the Tag RAM and cache memory, set 0 and set 1, can be manipulated exactly like normal memory. You can flush the cache by writing zero to the Tag RAM and making all data of the cache invalid. The structure of the Tag RAM and cache memory is shown in Figure 5-l. The memory location of the Tag RAM and set 0,1 cache memory is as follows:

Tag RAM	0x11000000–0x110000ff
Set 0	0x10000000–0x100003ff
Set 1	0x10800000–0x108003ff

NOTE: Cache flushing must be executed only in the cache disable mode.

CACHE CONTROL REGISTERS

KS32C6200 cache provides two non-cacheable areas. It has four Cache Control registers to specify two non-cacheable areas. Usually a cache stores any data in the whole system memory area, but sometimes it needs a non-cacheable region to keep the data consistency between the external memory and the cache memory.

The KS32C6200 provides two non-cacheable areas and each of them requires two Cache Control Registers to indicate the start and end address of the non-cacheable area. If a non-cacheable area is specified, the area is not cached when read miss occurs.

Register	Offset Address	R/W	Description	Reset Value
CACHNAB0	0x1000	R/W	Start address of non-cacheable area 0	0x00000000
CACHNAE0	0x1800	R/W	End address of non-cacheable area 0 plus one	0x00000000
CACHNAB1	0x2000	R/W	Start address of non-cacheable area 1	0x00000000
CACHNAE1	0x2800	R/W	End address of non-cacheable area 1 plus one	0x00000000

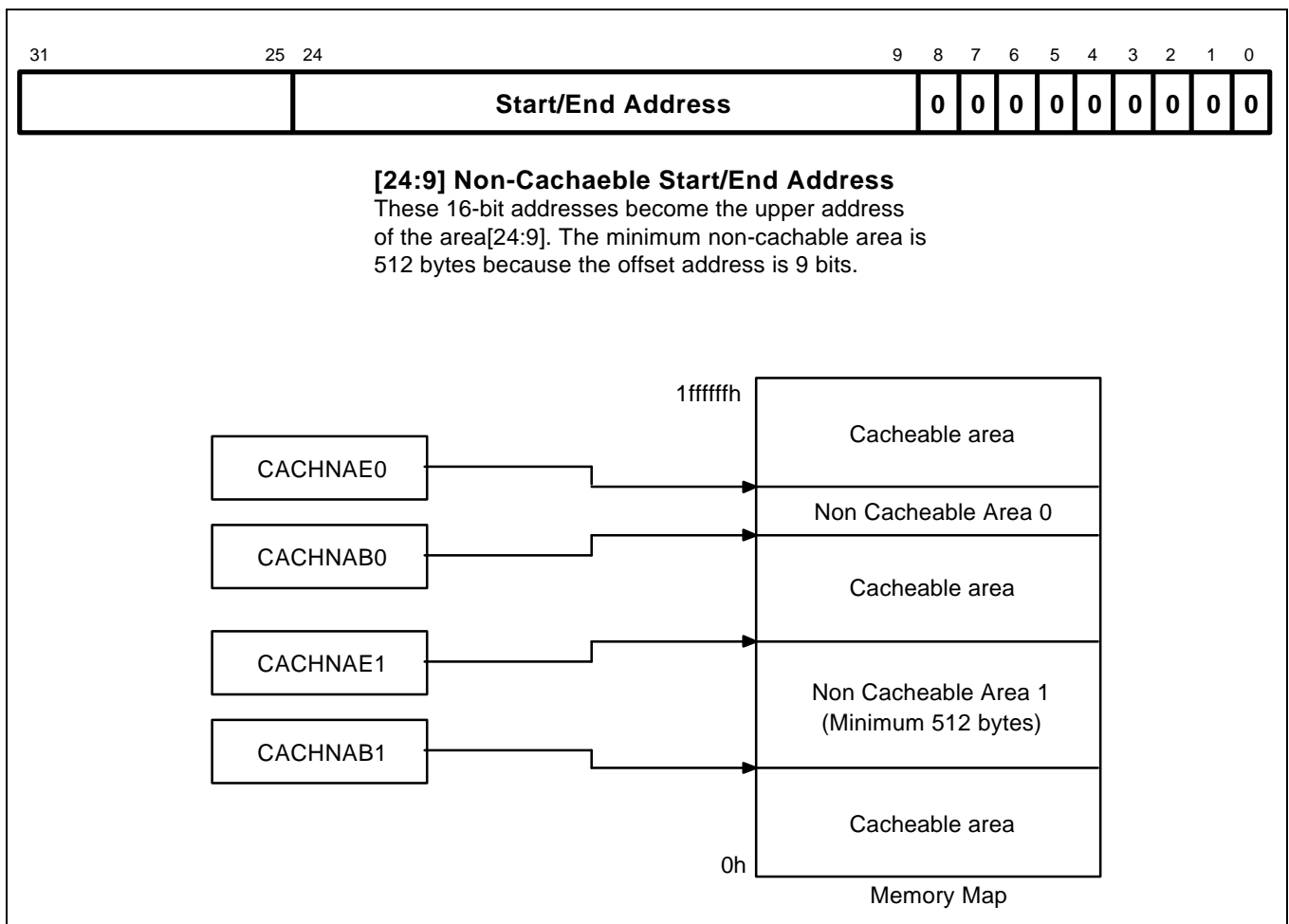


Figure 5-4. Non-cacheable Area Register

NOTES

6

Derasterizer

OVERVIEW

KS32C6200 Derasterizer provides 16 x 16 bit image data rotation feature. The derasterizer consists of 16 registers with 16-bit data width. The 16 x 16-bit register array is used to rotate the raster image data 90 or 270 degree.

Register	Offset Address	R/W	Description	Reset Value
DRAST0	0x00006000	R/W	16-bit derasterizer data register 0	Undefined
DRAST1	0x00006004	R/W	16-bit derasterizer data register 1	Undefined
...
DRAST14	0x00006038	R/W	16-bit derasterizer data register 14	Undefined
DRAST15	0x0000603c	R/W	16-bit derasterizer data register 15	Undefined

NOTE: When h[15:0] is written and v[15:0] is read, the address of DRAST0–DRAST15 is read.

Rotation

To rotate image data, you should fill image data into the 16 x 16 bit register array from DRAST0 to DRAST15, horizontally. The image data, which is made by reading the 16 x 16, has the rotated image. You can change the rotation direction by manipulating the shift control register, SFTCON[3]. When SFTCON[3] is 0, the read image data is rotated by 90 degree and when SFTCON[3] is set to 1, the image data is rotated by 270 degree.

Write: h0 → h15 (DRAST0 → DRAST15)

Read: 90 degree: (horizontal direction) v0 → v15 ⇒ (vertical direction) MSB → h15, LSB → h0

270 degree: (horizontal direction) v15 → v0 ⇒ (vertical direction) MSB → h0, LSB → h15

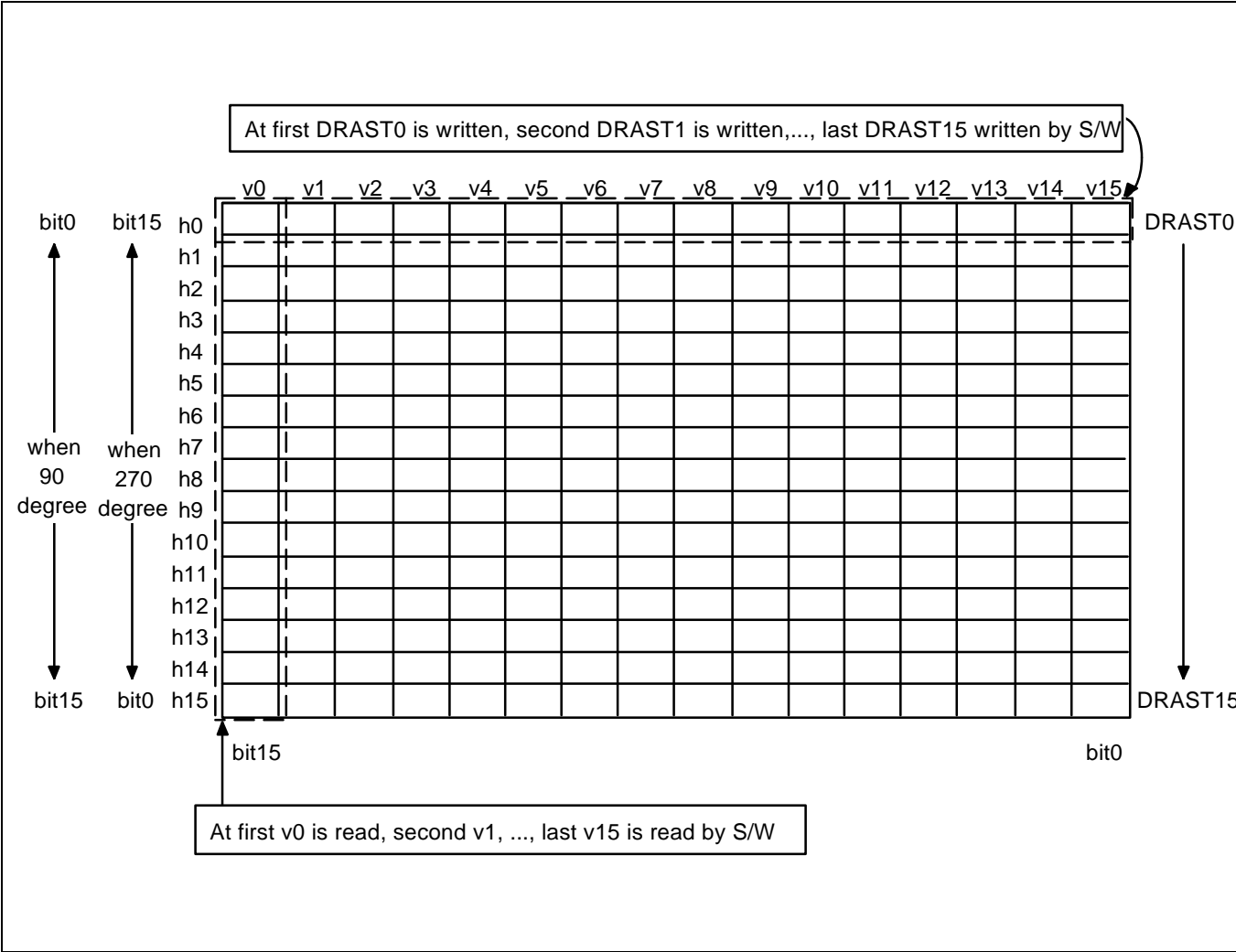


Figure 6-1. Rotate Configuration

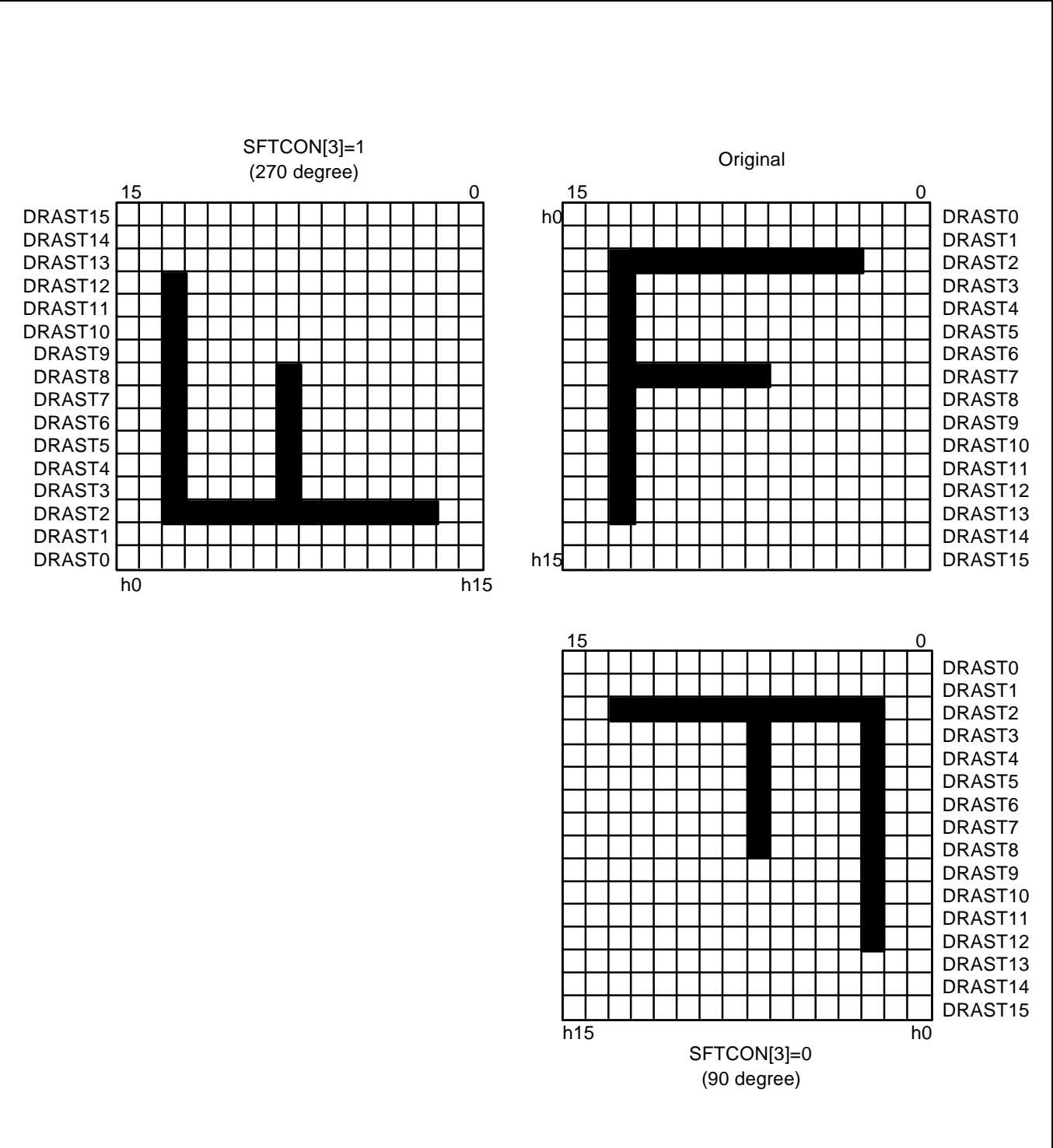


Figure 6-2. Derasterizer Rotation Example

NOTES

7

Shift Control

OVERVIEW

The KS32C6200 provides data shift and reverse (180 degree rotation) features. Shift module has one 16 bit register for data reverse operation and six 32-bit registers and one 16-bit register for data shift operation. The seven data shift operation registers have seven serially connected registers for 224-bit data shift operation.

Data Reverse Register (DATARVS)

The data reverse operation is a kind of data position exchanger. The written data bit stream exchanges its position so that the MSB of the original goes to the LSB of the reversed data and the LSB of the original goes to the MSB of the reversed data.

Normal (MSB-----LSB) => reversed (LSB-----MSB)

Data reverse operation is useful for data rotation operation. The rotator block supports only 90- and 270-degree operation. Data reverse operation behaves the same as to the 180 degree rotator operation.

Register	Offset Address	R/W	Description	Reset Value
DATARVS	0x7000	R/W	Data transfer	0000h

[15:0] Data reverse

DATARVS keeps the reversed bit stream of the data. You can get reversed data right by read operation after writing.

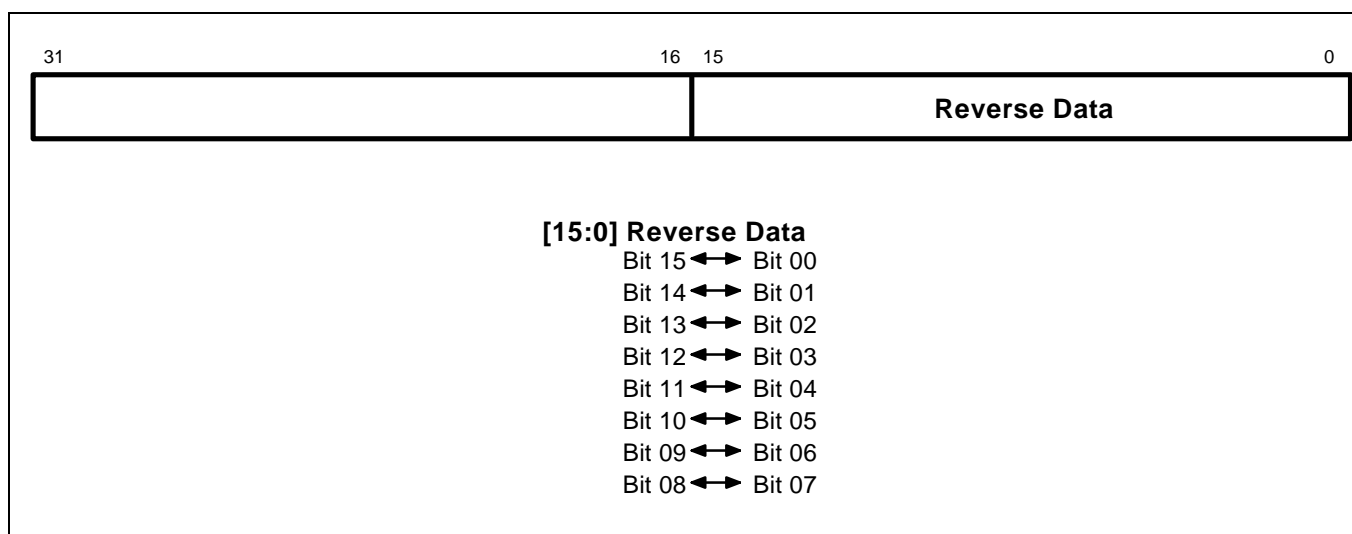


Figure 7-1. Reverser

NOTES

1. MSB is placed at bit 15 and LSB at bit 0 while writing 16-bit data to this register
2. Read the data from this register; MSB is bit 0 and LSB is bit 15.

Example:

Input -----> Reverse data register -----> Output
 "1011 0111 0000 1010" "0101 0000 1110 1101"

SHIFT CONTROL REGISTER

Shift Control Register (SFTCON) in the KS32C6200 specifies the rotation degree of the derasterizer data, the direction of data shift, the shift mode, and the status of the shift mode operation (shifting or finished).

Register	Offset Address	R/W	Description	Reset Value
SFTCON	0x7004	R/W	Shift control register	0h

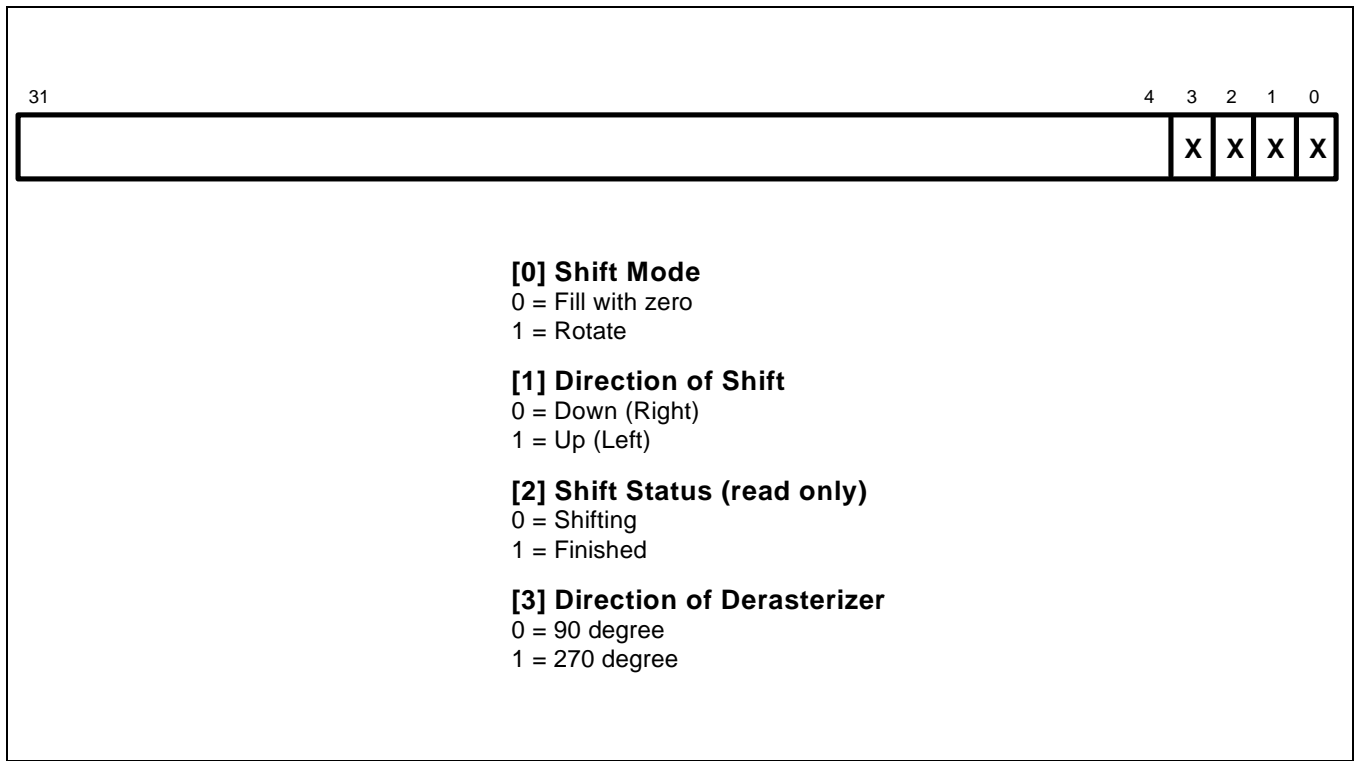


Figure 7-2. Shift Control Register

SHIFT COUNT REGISTER

Shift Count Register (SFTCNT) specifies the amount of shift operation. When a non-zero value is written into this register, the shifter starts data shifting up to the specific amount. When shift operation has finished, the SFTCON[2] is set to "1". The execution time is proportional to the amount of shift.

Register	Offset Address	R/W	Description	Reset Value
SFTCNT	0x7008	R/W	Shift count register	00h

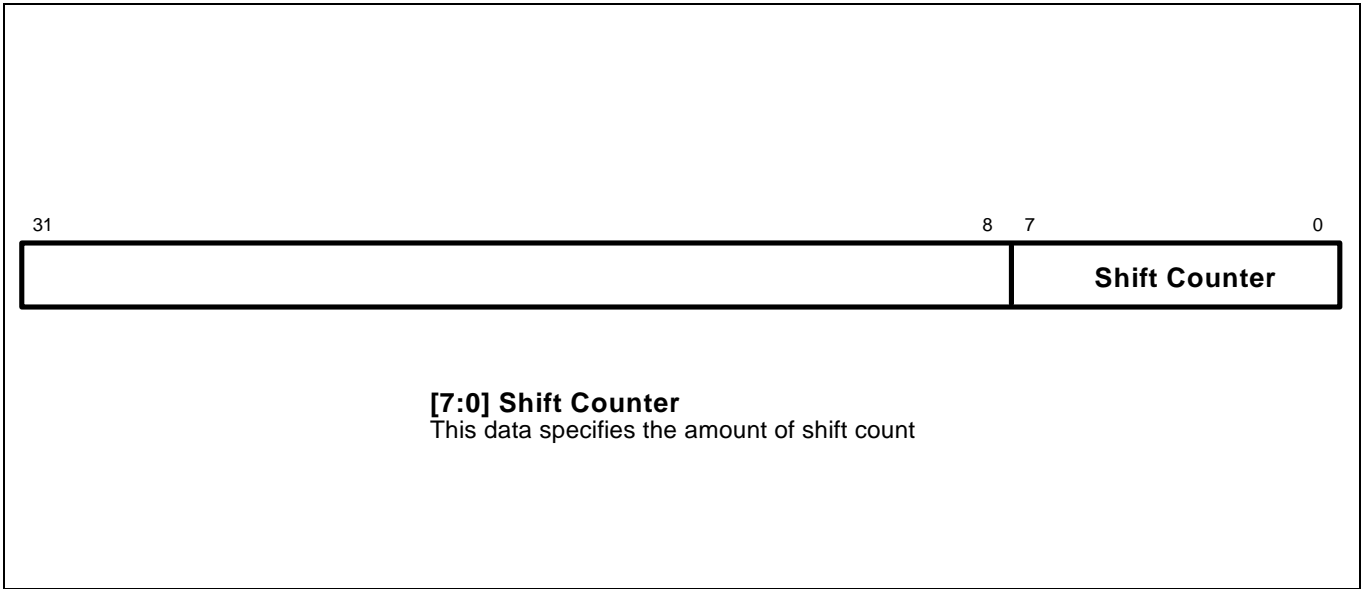


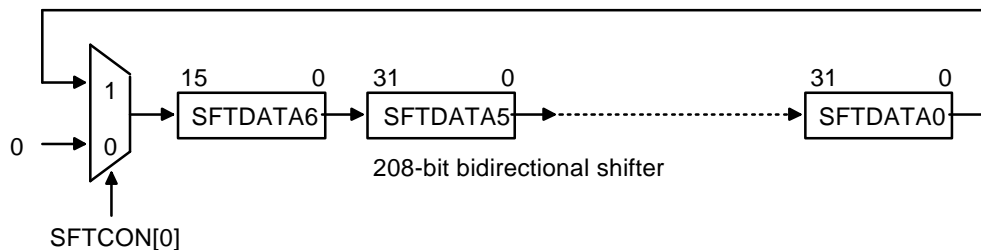
Figure 7-3. Shift Count Register

SHIFT WORD DATA REGISTER

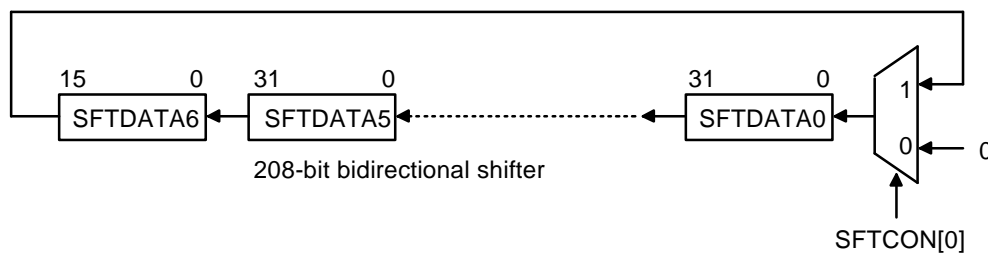
The shift control block in KS32C6200 has seven serially connected Shift Word Data Registers (SFTDATA0-6). These serially connected registers form 208-bit bidirectional shift register block. The shift operation mode can be manipulated by the SFTCON register. The SFTCON[0] bit controls the shift mode such as fill-with-0 mode or rotate (wrap) mode and the SFTCON[1] bit controls shift direction, left or right.

Register	Offset Address	R/W	Description	Reset Value
SFTDATA0	0x700c	R/W	Shift word data 0 register, 32 bits	Undefined
SFTDATA1	0x7010	R/W	Shift word data 1 register, 32 bits	Undefined
....
SFTDATA6	0x7024	R/W	Shift word data 16 register, 16 bits	Undefined

Example 1: Shift Right Mode (when SFTCON[1]='0')



Example 2: Shift Left Mode (when SFTCON[1]='1')



When programming:

1. Load values into SFTDATA0–SFTDATA6 (Shift Word Data Registers)
2. Second: Set the value of SFTCON (Shift Control Register)
3. Third: Set the value of SFTCNT (Shift Counter Register)

NOTES

8

Timer

OVERVIEW

The KS32C6200 has three 16-bit timers. The three timer blocks share an 8-bit prescaler and a clock-divider which has 4 different divided signals. Each timer block receives its own clock signals, for example, the Timer Clock, from the clock divider which receives the clock from the 8-bit prescaler. The 8-bit prescaler is programmable and divides the MCLK signal depending on the loading value which is stored in TSTCON[14:7].

The timer count value register (TBCNTn) has the initial count value which is loaded into the down-counter when the timer is enabled. Each timer has its own 16-bit down-counter which is driven by the timer clock. When the down-counter reaches zero, the timer interrupt request is generated to inform the CPU that the timer operation is completed. When the timer counter reaches zero, the value of corresponding TBCNTn is automatically loaded into the down-counter to continue the next operation. However, if the timer stops, for example, by clearing the timer enable bit of TCON during the timer running mode, the count value of TBCNTn will not be reloaded into the counter.

The timer count value register is used to define the duration of the timer operation, and contains the number of timer clocks needed for one operation duration.

The timer duration is

$\text{Timer_clock} = \text{MCLK} / (\text{prescale_value} + 1) / \text{division_factor} \text{ (Hz)}$

$\text{Timer_duration} = \text{count_value} \times \text{Timer_clock_period} = \text{count_value} / \text{Timer_clock}$

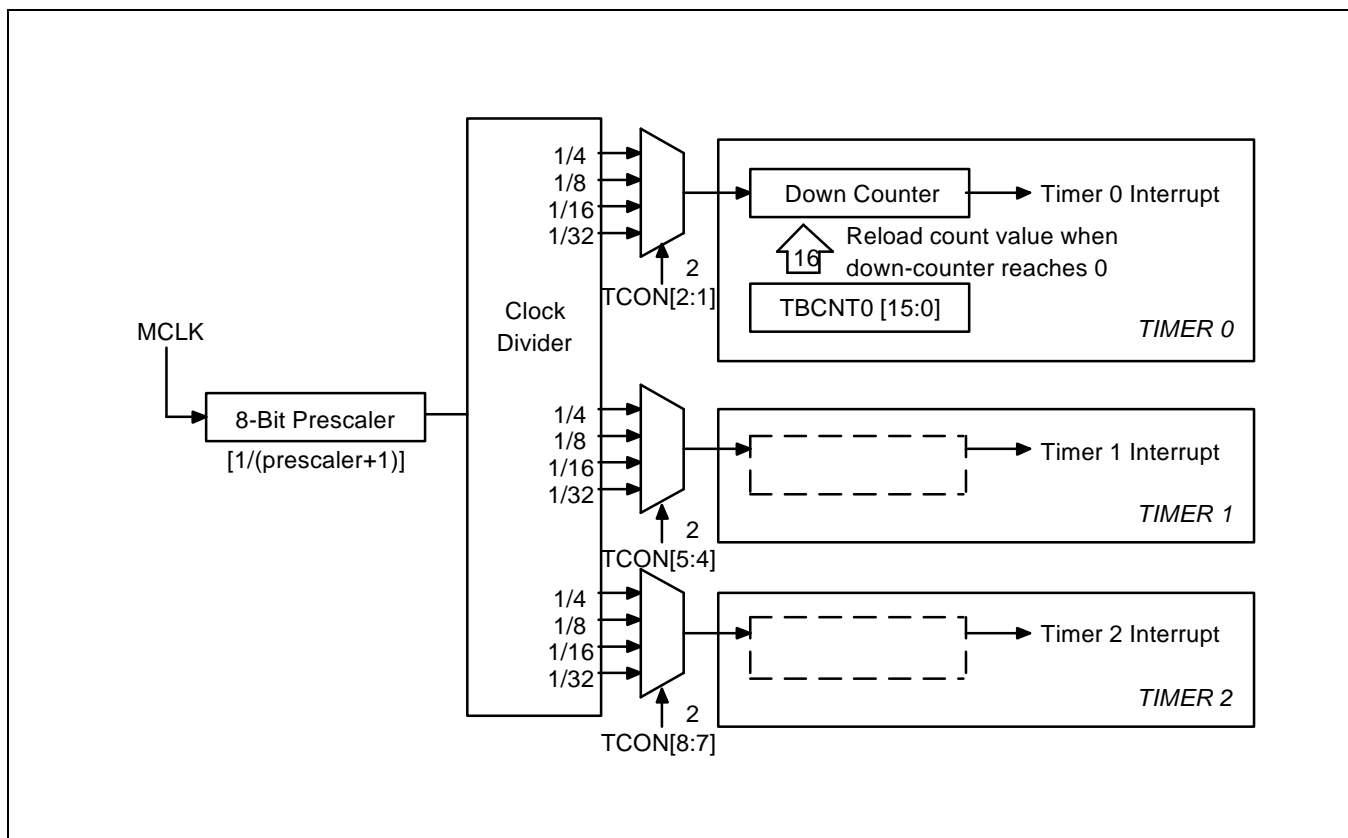


Figure 8-1. 16-Bit Timer Block Diagram

TIMER CONTROL REGISTER

You can disable or enable the timer operation and select the clock-divider output from four-divided signals by using the Timer Control Register (TCON)

Register	Offset Address	R/W	Description	Reset Value
TCON	0x5800	R/W	System timers control register	000h

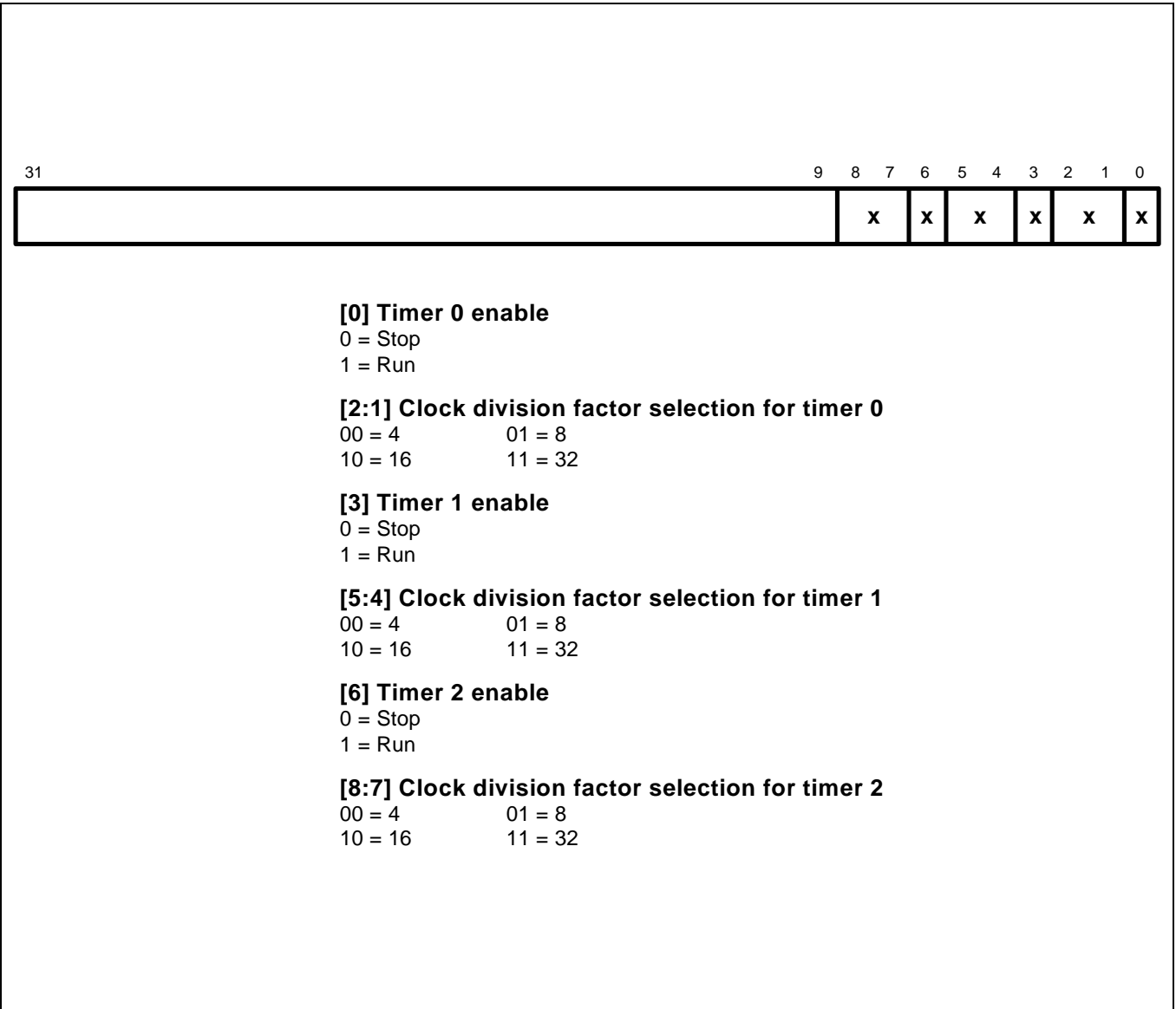


Figure 8-2. Timer Control Register

TIMER COUNT VALUE REGISTER

The timer count value registers, TBCNTn, are used to specify the time-out duration of each timers. The count value will be loaded or reloaded into the down-counter automatically when timer operation is enabled or the down-counter is decreased to zero.

Register	Offset Address	R/W	Description	Reset Value
TBCNT0	0x5804	R/W	Timer 0 count value register	Undefined
TBCNT1	0x5808	R/W	Timer 1 count value register	Undefined
TBCNT2	0x580c	R/W	Timer 2 count value register	Undefined

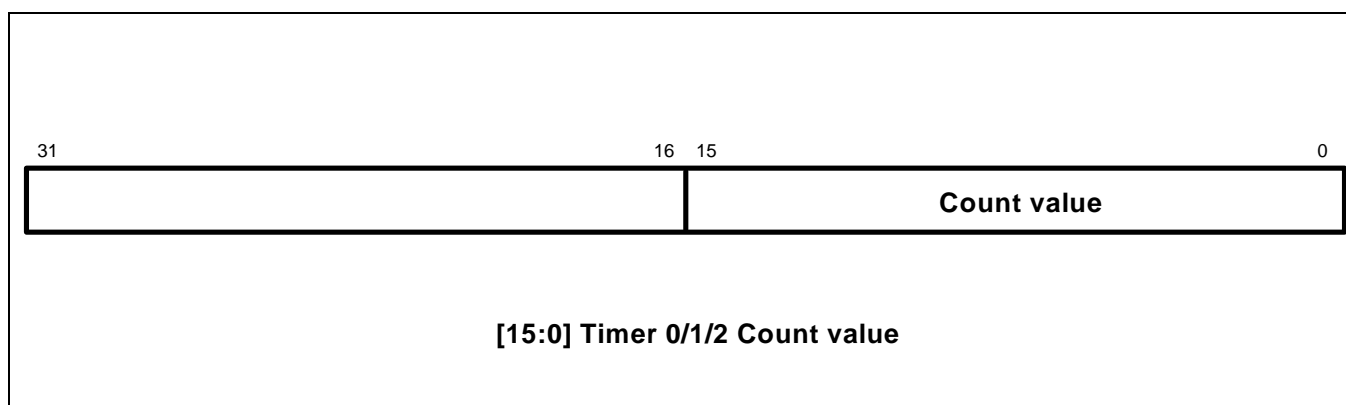


Figure 8-3. Timer Count Value Register

If you change the value of TBCNTn while the timer is running, the new value will be written into TBCNTn and the counter resumes count with the new value. The timer programming sequence is shown in Figure 8-4. The count value and the definition of timer clock, including the prescaling value and the clock division factor, should be specified before setting the timer-enable bit.

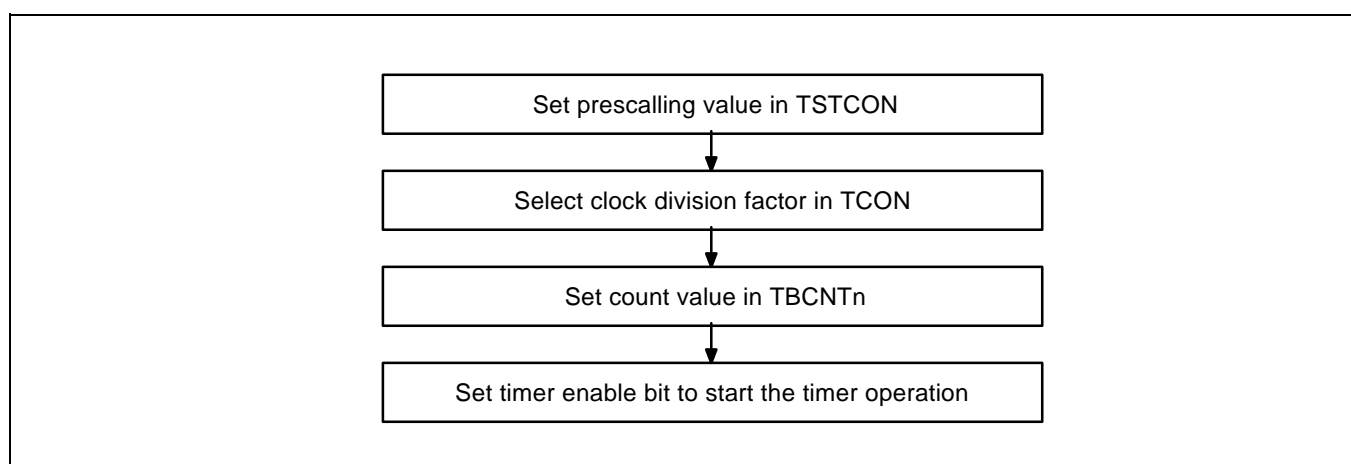


Figure 8-4. Timer Programming Sequence

9 DMA

OVERVIEW

The KS32C6200 has two general direct memory access channels (DMA0, DMA1) that perform the data transfers between the following sources without CPU intervention:

- Memory and memory
- Parallel port and memory
- Serial port and memory

The on-chip DMA controller can be started by software or by an external DMA request. DMA operation can also be stopped and restarted by software. The CPU can recognize when a DMA operation has been completed by software polling or by a DMA interrupt request. The KS32C6200 DMA controller can increase or decrease source or destination addresses and conduct 8-bit (byte), 16-bit (half-word), or 32-bit (word) data transfers.

Detailed information about the DMA block's operation is provided in the descriptions of the DMA registers.

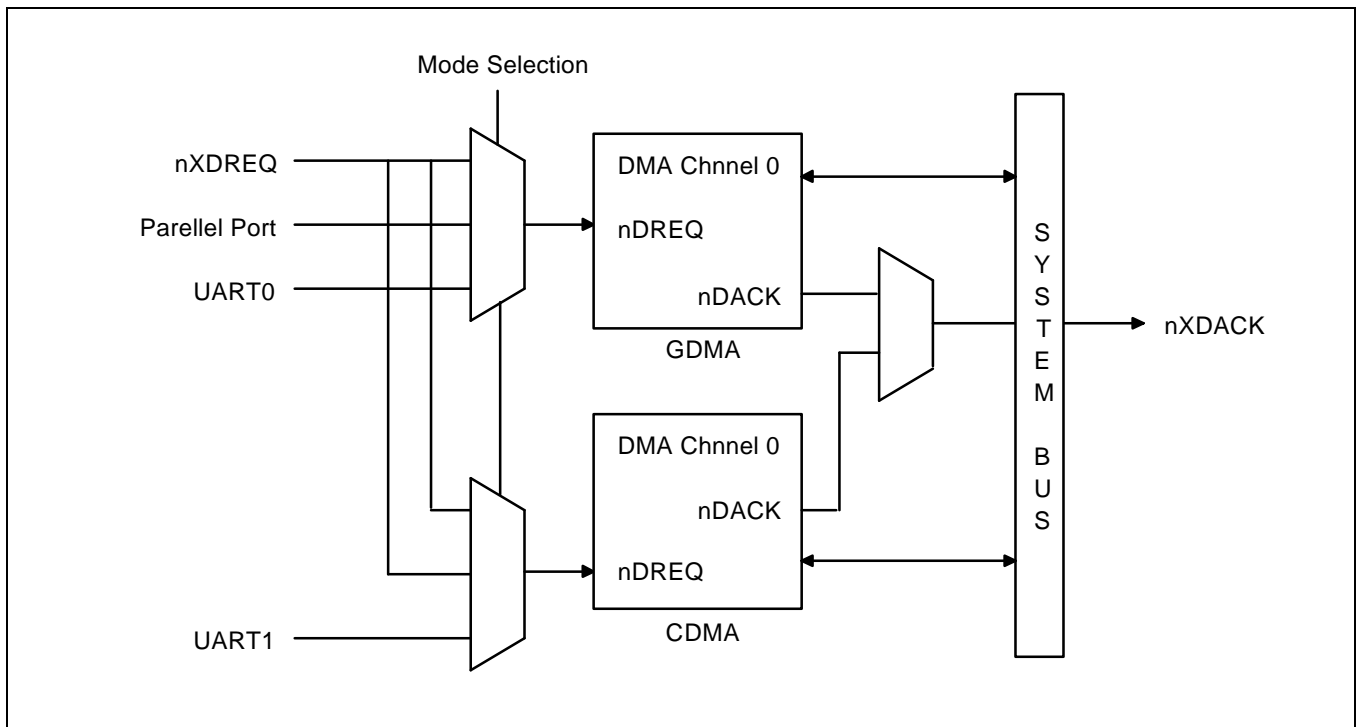


Figure 9-1. DMA0/DMA1 Unit Block Diagram

DMA OPERATION

The DMA operation can be summarized as follows:

- DMA transfers
- Bus control arbitration
- Starting/ending DMA transfers

DMA Transfers

The DMA transfers data directly between a requester and a target. The requester or the target is memory, UART, parallel port, or an external device. An external device requests DMA service by activating the nXDREQ signal.

A channel is programmed by writing the DMA control registers, which contain control information such as requester address, target address, and the amount of data. UART, parallel port, external I/O, and software (memory) can request DMA service. UART and parallel port are internally connected to the DMA. Especially, UART0 requests the DMA service to DMA0 and UART1 to DMA1.

Bus Control Arbitration

Because DMA0, DMA1 and DRAM controller (DRAM refresh) can request bus control, the bus control priority must be arbitrated. The priority of these bus masters is as follows:

Bus Master Type	Priority
DMA0	1
DRAM controller (DRAM refresh)	2
DMA1	3
Write buffer	4
CPU core	5

For fast response to the DMA0 request, the DMA0 has the highest priority. Because the DMA0 has higher priority than the DRAM controller, the DMA0 have to be used very carefully to avoid disturbing the DRAM controller to refresh the DRAM. You may think that the DMA0 can not move the large data of DRAM because of the DRAM refresh. But, the DMA0 can transfer the large DRAM data if you do not use continuous DMA0 mode. The DMA0, which doesn't use the continuous mode, releases the internal bus request in a short time after one unit of data is transferred (one word, one half-word (16-bit) or one byte). Just after the bus is released, the DRAM controller can control the bus and refresh DRAMs.

If the DMA1, which has lower priority than the DRAM controller, holds bus by the continuous mode, the DRAM refresh controller can not control the bus until the DMA1 frees the bus control.

The transfer-rate of DMA is changed by the CPU core status. If the CPU core fetches data or instructions from external memories (cache memory), the DMA transfer-rate became low (high), because the CPU core, which has the lowest priority, may control the bus in order to read/write the data or instructions from external memories.

Starting/Ending DMA Transfers

DMA starts to transfer data after the DMA receives the service request from the nXDREQ signal, UART, parallel port, or software. When the entire data buffer has been transferred, the DMA becomes idle. If you want to preform another buffer transfer, the DMA must be reprogrammed. When the same buffer transfer is preformed again, the DMA must be reprogrammed.

DATA TRANSFER MODE

Single Mode

The DMA request (nXDREQ, UART or PPIC) causes one byte, one half-word, or one word to be transmitted. The single mode requires the DMA request for every data transfer. The nXDREQ signal may be deasserted after checking whether or not the nXDACK is asserted.

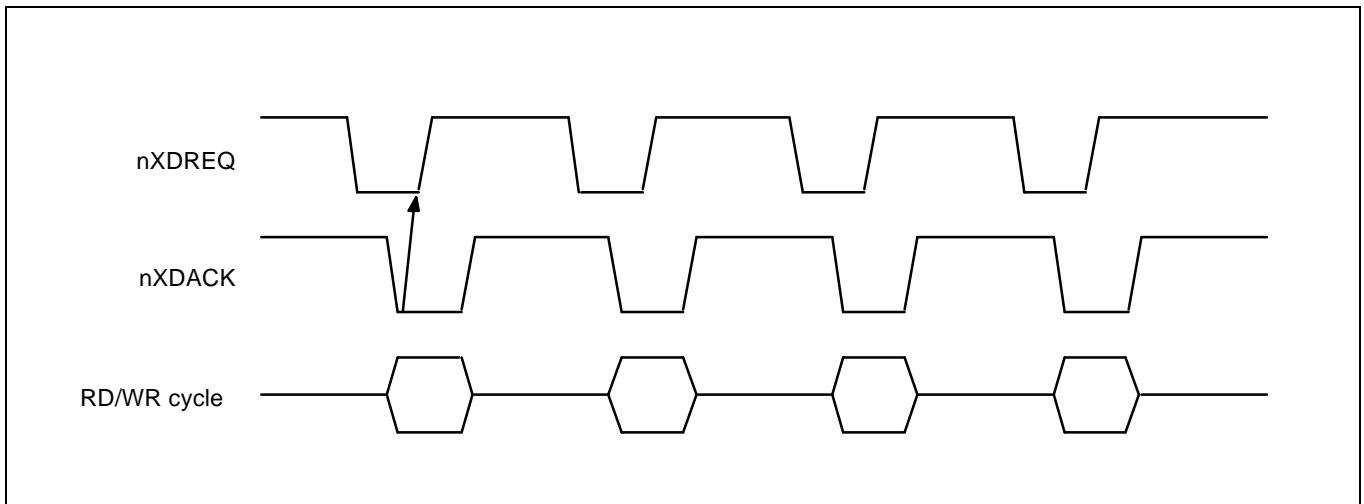


Figure 9-2. External DMA Requests (Single Mode)

Block Mode

The assertion of only one DMA request (nXDREQ, UART, PPIC or S/W) causes the entire data, which is set in control registers, to be transmitted. The DMA transfer will be completed when the counter reaches zero. The nXDREQ signal may be deasserted after checking whether or not nXDACK is transmitted.

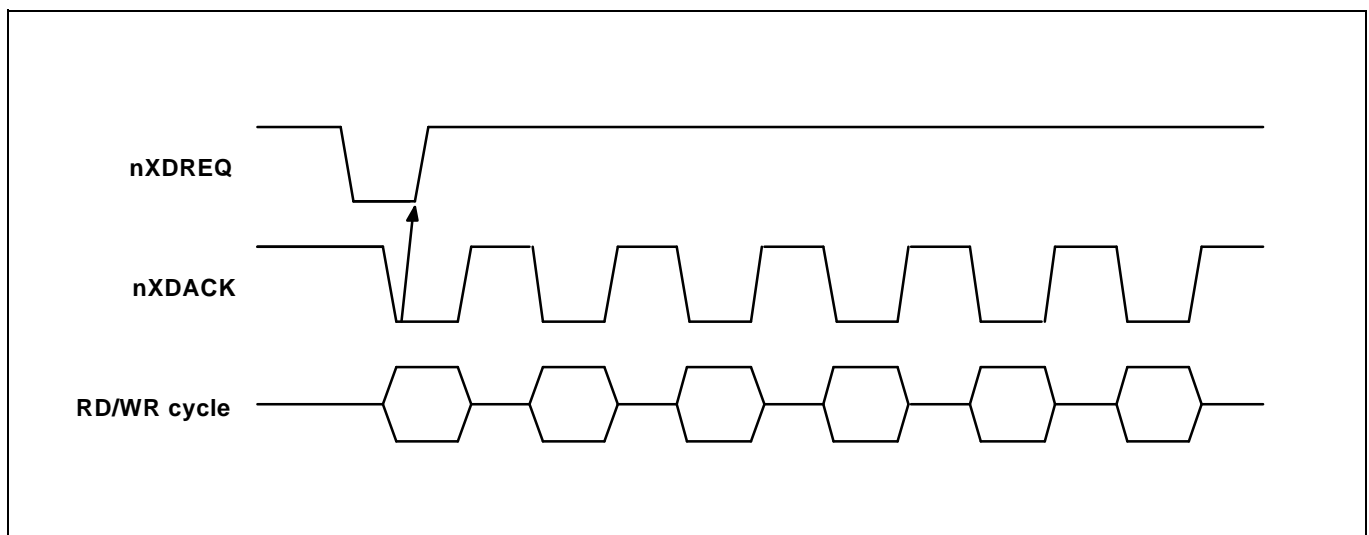


Figure 9-3. External DMA Requests (Block Mode)

Demand Mode

The amount of data that DMA transfers depends on how long the DMA request input (nXDREQ) is held active. In demand mode, the DMA (DMA1 only) continues to transfer data while the DMA request input (nXDREQ) is held active.

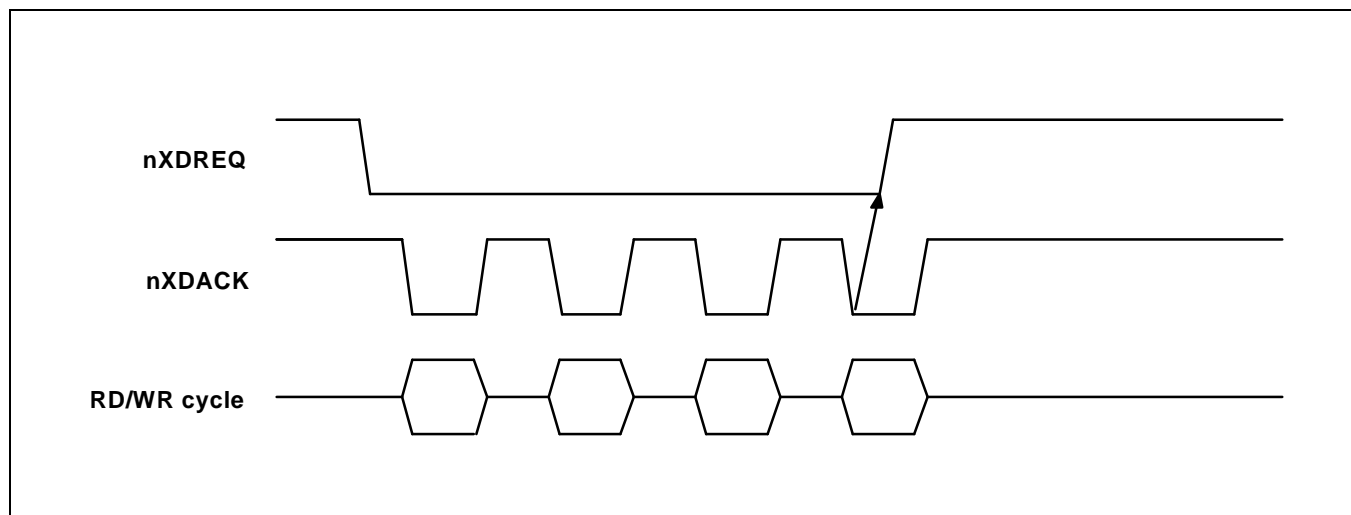


Figure 9-4. External DMA Requests (Demand Mode)

DMA0 CONTROL REGISTER

Register	Offset Address	R/W	Description	Reset Value
DMACON0	0xc000	R/W	DMA0 control register	0x0000

[0] Run enable/disable	When you set this bit to '1', the DMA operation starts. To stop the DMA operation, you must clear this bit to '0'. To control only this bit, use the address 0xc020. By using the 0xc020 address, you can avoid affecting the other values of the control register.			
[1] BUSY status ⁽²⁾	When the DMA starts, this read-only status bit is automatically set to '1'. When the DMA is in an idle state, this bit is set to '0'.			
[3:2] DMA0 mode selection	Four sources can initiate DMA operation: software (memory to memory), an external DMA request (nXDREQ), the parallel port, and the UART block. The mode selection bits determine which source can initiate a DMA operation at any given time (see Figure 9-5).			
[4] Destination address direction	This bit determines whether the destination address will be decreased or increased during a DMA operation.			
[5] Source address direction	This bit determines whether the source address will be decreased or increased during a DMA operation.			
[6] Destination address fix	This bit determines whether the destination address will be changed during a DMA operation. You can use this bit to transfer data from multiple sources to a single destination.			
[7] Source address fix	This bit determines whether the source address will be changed during a DMA operation. You can use this bit to transfer data from a single source to multiple destinations.			
[8] Stop interrupt enable	A DMA operation is started by setting the run enable bit to one and is stopped by clearing the run disable bit to zero. When this bit is set to '1' and the DMA is forced to stop, the 'stop interrupt' is generated. If this bit is '0', the 'stop interrupt' is not generated. The interrupt, which is generated when the DMA counter is expired, can not be masked by this bit.			
[9] Reset	If this bit is set to '1', the value of DMA0 control registers will be initialized. When this bit is cleared to '0', you can specify other control values.			
[10] Peripheral direction	When the mode bit is set to '10' (parallel port from / to memory) or '11' (UART from / to memory), this direction bit specifies the direction of the DMA operation. If this bit is set to '1', the DMA operates from memory to peripheral (parallel port/UART). If this bit is cleared to '0', the DMA operates from peripheral to memory.			

[11] Single/block mode	This bit determines the number of external DMA requests (nXDREQ) that are required for the DMA operation. At single mode (this bit is set to '0'), the KS32C6200 requires an external DMA request for every DMA operation. At block mode (this bit is set to '1'), the KS32C6200 requires only one DMA request during the entire DMA operation. An entire DMA operation is the DMA operation before the value of counter becomes zero.
[13:12] Transfer width	This determines the transfer data width: byte (8-bit), half-word (16-bit), or word (32-bit). If the transfer data width is a byte, source/destination address will be increased/decreased by one. If it is a half-word, the address will be increased/decreased by two. If it is a word, the address will be increased/decreased by four. Note that the "transfer width" is not the physical size of data bus. The physical size of data bus is determined by SMR (System Manager Registers) configurations.
[14] Continuous mode	This bit specifies whether or not the DMA operation holds the system bus until the count value is changed to zero. Therefore, this bit must be carefully used so that the whole operation time does not exceed appropriate intervals such as DRAM Refresh.
[15] Demand mode	To speed up the external DMA operation, set this bit. If this bit has been set during the DMA operation, the DMA never goes to the idle state. The external device can control the amount of data transferred/received by hardware. The amount of data is depends on how long nXDREQ signal is active.

NOTES

1. All control bits have to be configured independently and carefully.
2. The BUSY status is "read-only" bit.

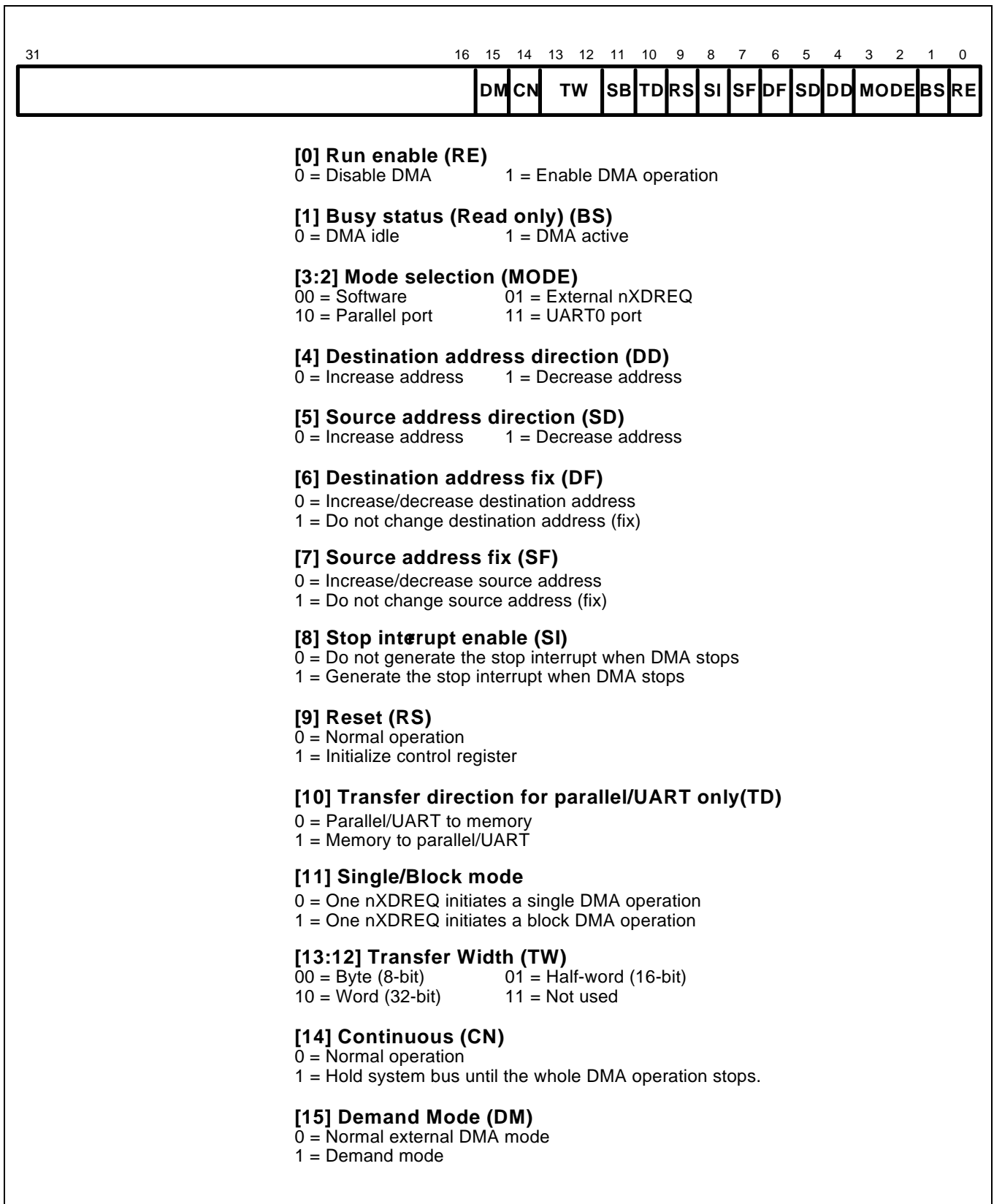


Figure 9-5. DMA0 Control Register

DMA0 SOURCE/DESTINATION ADDRESS REGISTER

These registers contain the 25-bit source/destination address of a DMA channel. Depending on the setting of the DMA control register (DMACON0), these addresses will be increased/decreased or will remain the same.

Register	Offset Address	R/W	Description	Reset Value
DMA_SRC0	0xc004	R/W	DMA source address register	Undefined
DMA_DST0	0xc008	R/W	DMA destination address register	Undefined

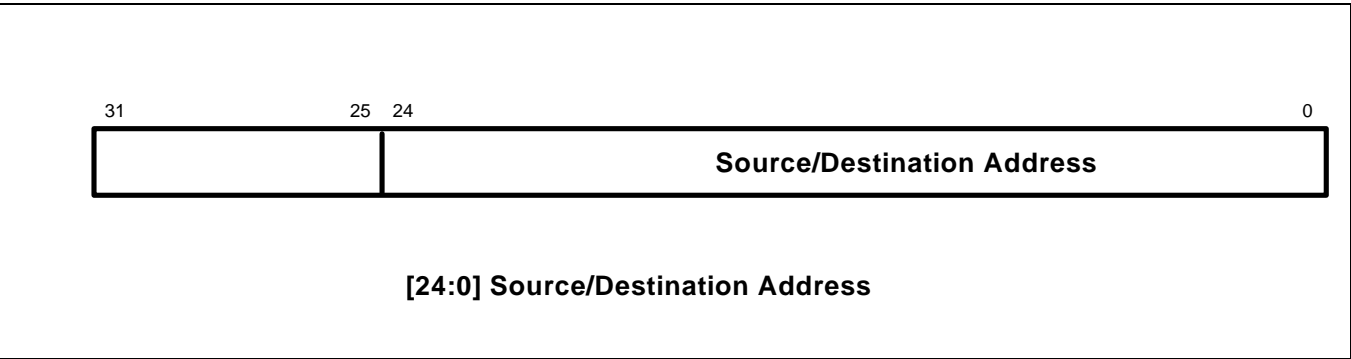


Figure 9-6. DMA0 Source/Destination Address Register

DMA0 TRANSFER COUNT REGISTER

This register contains the 24-bit value which is the number of DMA transfers. This value is decreased by 1 when one DMA operation is completed regardless of the width of the data which was transferred.

Register	Offset Address	R/W	Description	Reset Value
DMA_CNT0	0xc00c	R/W	DMA transfer count register	Undefined

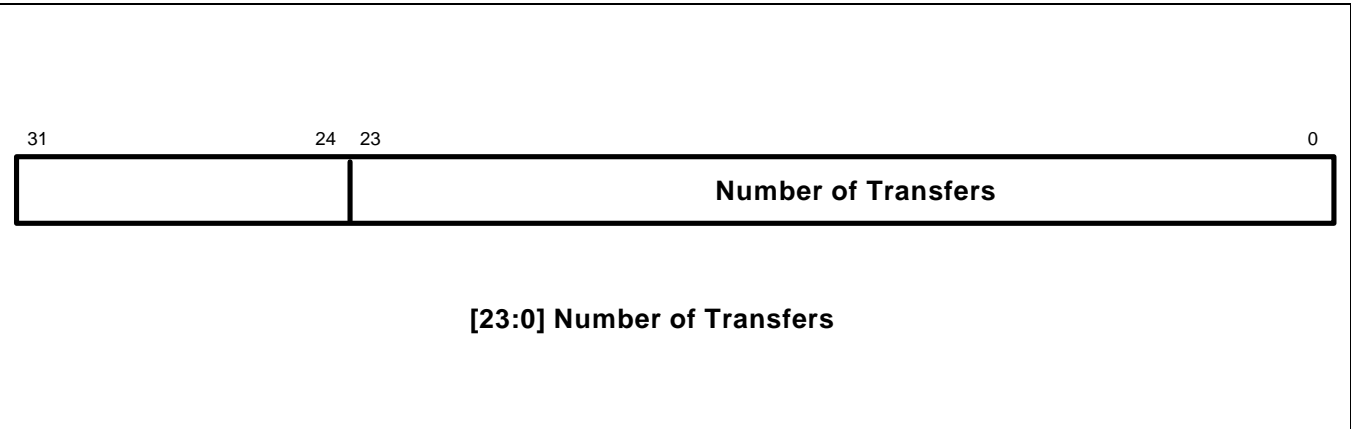


Figure 9-7. DMA0 Transfer Count Register

DMA1 CONTROL REGISTER

DMA1 is the second DMA. The UART1 can transfer data through only DMA1.

Register	Offset Address	R/W	Description	Reset Value
DMACON1	0xc800	R/W	DMA1 control register	0x0000000

[0] Run enable/disable	When you set this bit to '1', the DMA1 operation starts. To stop the DMA1 operation, you must clear this bit to '0'. To control this bit only, use the address 0xc810. By using the 0xc810 address, the other values in the control register will not be affected.			
[1] BUSY status ⁽²⁾	When DMA1 starts, this read-only status bit is automatically set to '1'. When it is cleared to '0', the DMA1 goes into an idle status.			
[3:2] DMA1 mode selection	Four sources can initiate a DMA1 operation: software, an external DMA request (nXDREQ), the parallel port, and the UART block. The DMA1 mode selection bits determine which source can initiate the DMA1 operation at any given time (see Figure 9-8).			
[4] Destination address direction	This bit determines whether the destination address will be decreased or increased during a DMA1 operation.			
[5] Source address direction	This bit determines whether the source address will be decreased or increased during a DMA1 operation.			
[6] Destination address fix	This bit determines whether the destination address will or will not change during a DMA1 operation. This feature is used when transferring data from multiple sources to a single destination.			
[7] Source address fix	This bit determines whether or not the source address will change during a DMA1 operation. This feature is used when transferring data from a single source to multiple destinations.			
[8] Stop interrupt enable	A DMA1 operation is started by setting the run enable bit to one, and is stopped by clearing the disable bit to zero. When this bit is set to one and the DMA operation is forced to stop, the "stop interrupt" is generated. If this bit is set to zero, the "stop interrupt" is not generated. The interrupt, which is generated when the DMA counter is expired, can not be masked by this bit.			
[9] Reset	If this bit is set to one, the DMA1 control register value will be initialized. When this bit is cleared to zero, you can specify other control values.			
[10] Peripheral direction	When the mode bit is set to '10' (parallel port from/to memory) or '11' (UART from/to memory), this direction bit specifies the direction of the DMA1 operation. If this bit is set to '1', the DMA1 operates from memory to peripheral (parallel port/UART). If this bit is cleared to '0', the DMA1 operates from peripheral to memory.			

[11] Single / Block mode	This bit determines the number of external DMA requests (nXDREQ) that are required for the DMA operation. At single mode (this bit is set to zero), the KS32C6200 requires an external DMA request for every DMA operation. At block mode (this bit is set to one), the KS32C6200 requires only one DMA request during the entire DMA operation. An entire DMA operation is the DMA operation before the value of counter becomes zero.
[13:12] Transfer width	This determines the width of the data being transferred: byte, half-word, or word. If the data transfer width is a byte, the source/destination address will be increased/decreased by one. If it is a half-word, the address changes by 2. If a word, the address changes by 4. Note that the "transfer width" is not the size of the physical data bus. The size of physical data bus is determined by SMR (System Manager Registers) configurations.
[14] Continuous mode	This bit specifies that the DMA1 operation hold the system bus until the count value to set to "0". Therefore, this bit must be carefully used unless the whole operation time can not exceed the appropriate interval such as DRAM refresh.

NOTES

1. All control bits have to be configured independently and carefully.
2. The BUSY status is "read-only" bit.

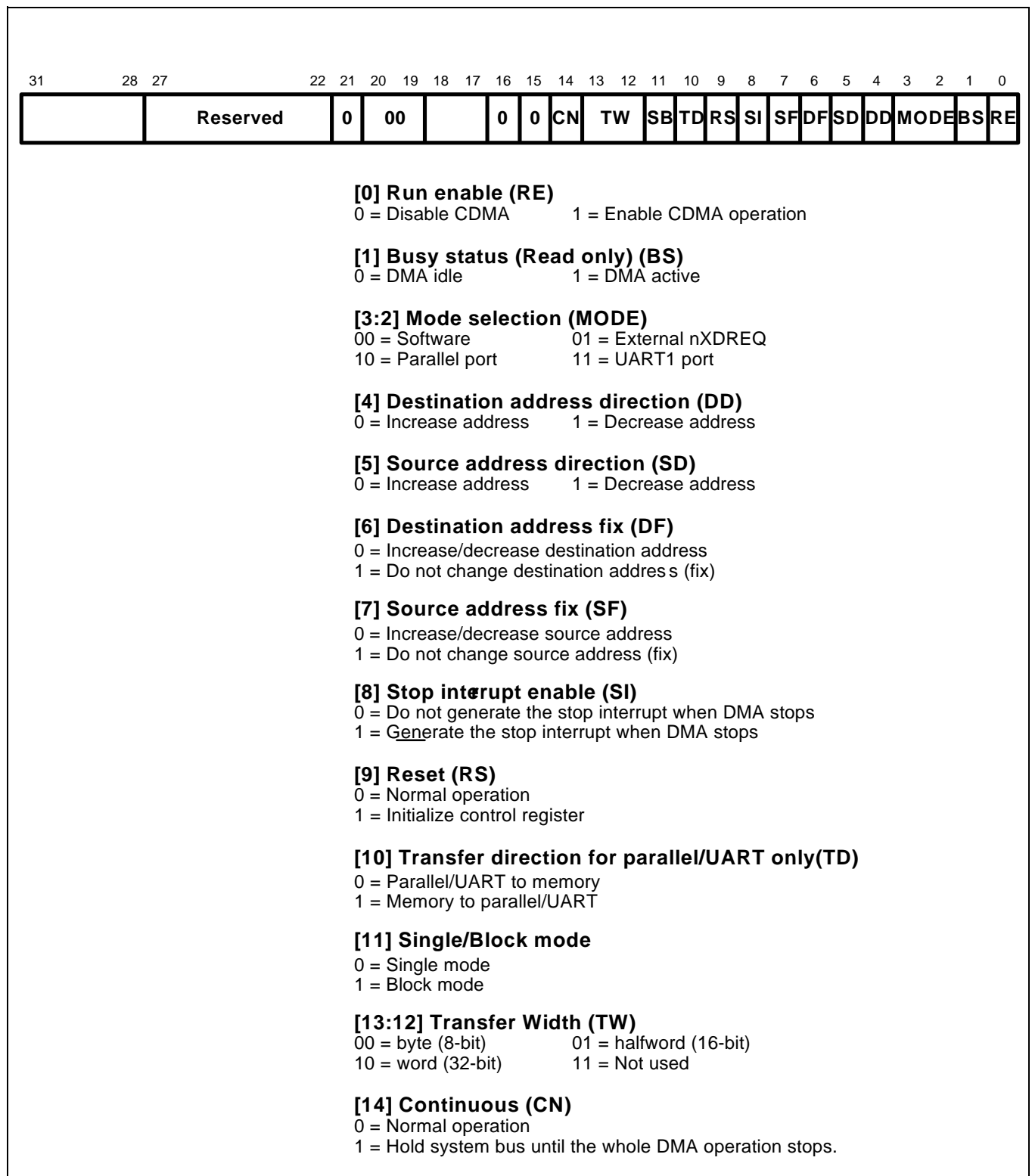


Figure 9-8. DMA1 Control Register

DMA1 SOURCE/DESTINATION ADDRESS REGISTER

These registers contain the 25-bit source/destination address of a DMA1 channel. Depending on the setting of the DMA1 control register(DMACON1), these address will be increased, decreased, or will remain the same.

Register	Offset Address	R/W	Description	Reset Value
DMA1SRC1	0xc804	R/W	DMA1 source address register	Undefined
DMA1DST1	0xc808	R/W	DMA1 destination address register	Undefined

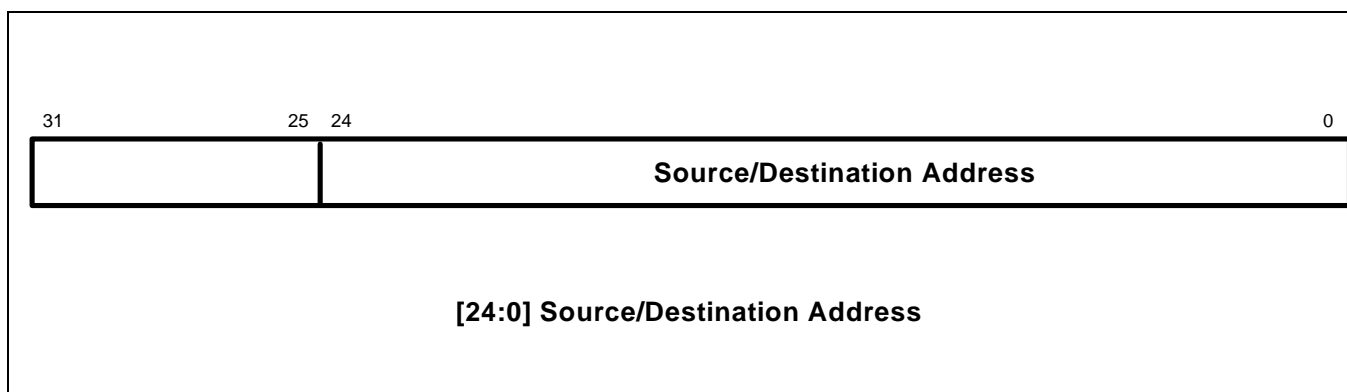


Figure 9-9. DMA1 Source/Destination Address Register

DMA1 TRANSFER COUNT REGISTER

This register contains the 24-bit value which is the number of DMA1 transfers. This value is decreased by 1 when one DMA operation is completed regardless of the width of the data transferred.

Register	Offset Address	R/W	Description	Reset Value
DMA1CNT1	0xc80c	R/W	DMA1 transfer count register	Undefined

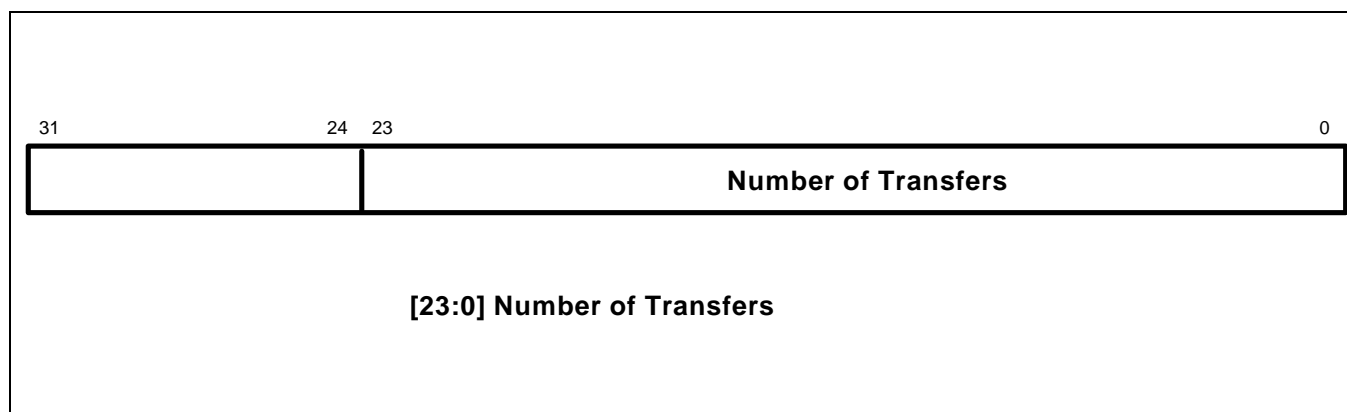


Figure 9-10. DMA1 Transfer Count Register

10

Parallel Port Interface

OVERVIEW

The KS32C6200's parallel port interface controller (PPIC) supports four IEEE Standard 1284 communication modes:

- Compatibility mode (Centronics™)
- Nibble mode
- Byte mode
- Enhanced Capabilities Port (ECP) mode

The PPIC also supports all variants of these communication modes, including device ID requests and run-length encoded (RLE) data compression. The PPIC contains specific hardware to support the following operations:

- Automatic hardware handshaking between host and peripheral compatible with ECP modes
- Run-length detection and compression/decompression data between host and peripheral during ECP mode transfers

These features can substantially improve data transfer rates when KS32C6200 operates the parallel port in the Compatibility or ECP mode.

In addition, hardware handshaking over the parallel port can be enabled or disabled by software. This gives you the direct control of PPIC signals as well as the eventual use of future protocols. Other operations defined in the IEEE Standard 1284, such as negotiation, Nibble mode and Byte mode data transfers, and termination cycles, must be carried out by software. The IEEE 1284 EPP communications mode is not supported.

NOTE

Here we assume that you are familiar with the parallel port communication protocols specified in the IEEE 1284 Parallel Port Standard. If you are not, we strongly recommend for you to read this standard beforehand. It would be helpful for you in understanding the contents described in this section.

A detailed technical introduction to the IEEE 1284 Parallel Port Standard can be found in the Web site:
<http://www.fapo.com/ieee1284.htm>

PPIC OPERATING MODES

The KS32C6200 PPIC supports four kinds of handshaking modes for data transfers:

- Software handshaking mode to forward and reverse data transfers
- Compatibility hardware handshaking mode to forward data transfers
- ECP hardware handshaking without RLE support (ECP-without-RLE) mode to forward and reverse data transfers
- ECP hardware handshaking with RLE support (ECP-with-RLE) mode to forward and reverse data transfers

Mode selection is specified in the PPIC control register (PPCON). By setting the PPCON[3:2], one of these four modes is enabled.

Software Handshaking Mode

This mode is enabled by setting the PPCON's mode-selection bits, PPCON[3:2], to "00."

In this mode, you can use PPIC interrupt event registers (PPINTEN and PPINTPND) and the read/write PPIC status register (PPSTAT) to detect and control the logic levels on all parallel port signal pins. Software can control all parallel port operations, including all four kinds of parallel port communications protocols supported by the KS32C6200 (refer to IEEE 1284 standard for operation control). In addition, it also gives software the flexibility of adopting new and revised protocols.

Compatibility Hardware Handshaking Mode

Compatibility hardware handshaking mode is enabled by setting the PPCON's mode-selection bits as "01", i.e. PPCON[3:2] = 01. In this mode, hardware generates all handshaking signals needed to implement compatibility mode of the parallel port communication protocol.

When this mode is enabled, the PPIC automatically generates a BUSY signal to receive the leading edge of nSTROBE from the host, and latches the logic levels on PPD7-PPD0 pins into the PPDATA register. The PPIC then waits for nSTROBE to negate it and for the PPDATA's data field to be read. After the PPDATA is read, the PPIC asserts nACK for the duration specified in the ACK Width Register (PPACKWTH), and then negates the nACK and BUSY signal to conclude the data transfer, as shown in Figure 10-1.

NOTE

The BUSY-control bit's initial value in the PPSTAT register, PPSTAT[3], which is "1" after a system reset, results in the high logic level on BUSY output and handshaking disable. To enable hardware handshaking in this mode, the BUSY-control bit PPSTAT[3] must be cleared to "0" by software beforehand.

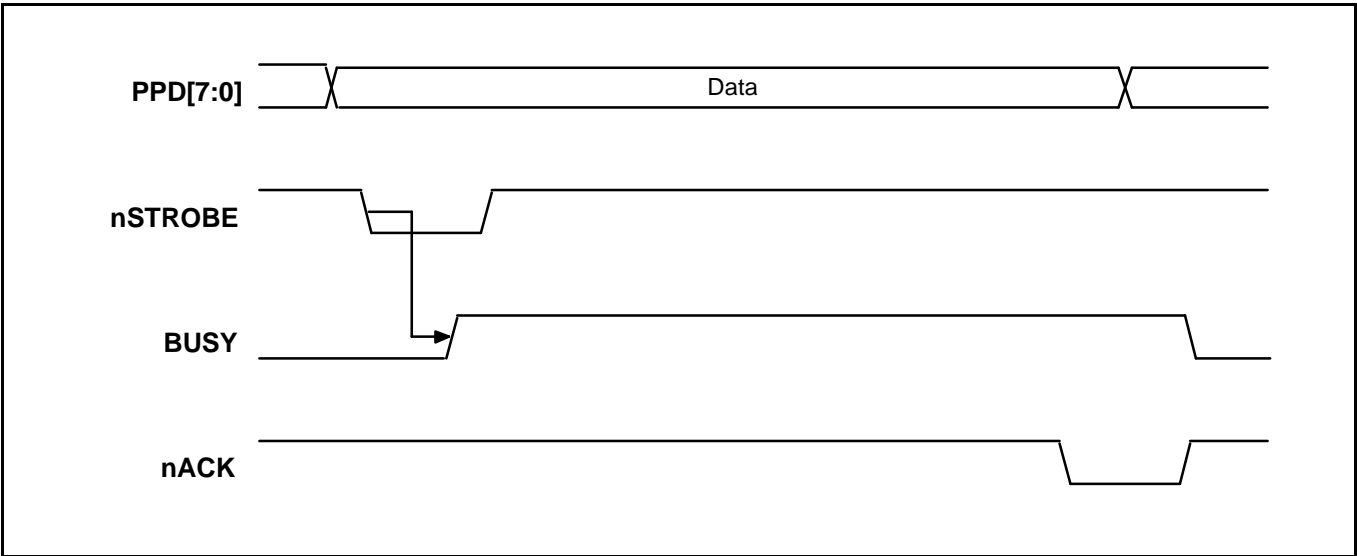


Figure 10-1. Compatibility Hardware Handshaking Timing

ECP-without-RLE Mode

ECP-without-RLE hardware handshaking mode is enabled by setting the PPCON's mode-selection bits to "10", i.e. $PPCON[3:2] = 10$. In this mode, hardware generates handshaking signals needed to implement ECP mode of the parallel port communication protocol.

When receiving data from the host, the PPIC automatically responds to the high-to-low transition on the $nSTROBE$ by latching the logic levels on the PPD7-PPD0 and $nAUTOFD$ in the PPDATA register. The $nAUTOFD$ logic level, which is latched to the PPDATA[8], indicates whether the current data on the PPD[7:0] is a data-byte or a command-byte. When the PPDATA is read, the PPIC drives $BUSY$ to high level and waits for $nSTROBE$ to go high level. It then drives $BUSY$ to low level to conclude one forward data transfer operation, as shown in Figure 10-2.

The reception of a command byte, indicated by $PPDATA[8]=0$, causes the command received-bit in the PPIC interrupt pending register, $PPINTPND[9]$, to be set to "1". By examining the $PPDATA[7]$, software will interpret the command byte as a channel address if it is "1" and carry out the corresponding operation, or interpret the command byte as a run-length count if it is "0" and then perform data decompression.

During reverse data transfers, software is responsible for data compression, and writing data or command byte in PPDATA to define the logic levels on PPD7-PPD0 and $BUSY$ pins. The $PPDATA[8]$ indicates whether the current data on the $PPDATA[7:0]$ is a data-byte or a command-byte. The state of $PPDATA[8]$ is output through the $BUSY$ pin. In response to writing the PPDATA, the PPIC automatically drives the $nACK$ to low level and waits for the $nAUTOFD$ to go to high level. It then drives $nACK$ to high level to conclude one reverse data transfer operation, as shown in Figure 10-3.

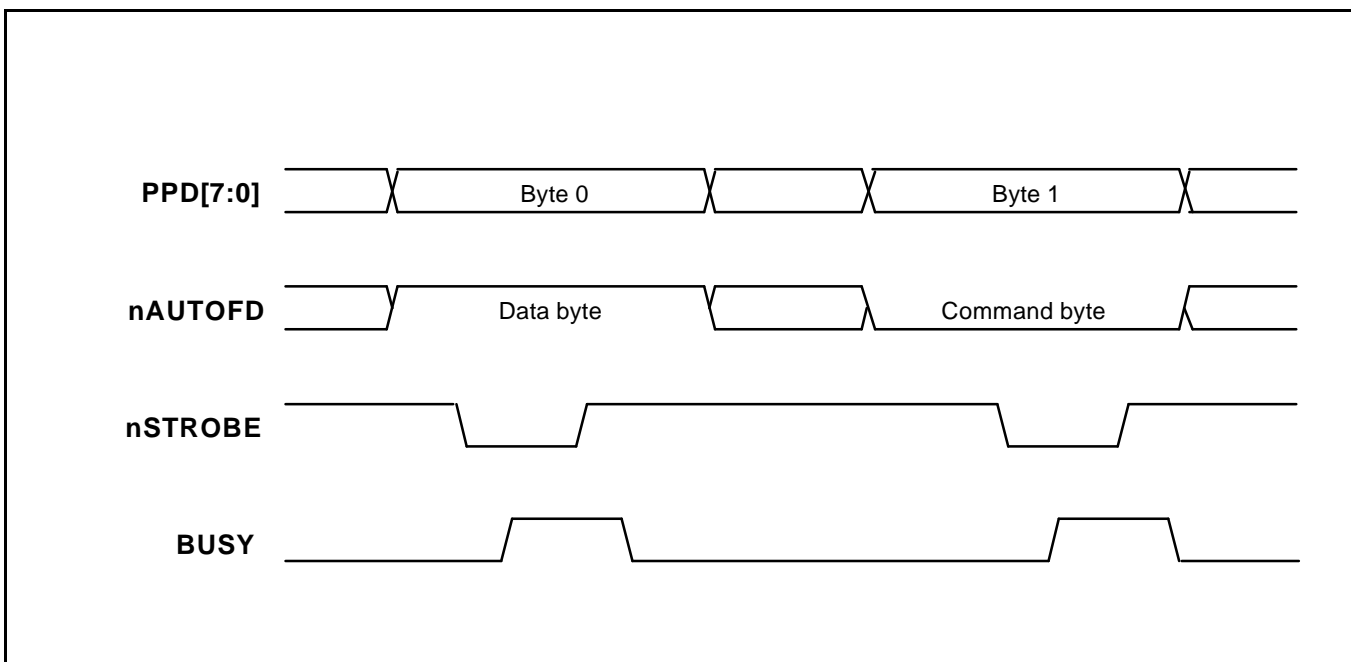


Figure 10-2. ECP Hardware Handshaking Timing (Forward)

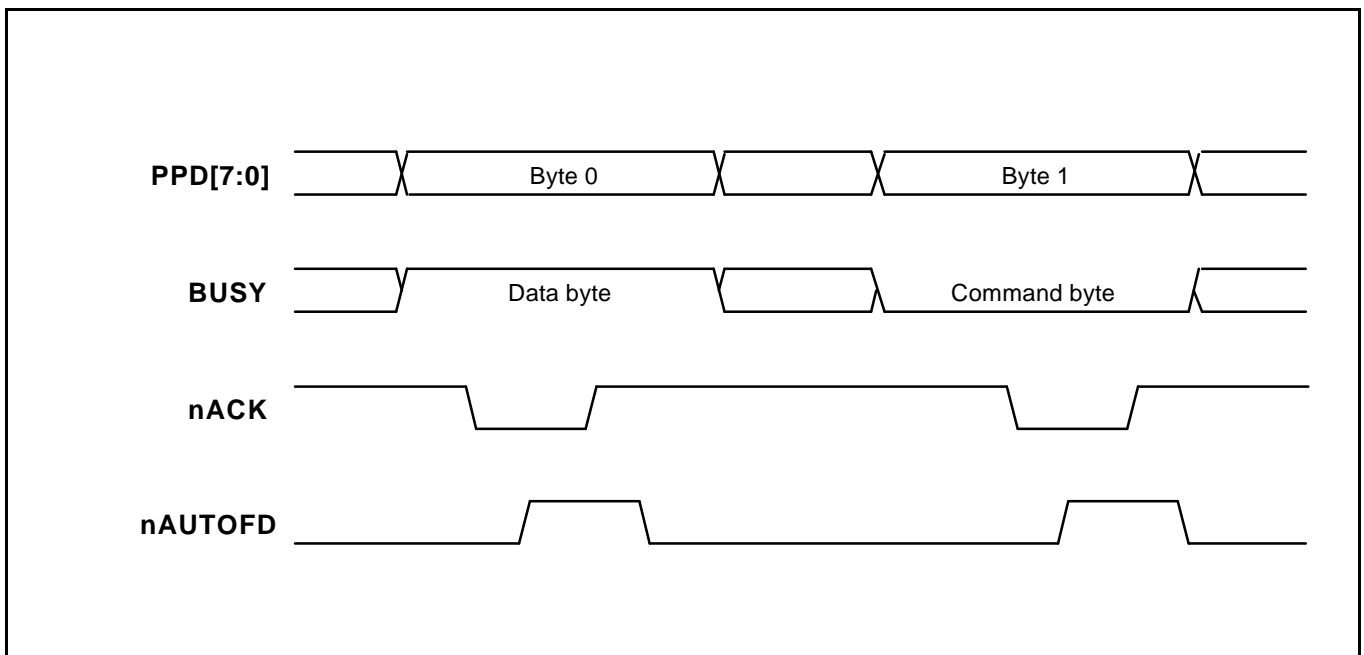


Figure 10-3. ECP hardware Handshaking Timing (Reverse)

ECP-with-RLE Mode

ECP-with-RLE hardware handshaking mode is enabled by setting the PPCON's mode-selection bits, PPCON[3:2], to "11." In this mode, the PPIC performs the same ECP mode handshaking as in ECP-without-RLE mode, except for the fact that run-length compression/decompression is also carried out by hardware.

During forward data transfers, the PPIC automatically detects and intercepts run-length counts, and carries out data decompression. Only the channel addresses will cause the command-received-bit in the PPINTPND register, PPINTPND[9], to be set to one. If the command-receive interrupt occurs in ECP-with-RLE mode, the software performs the operations associated with the channel address.

Similarly, the PPIC automatically carries out the data compression in PPDATA during the reverse data transfers.

Digital Filtering

The KS32C6200 provides digital filtering function on host control signal inputs, nSELECTIN, nSTROBE, nAUTOFD and nINIT, to improve noise immunity and make the PPIC more impervious to the inductive switching noise. The digital filtering function can be enabled regardless of hardware handshaking or software handshaking.

If this function is enabled, the host control signal can be detected only when its input level keeps stable during two sampling periods.

Digital filtering can be disabled to avoid signal missing in some specialized applications with high bandwidth requirement. Otherwise, it is recommended that digital filtering be enabled.

PPIC SPECIAL REGISTERS

PARALLEL PORT DATA REGISTER

The parallel port data register, PPDATA, contains an 8-bit data field, PPDATA[7:0], that defines the logic level on the parallel port data pins, PPD[7:0]. It also contains a status bit, PPDATA[8], which is used to indicate when a command byte (RLE count or channel address) is received during forward data transfers in ECP mode.

Register	Offset Address	R/W	Description	Reset Value
PPDATA	0xb000	R/W	Parallel port data register	0x000

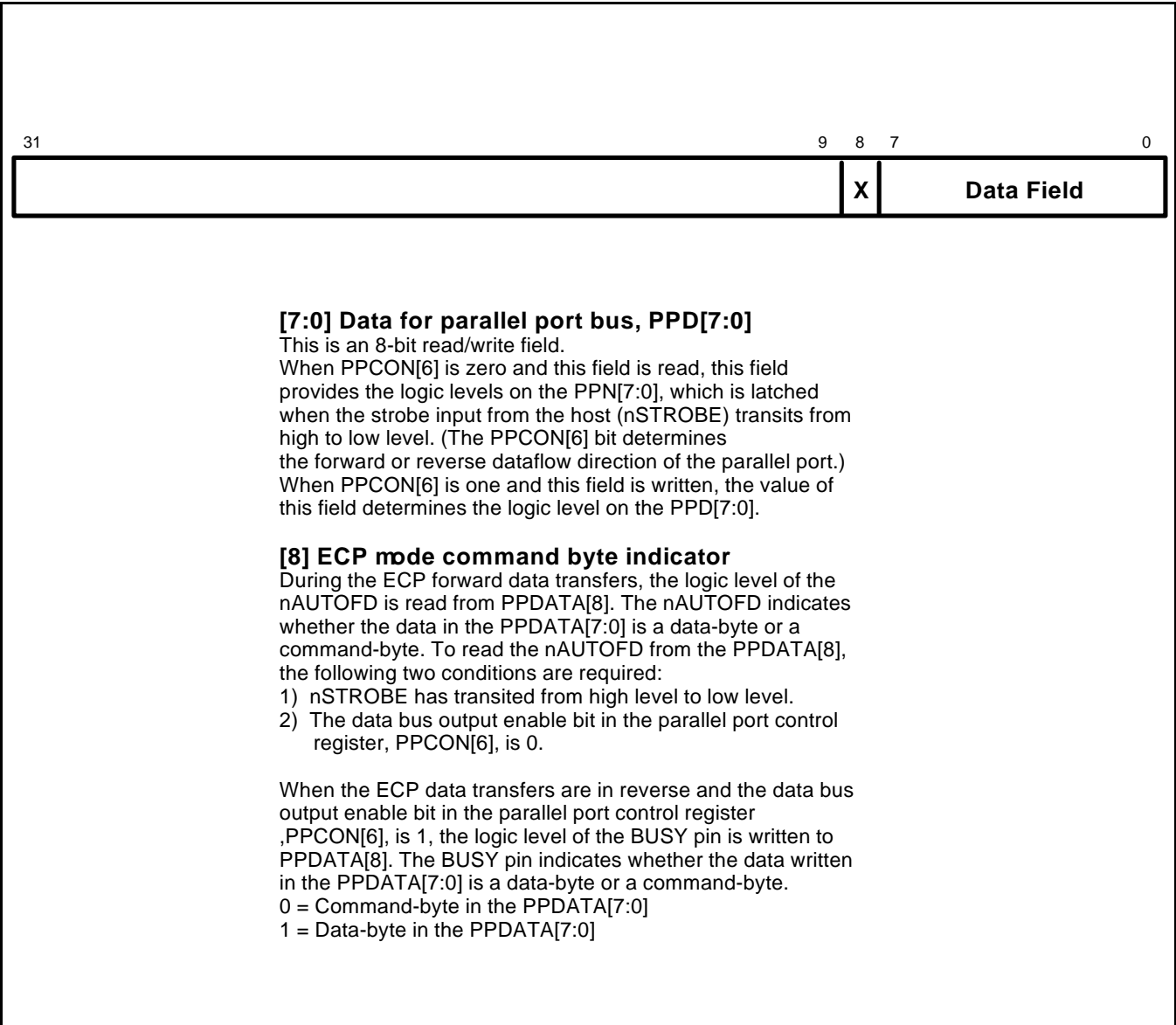


Figure 10-4. Parallel Port Data Register

PARALLEL PORT STATUS REGISTER

The parallel port status register, PPSTAT, contains eleven bits to control the parallel port interface signals. These eleven bits consist of four read-only bits to read the logic level of the host input pins, two read-only bits to read the logic level on the BUSY and nACK output pins, and five read/write bits to control the logic levels on the printer output pins by software for handshaking control.

Register	Offset Address	R/W	Description	Reset Value
PPSTAT	0xb004	R/W	Parallel port status register	0x000007e8

[0] nFAULT control	Setting this bit drives the nFAULT output to low level; clearing it drives the signal high level on the external nFAULT pin. The nFAULT informs the host of a fault condition in the printer engine.
[1] SELECT control	Setting this bit to one drives the SELECT output to High level; clearing it to zero drives the signal low on the external SELECT pin. The SELECT informs the host of a response from the printer engine.
[2] PERROR control	Setting this bit drives PERROR output to high level; clearing it drives the signal low level on the external PERROR pin. The PERROR informs the host that a paper error has occurred in the engine.
[3] BUSY control	Setting this bit drives the external BUSY output to high level by force. This disables hardware handshaking. When this bit is zero, the external BUSY output is the internal BUSY signal.
[4] nACK control	Setting this bit drives the external nACK output to high level by force. This is generally done when hardware handshaking is disabled. When this bit is one, the external nACK is the internal nACK signal.
[5] BUSY status	This read-only bit reflects the logic level on the external BUSY output pin. After a system reset, the PPSTAT[3] is "1", which results in one, the value of PPSTAT[5] being "1". So, for compatibility mode operation, you must clear the PPSTAT[3] by software beforehand so as to enable the hardware handshaking.
[6] nACK status	This read-only bit reflects the inverted logic level on the external nACK output pin. After a system reset, PPSTAT[6] is "1".
[7] nSLCTIN status	This read-only bit reflects the level read on the nSLCTIN input pin after synchronization and optional digital filtering when the digital filtering enable bit, PPCON[1], is set to one.
[8] nSTROBE status	This read-only bit reflects the level read on the nSTROBE input pin after synchronization and optional digital filtering when the digital filtering enable bit, PPCON[1], is set to one.
[9] nAUTOFD status	This read-only bit reflects the level read on the nAUTOFD input pin after synchronization and optional digital filtering when the digital filtering enable bit, PPCON[1], is set to one.

[10] nINIT status

This read-only bit reflects the level read on the nINIT input pin after synchronization and optional digital filtering when the digital filtering enable bit, PPCON[1], is set to one.

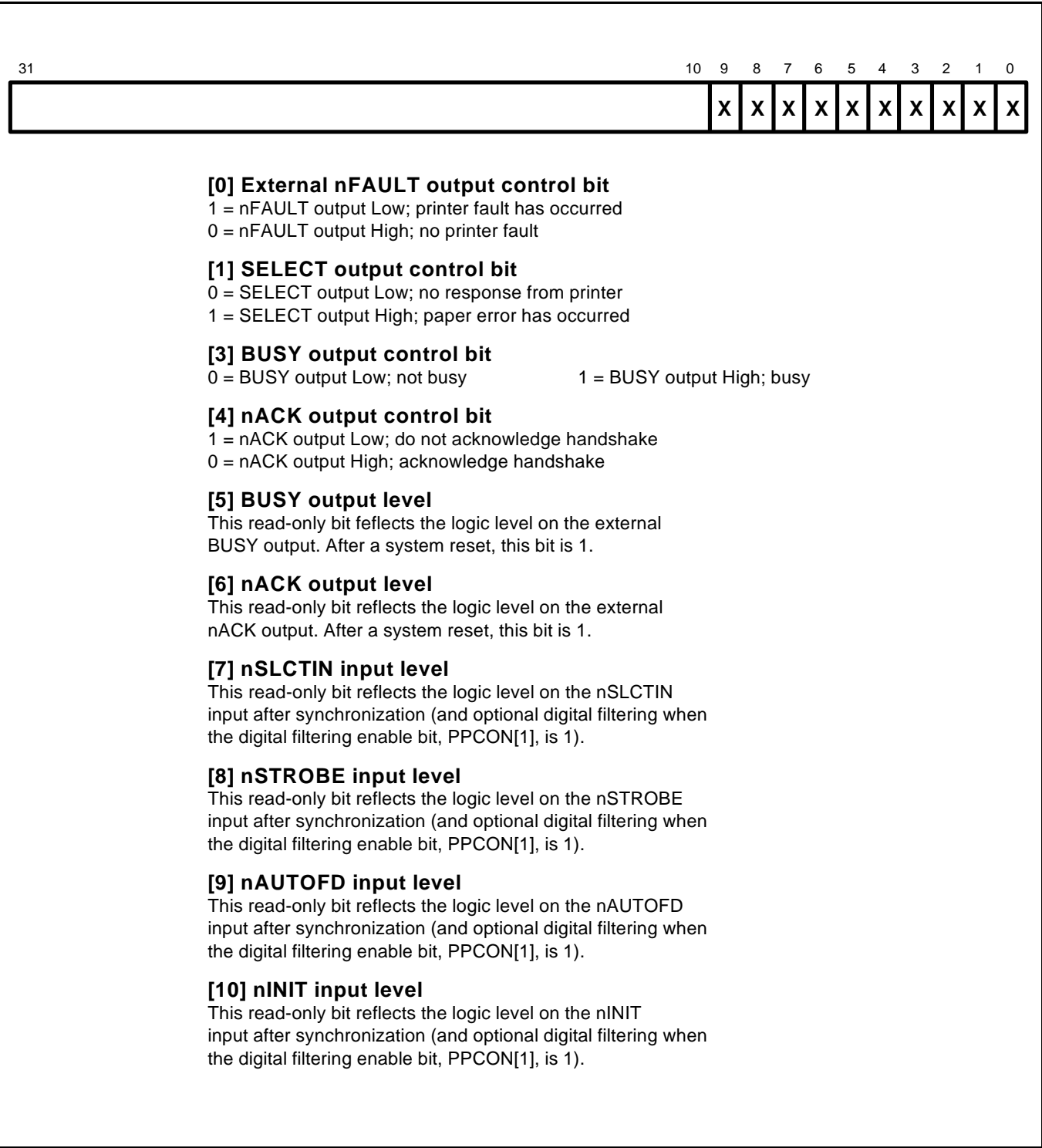


Figure 10-5. Parallel Port Status Register

PARALLEL PORT ACK WIDTH REGISTER

This register contains the 9-bit nACK pulse width field. This value defines the nACK pulse width whenever the parallel port interface controller enters Compatibility mode, that is, when the parallel port control register mode bits, PPCON[3:2], are set to “01”. The nACK pulse width is selectable from 0 to 511 MCLK periods.

The nACK pulse width can be modified at any time and with any PPIC operation mode selection, but it can only be used during a compatibility handshaking cycle. If you change the nACK width near the end of a data transfer (when nACK is already low), the new pulse width value does not affect the current cycle. The new pulse width value would be used at the start of the next cycle.

Register	Offset Address	R/W	Description	Reset Value
PPACKWTH	0xb008	R/W	Parallel port acknowledge width register	Undefined

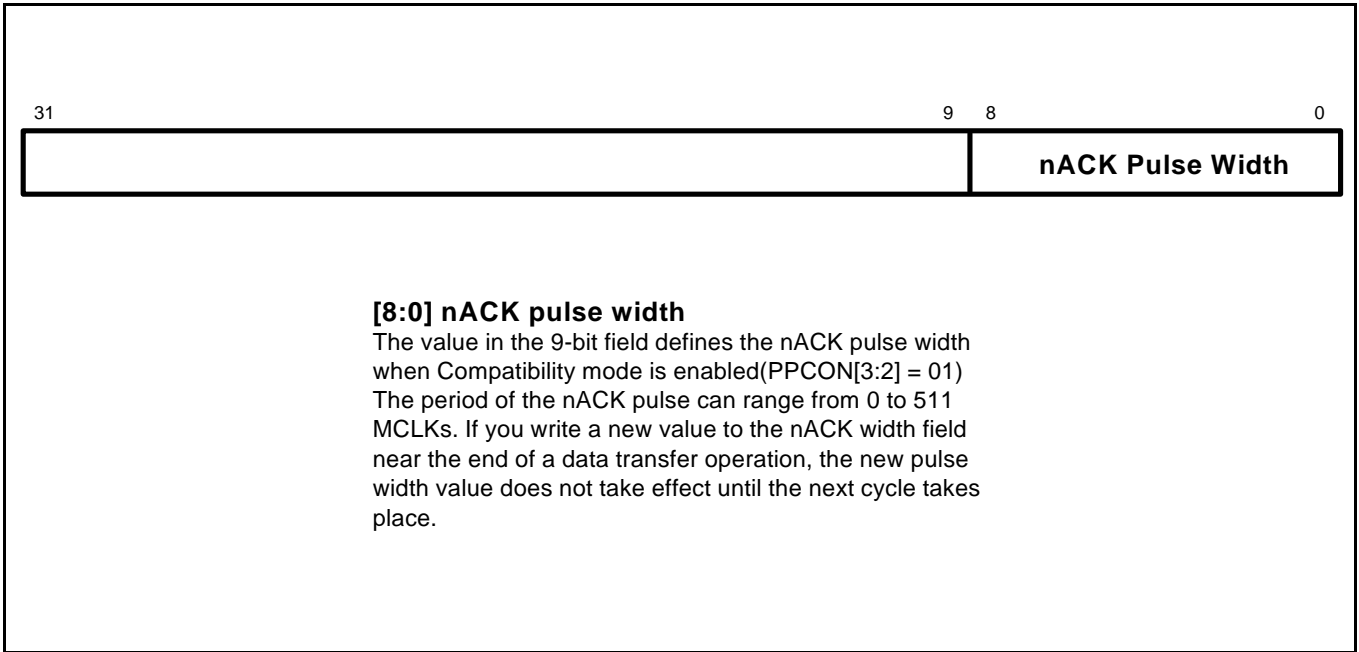


Figure 10-6. Parallel Port ACK Width Register

PARALLEL PORT CONTROL REGISTER

The parallel port control register, PPCON, is used to configure the PPI operations, such as handshaking, digital filtering, operating mode, data bus output, abort operations, and DMA. The PPCON[15:13] bits are read-only.

Register	Offset Address	R/W	Description	Reset Value
PPCON	0xb00c	R/W	Parallel port control register	0x00000000

- [0] Software reset
- Setting the software reset bit causes the PPIC's handshaking control and compression/decompression logic to immediately terminate the current operation and return to software Idle state. When PPCON[0] is set to "1", the run-length decompression status bit, PPCON[13], and the full status bit, PPCON[14], are automatically cleared to "0".
- [1] Digital filter enable
- Setting this bit enables digital filtering on all four host control signal inputs: nSELECTIN, nSTROBE, nAUTOFD, and nINIT.
- [3:2] Mode selection
- This two-bit value selects the current operating mode of the parallel port interface (see Figure 14-4).
- Software mode** disables all hardware handshaking so that handshaking can be performed by software.
- Compatibility mode** Compatibility mode hardware handshaking can be enabled during a forward data transfer.
- You can change the mode selection at any time, but if a Compatibility mode operation is currently in-progress, it will be completed as a normal operation. Mode should be changed from Compatibility mode to another mode only when BUSY is high level. This ensures that there is no parallel port activity while the parallel port is being re-configured.
- ECP-without-RLE mode** ECP mode hardware handshaking without RLE support can be enabled during forward or reverse data transfers. You can change the mode selection at any time, but if an ECP cycle is currently in progress, it will be completed as a normal operation.
- ECP-with-RLE mode** ECP mode hardware handshaking with RLE support can be enabled during forward or reverse data transfers. Changing the mode doesn't affect current data transfer operation, including compression/decompression, until data transfer operation is completed. To abort an operation immediately, set the software-reset bit, PPCON[0], to "1".
- [4] ECP direction
- This bit determines whether the direction of ECP is forward or reverse. If this bit is set to '1', then the reverse direction is operated.

[5] Error cycle	<p>The error cycle bit is used to execute an error cycle in compatibility mode. When PPCON[5] is set to "1", the BUSY status bit in the parallel port status register, PPSTAT[5], is set to "1". This immediately causes the KS32C6200 to drive the BUSY to high level. If you set the error cycle bit while a compatibility mode handshaking sequence is in progress, the PPSTAT[5] will remain to be set to one beyond the end of the current cycle.</p> <p>The error cycle bit does not affect the nACK pulse if it is already active, but it will delay an nACK pulse if it is about to be generated. When PPCON[5] is "1", software can set or clear the parallel port status register control bits: PPSTAT[0] (nFAULT control), PPSTAT[1] (SELECT control), and PPSTAT[2] (PERROR control). When PPCON[5] is cleared to "0", the parallel port interface controller generates a delayed nACK pulse and makes BUSY low active to finish the error cycle</p>
[6] Data bus output enable	<p>The parallel port data bus output enable bit performs two functions:</p> <ol style="list-style-type: none">1) It controls the state of the tri-state output drivers.2) It qualifies the data latching from the output drivers into the parallel port data register's data field, PPDATA[7:0]. <p>When PPCON[6] is "0", the parallel port data bus lines, PPD[7:0] are disabled. This allows data to be latched onto the PPDATA's data field. When PPCON [6] is "1", the PPD[7:0] is enabled and data is prevented from being latched onto the PPDATA's data field. In this frozen state, the data field is unaffected by the transition of nSTROBE.</p> <p>The setting of the abort bit, PPCON[7], affects the operation of the data bus output enable bit, PPCON[6]. If PPCON[7] is "1", the nSELECTIN must remain high to allow PPCON[6] to be set, or to remain set. If PPCON[6] is "1" and nSELECTIN goes low, the PPCON[6] is cleared and setting this bit will have no effect.</p>
[7] Abort	<p>The abort bit causes the parallel port interface controller to use nSELECTIN to detect the time when the host suddenly aborts a reverse transfer and returns to compatibility mode; If PPCON[7] is "1", the low level on nSELECTIN causes the parallel port data bus output enable bit PPCON[6] to be cleared, and the output drivers for the data bus lines PPD[7:0] to be tri-stated.</p>
[8] DMA selection	<p>The PPIC can issue a DMA request in compatibility mode, ECP-without-RLE mode, or in ECP-with-RLE mode, if the DMA request enable bit PPCON[9] is set to one. The DMA selection bit determines which DMA channel is used for data transfer. When PPCON[8] is "0", the DMA channel 0 is used; when it is "1", the DMA channel 1 is used.</p>
[9] DMA request enable	<p>When this bit is set to "1", the PPIC issues a DMA request to DMA channel 0 or 1 during a data transfer. Otherwise, an interrupt is requested for the data transfer.</p>

[10] Flush request	When this bit is set to "1", the PPIC issues a flush request to send the remaining data to the parallel port. The remaining data is the run-length code and the data in the PPIC's buffer while reverse ECP-with-RLE mode is operating.
[12] Zero insert	When the run-length count is '0', this bit specifies whether or not to send the RLE count during ECP-with-RLE reverse data transfers. If this bit is set to '1', then the count "0" will be sent. Otherwise, it will not be sent.
[13] RLE status	This bit indicates that the run-length decompression is taking place during forward data transfers in ECP-with-RLE mode. It is set when a run-length count is received and loaded into the internal counter, and cleared when the last read of the PPDATA data field occurs.
[14] Data latch status	If a data is latched to PPDATA, then this bit is set to '1'. It is automatically cleared to zero when the PPDATA is read.
[15] Data empty	In reverse ECP mode, this bit specifies the PPDATA is empty. It is automatically cleared to zero while the PPDATA is written with a new data.

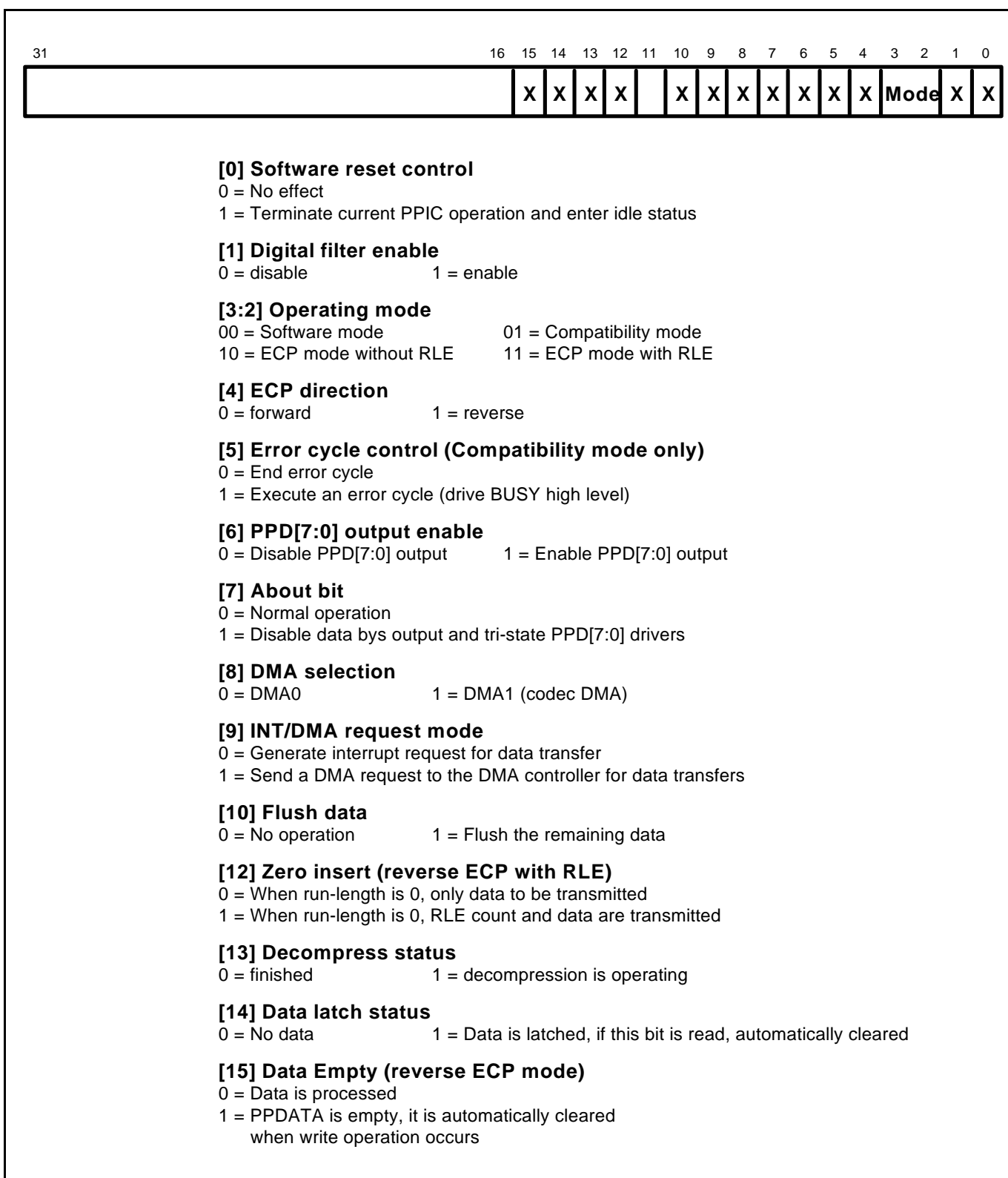


Figure 10-7. Parallel Port Control Register

PARALLEL PORT INTERRUPT EVENT REGISTERS

The two parallel port interrupt event registers, PPINTEN and PPINTPND, control interrupt-related events for the input signal originating from the host, as well as data reception, command reception, and invalid events. The parallel port interrupt enable register, PPINTEN, contains the interrupt enable bits for each interrupt event that is indicated by the PPINTPND status bits. If the PPINTEN enable bit is "1", the corresponding event causes the KS32C6200 CPU to generate an interrupt request. Otherwise, no interrupt request is issued.

NOTE

To clear the corresponding pending bit to zero after a interrupt service routine, write the pending bit to one. The value of the pending bit is changed from one to zero automatically.

Register	Offset Address	R/W	Description	Reset Value
PPINTEN	0xb010	R/W	Parallel port interrupt enable register	0x00000000
PPINTPND	0xb014	R/W	Parallel port interrupt pending register	0x00000000

[0] nSLCTIN Low-to-High	The bit of PPINTPND is set when a Low-to-High transition on nSLCTIN is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.
[1] nSLCTIN High-to-Low	The bit of PPINTPND is set when a High-to-Low transition on nSLCTIN is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.
[2] nSTROBE Low-to-High	The bit of PPINTPND is set when a Low-to-High transition in the nSTROBE is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.
[3] nSTROBE High-to-Low	The bit of PPINTPND is set when a High-to-Low transition in the nSTROBE is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.
[4] nAUTOFD Low-to-High	The bit of PPINTPND is set when a Low-to-High transition in the nAUTOFD is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.
[5] nAUTOFD High-to-Low	The bit of PPINTPND is set when a High-to-Low transition in the nAUTOFD is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.
[6] nINIT Low-to-High	The bit of PPINTPND is set when a Low-to-High transition in the nINIT is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.
[7] nINIT High-to-Low	The bit of PPINTPND is set when a High-to-Low transition in the nINIT is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.

[8] Data received	The bit of PPINTPND is set when data is latched into the PPDATA register's data field. This occurs during every High-to-Low transition of nSTROBE when the parallel port data bus enable bit, PPCON[6], is "0". An interrupt is also generated if the ECP-with-RLE mode is enabled, and if a data decompression is in progress.
[9] Command received	The bit of PPINTPND is set when a command byte is latched into the PPDATA register data field. If ECP-without-RLE mode is enabled, the command received interrupt is issued whenever a run-length or channel address is received. If ECP-with-RLE mode is enabled, the command received interrupt is issued only when a channel address is received. This event can be posted only when ECP mode is enabled. The corresponding enable bit in the PPINTEN register determines whether or not an interrupt request will be generated when a command byte is received.
[10] Invalid transition	The bit of PPINTPND is set when nSLCTIN transitions high-to-low in the middle of an ECP forward data transfer handshaking sequence. This interrupt is issued if nSLCTIN is low when nSTROBE is Low or when BUSY is high. This event can be detected only when ECP mode is enabled.
[11] Transmit Data Empty	The bit of PPINTPND is set to one when the transmit data register (=PPDATA) can be written during an ECP reverse data transfers

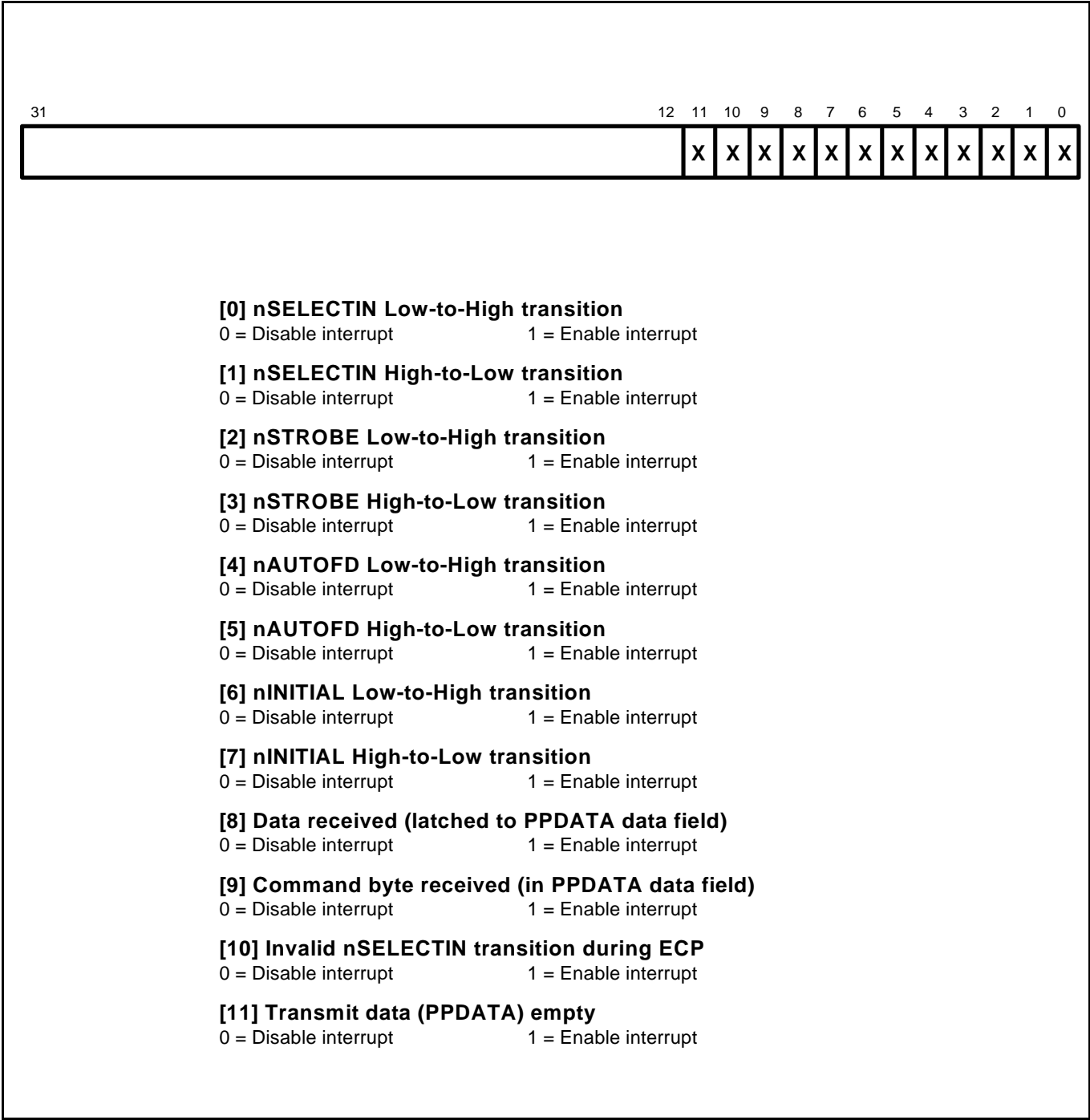


Figure 10-8. Parallel Port Event Interrupt Enable Register (PPINTEN)

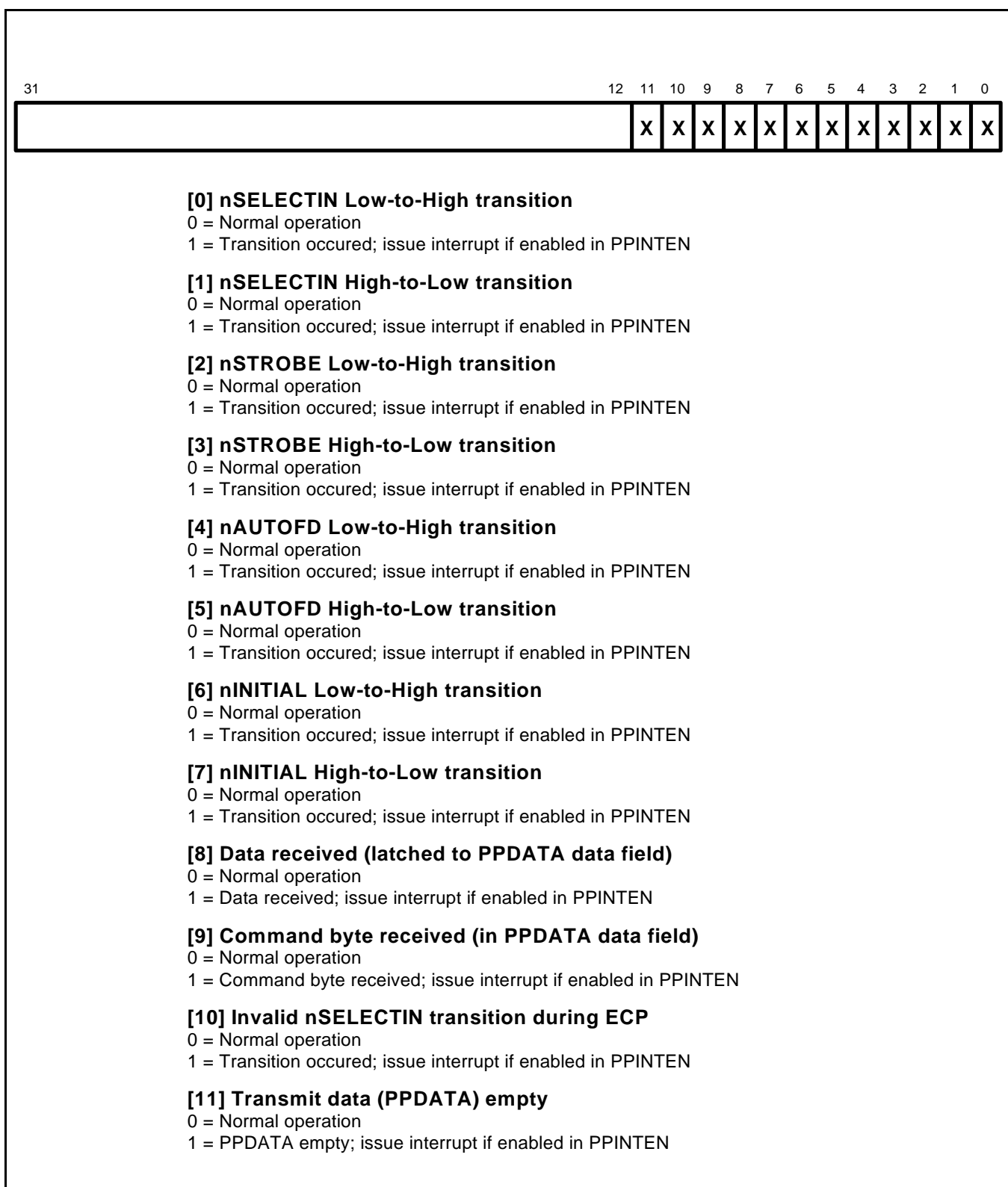


Figure 10-9. Parallel Port Event Interrupt Pending Register (PPINTPND)

NOTES

11

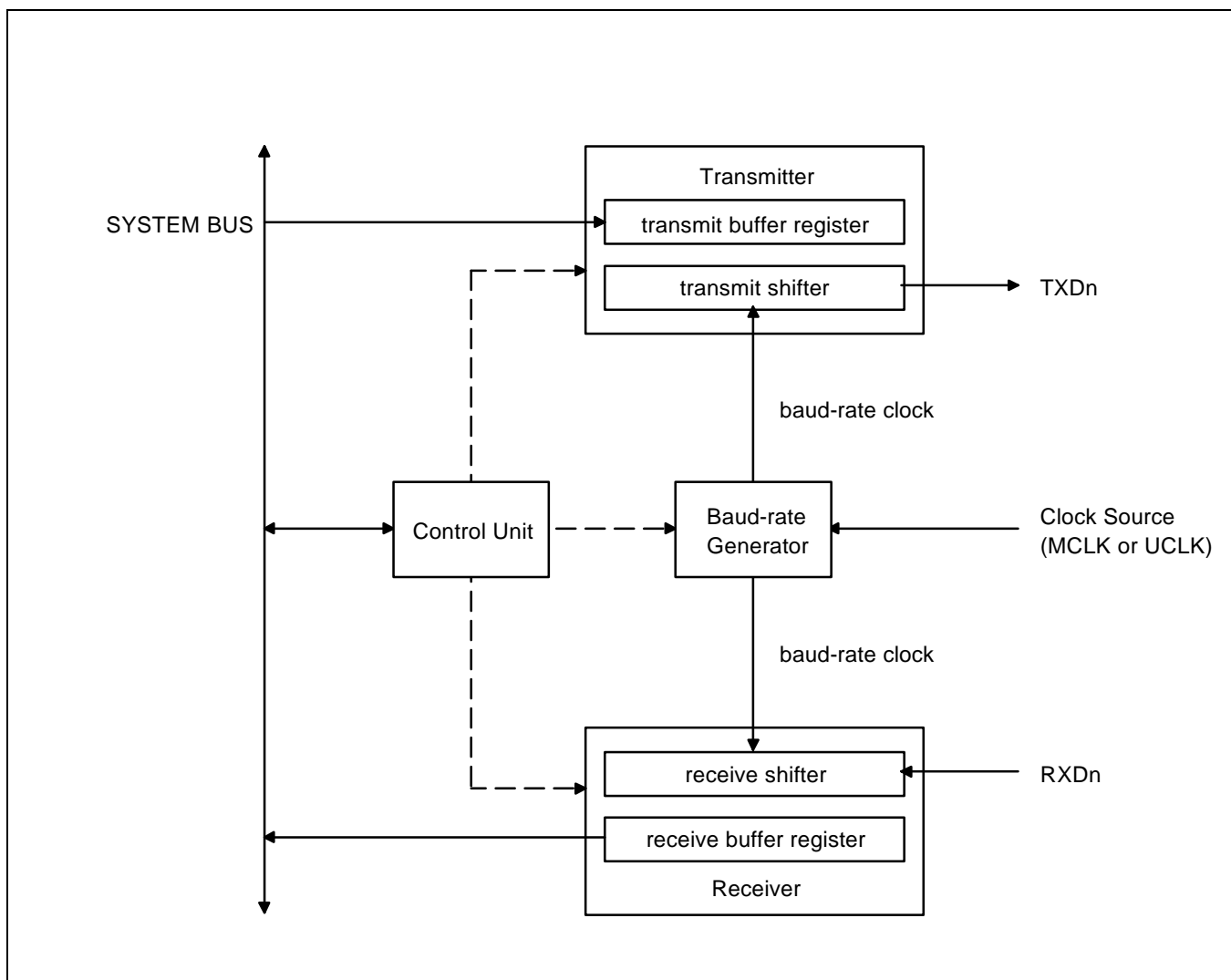
UART

OVERVIEW

The KS32C6200 UART (Universal Asynchronous Receiver and Transmitter) unit provides two independent asynchronous serial I/O (SIO) ports, each of which can operate in interrupt-based or DMA-based mode, i.e. SIO can generate interrupt or DMA request to transfer data between CPU and SIO.

The KS32C6200 UART includes programmable baud-rates, infra-red (IR) transmit/receive, one or two stop bit insertion, 5-bit, 6-bit, 7-bit or 8-bit data transfers, and parity checking.

Each SIO contains a baud-rate generator, transmitter, receiver and control unit, as shown in Figure11-1. The baud-rate generator can be clocked by either the internal system clock (MCLK) or the external clock, UCLK input from the UCLK pin. The transmitter and the receiver contain data buffer registers and data shifters. Data, which is to be transmitted, is written to the transmit holding register and then copied to the transmit shifter. It is then shifted out by the transmit data pin (TXDn). The received data is shifted by the receive data pin (RXDn), and then copied to the receive buffer register from the shifter once one data byte has been received. The control unit provides controls for mode selection and status/interrupt generation.

**Figure 11-1. Serial I/O Block Diagram**

UART OPERATION

The following sections describe the UART operations that include infra-red mode, loopback mode, interrupt generation, baud-rate generation, data transmission, data reception and so on.

Infra-red mode

The KS32C6200 UART block supports infra-red (IR) transmit and receive, which can be selected by setting the infra-red-mode bit in the line control register (ULCONn). The implementation of the mode is shown in Figure 11-2.

In IR mode, the transmit period is pulsed at a rate of 3/16 as is at the normal serial transmit rate (when the transmit data value in the UTXBUF register is zero); in IR receive mode, the receiver must detect the 3/16 pulsed period to recognize a zero value in the receive buffer register, URXBUF, as the IR receive data. (refer to the frame timing diagrams shown in Figure 11-15 and 11-16)

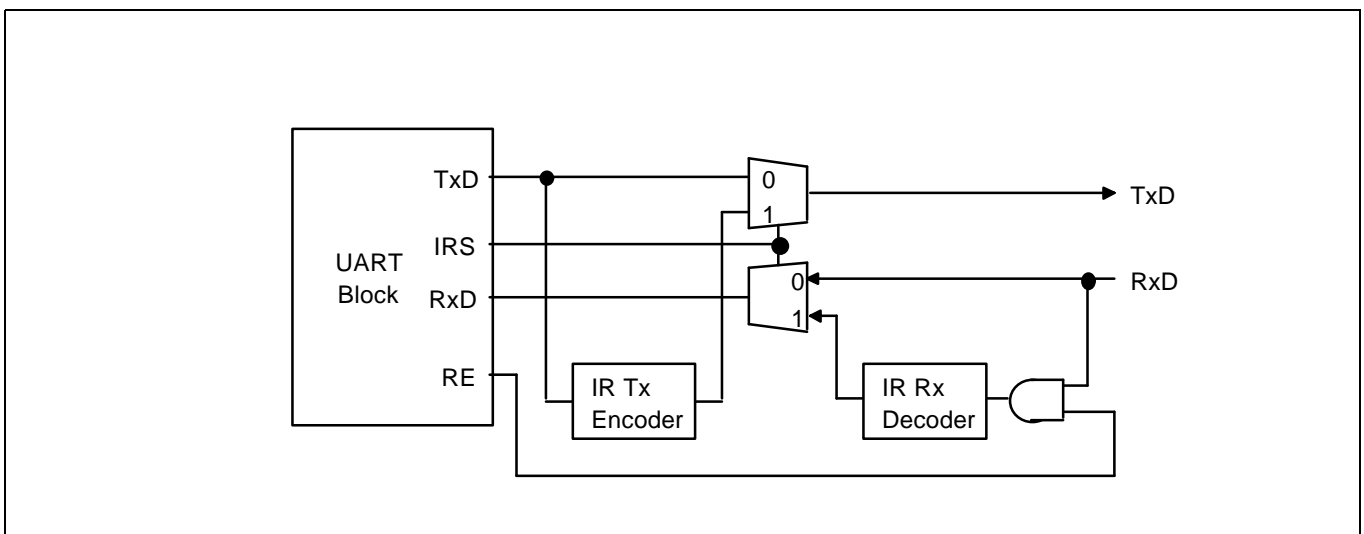


Figure 11-2. UART Block Diagram

Loopback mode

The KS32C6200 UART provides a test mode referred as the loopback mode to aid in isolating faults in the communication link. In this mode, the transmitted data is immediately received. This feature allows the processor to verify the internal transmit and to receive the data path of each SIO channel. This mode can be selected by setting the loopback-bit in the UART control register (UCONn).

INTERRUPT/DMA REQUEST GENERATION

Each SIO of KS32C6200 UART has seven status signals: overrun error, parity error, frame error, break, receive buffer full, transmit buffer register empty and transmitter empty, all of which are indicated by the corresponding UART status register (USTATn).

The overrun error, parity error, frame error and break condition are referred to as the receive status, each of which can cause the receive status interrupt request, if the receive-status-interrupt-enable bit is set to one in the control register UCONn. When a receive-status-interrupt-request is detected, you can know the signal which causes the request by reading the status register (USTATn).

When the receiver transfers the data of the receive shifter to the receive buffer register, it activates the receive buffer full status signal which will cause the receive interrupt, if the receive mode in control register is selected as the interrupt mode. When the transmitter transfers data from its transmit buffer register to its shifter, the transmit holding register empty status signal is activated. The signal causes the transmit interrupt if the transmit mode in control register is selected as interrupt mode.

The receive-buffer-full and transmit-buffer-register empty status signals can also be connected to generate the DMA request signals if the receive/transmit mode in the control register is selected as the DMA mode.

As mentioned before, two DMA channels, DMA0 and DMA1, are provided in the KS32C6200. Each SIO can be connected with a fixed DMA channel. In other words, the SIO0 can only generate the DMA0 request and the SIO1 can only generate the DMA1 request.

BAUD-RATE GENERATION

Each SIO's baud-rate generator provides the serial clock for transmitter and receiver. The source clock for the baud-rate generator can be selected with the KS32C6200's internal system clock (MCLK) or the external clock input (UCLK), which is determined by the serial-clock-selection bit in UART line control register (ULCONn). The baud-rate clock is generated by dividing the source clock by 16 and a 16-bit divisor specified by the UART baud-rate divisor register (UBRDIVn). The UBRDIVn can be determined as follows:

$$\text{UBRDIVn} = (\text{int})(\text{source_clock} / (\text{bps} \times 16)) - 1$$

where the divisor should be from 1 to ($2^{16}-1$). For example, if the baud-rate is 56,000 bps and MCLK is 33 MHz (use internal system clock), UBRDIVn is:

$$\begin{aligned}\text{UBRDIVn} &= (\text{int})(33000000 / (56000 \times 16)) - 1 \\ &= (\text{int})(36.83) - 1 \\ &= 36 - 1 = 35\end{aligned}$$

DATA TRANSMISSION

The data frame for transmission is programmable. It consists of a start bit, 5 to 8 data bits, an optional parity bit and 1 to 2 stop bits, which can be specified by the line control register (ULCONn). The transmitter can also produce the break condition. The break condition forces the serial output to logic 0 state for a duration longer than one frame transmission time. At the receiving end, the break condition sets an error flag as mentioned above.

The data transmission process is shown in Figure 11-3. The transmitter transfers data through a path as follows: data source -> transmit buffer register -> transmit shifter -> TXDn pin. It then completes the parallel-to-serial data conversion. Two flags (status signals), transmit buffer register empty and transmitter empty, are used to indicate the status of the transmit buffer register and transmitter.

DATA RECEPTION

Like the transmission, the data frame for reception is also programmable. It consists of a start bit, 5 to 8 data bits, an optional parity bit and 1 to 2 stop bits by settings in the line control register (ULCONn). The receiver can detect overrun error, parity error, frame error and break condition, each of which can set an error flag.

The overrun error indicates that new data has overwritten the old data before the old data has been read. The parity error indicates that the receiver has detected a parity condition other than what it was programmed for. The frame error indicates that the received data does not have a valid stop bit. The break condition indicates that the RXDn input is held in the logic 0 state for a duration longer than one frame transmission time.

The data reception process is shown in Figure 11-4. The receiver transfers data through a path as follows: RXDn pin -> receive shift register -> receive buffer register -> destination. This completes the serial-to-parallel data conversions. In addition to receive-error-status flags, a receive-buffer-full flag is used to indicate the status of the receive buffer register.

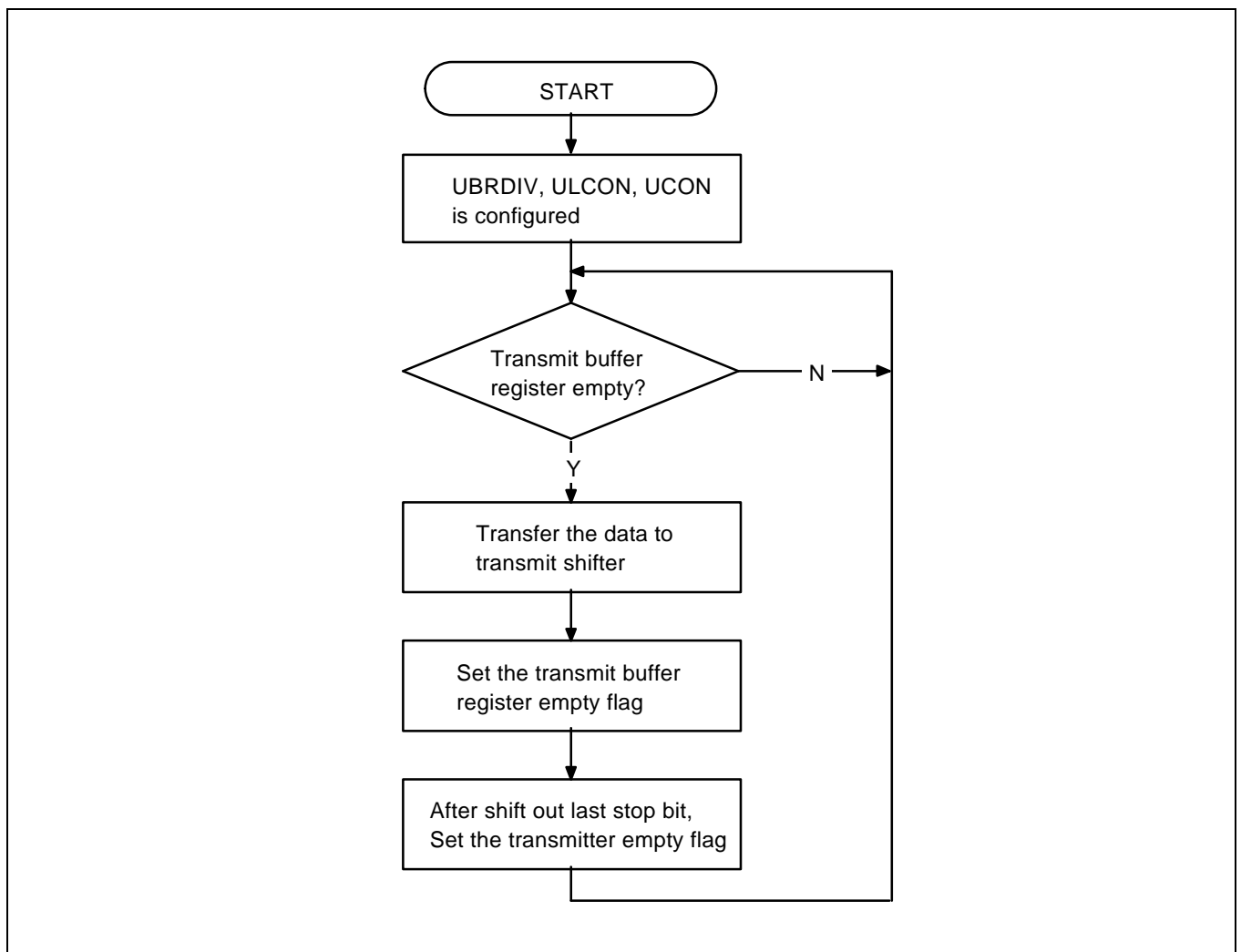


Figure 11-3. UART Data Transmission Process

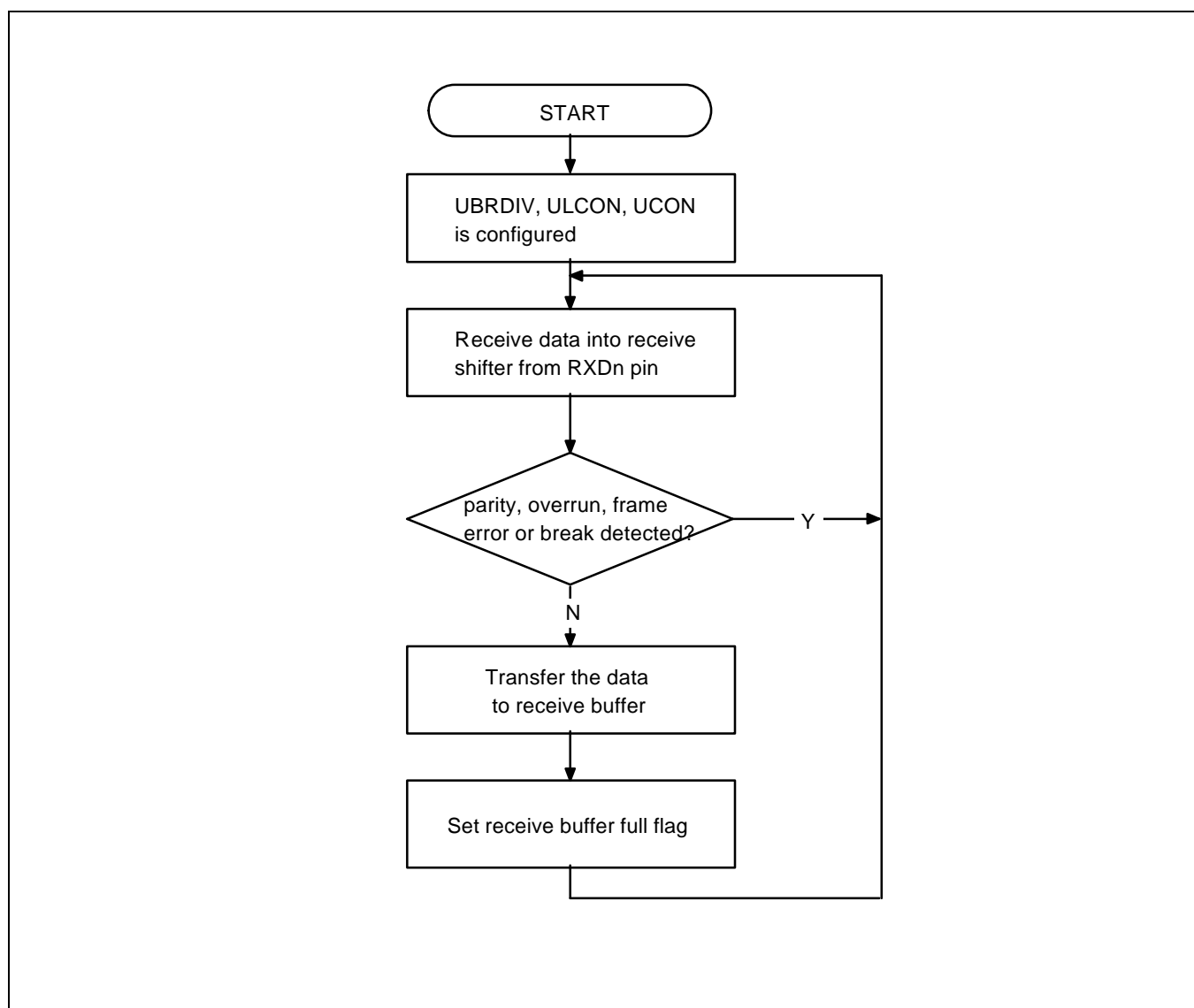


Figure 11-4. UART Data Reception Process

UART SPECIAL REGISTERS

UART LINE CONTROL REGISTER

There are two identical UART line control registers, ULCON0, 1, in the UART block, each for a SIO channel.

Register	Offset Address	R/W	Description	Reset Value
ULCON0	0xe000	R/W	UART channel 0 line control register	0x00
ULCON1	0xe800	R/W	UART channel 1 line control register	0x00

[1:0] Word Length (WL)

The word length indicates the number of data bits to be transmitted or received per frame.

00 = 5 bits
01 = 6 bits
10 = 7 bits
11 = 8 bits

[2] Number of stop bit

The number of stop bit specifies how many stop bits are used to signal end-of-frame (EOF).

0 = One stop bit per frame
1 = Two stop bit per frame

[5:3] Parity Mode (PMD)

The parity mode specifies how parity generation and checking are to be performed during UART transmit and receive operations.

0xx = No parity
100 = Odd parity
101 = Even parity
110 = Parity forced/checked as "1"
111 = Parity forced/checked as "0"

[6] Serial Clock Selection

This selection bit specifies the clock source.

0 = Internal (MCLK)
1 = External (UCLK)

[7] Infra-Red Mode

The infra-red mode determines whether or not to use the infra-red mode.

0 = Normal Mode Operation
1 = Infra-red Tx/Rx Mode

NOTE: The ULCONn has to be configured before the UCONn is configured.

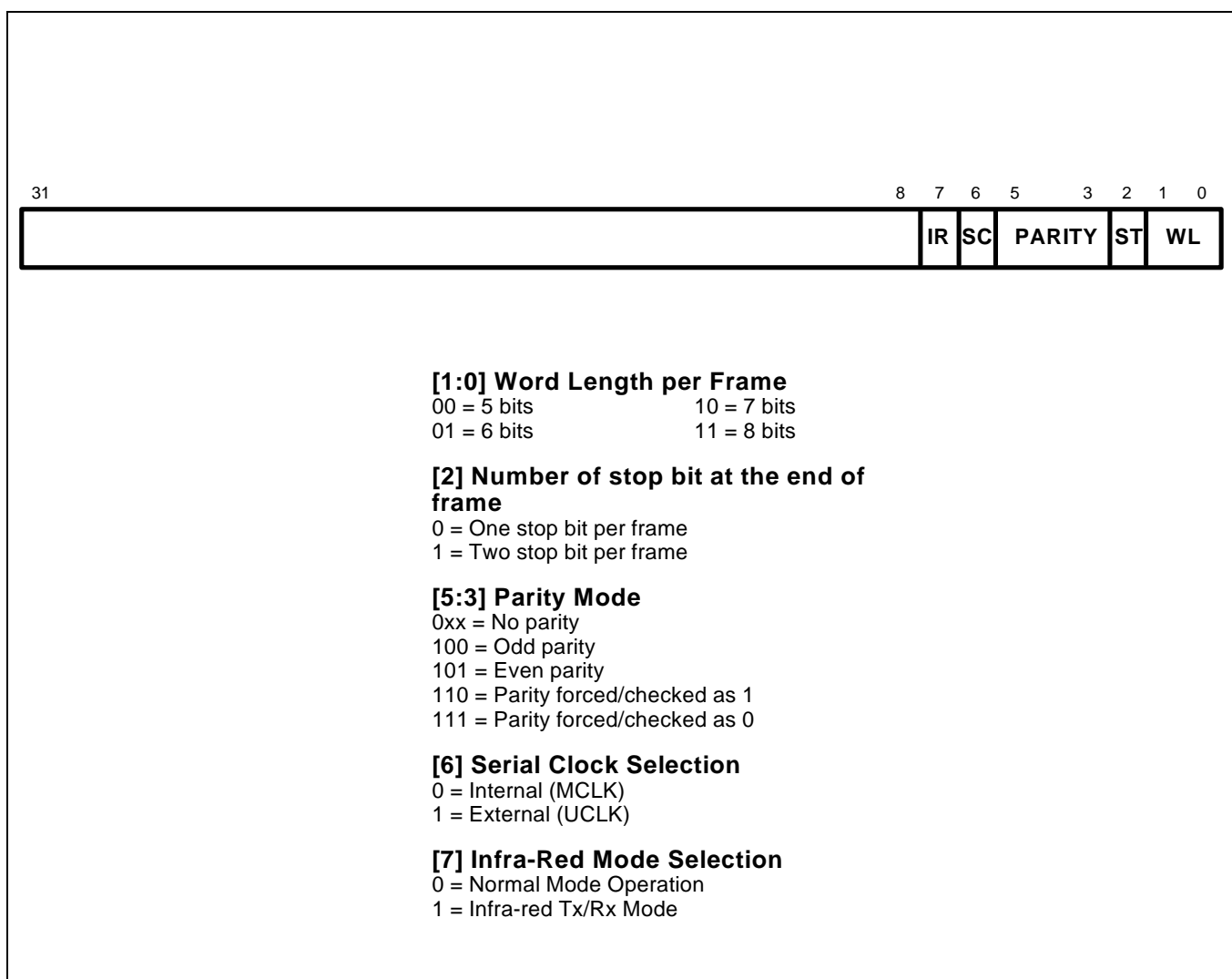


Figure 11-5. UART Line Control Register (ULCON0,1)

UART CONTROL REGISTER

There are two identical UART control registers (UCON0,1) in the UART block, each for a SIO channel. The UCONn has to be configured after the ULCONn is configured.

Register	Offset Address	R/W	Description	Reset Value
UCON0	0xe000	R/W	UART channel 0 line control register	0x00
UCON1	0xe800	R/W	UART channel 1 line control register	0x00

[1:0] Receive Mode (RxM)

The RxM determines which function is currently able to read data from the UART receive buffer register, URXBUF. The difference between UCON0 and UCON1 should be noted. SIO0 can only generate a DMA0 request and SIO1 can only generate a DMA1 request.

For UCON0

00 = disable SIO0
01 = interrupt request
10 = DMA0 request
11 = undefined

For UCON1

00 = disable SIO1
01 = interrupt request
10 = undefined
11 = DMA1 request

NOTE: Even if you do not use the interrupt request, the interrupt request is selected to use SIO.
If you don't need a interrupt request, you can disable the interrupt by configuring the interrupt mask register (INTMSK).

[2] Rx Status Interrupt Enable

This bit enables the UART to generate an interrupt if an exception, such as a break, frame error, parity error, or overrun error, occurs during a receive operation.

0 = do not generate receive status interrupt
1 = generate receive status interrupt

[4:3] Transmit Mode (TxM)

This TxM determines which function is currently able to write Tx data to the UART transmit buffer register, UTXBUF. The difference between UCON0 and UCON1 should be noted. SIO0 can only generate a DMA0 request and SIO1 can only generate a DMA1 request.

For UCON0

00 = disable SIO0
01 = interrupt request
10 = DMA0 request
11 = not used

For UCON1

00 = disable SIO1
01 = interrupt request
10 = not used
11 = DMA1 request

NOTE: Even if you do not use the interrupt request, the interrupt request is selected to use SIO.
If you don't need a interrupt request, you can disable the interrupt by configuring the interrupt mask register (INTMSK).

[6] Send Break

Setting UCON[6] causes the UART to send a break. The break is defined as a continuous low level signal on the transmit data output with a duration more than one frame transmission time.

This bit should be set to one when the transmitter is empty (transmitter empty bit, USTAT [7] = "1"). You can use the transmitter to measure a frame time interval.

When the USTAT[7] is "1", write the transmit buffer register, UTXBUF, with dummy data and then poll the USTAT[7] value. When it returns to "1", clear (reset) the send break bit UCON[6]. (You have sent the break for one frame time interval exactly.)

0 = not send break

1 = send break

[7] Loopback Bit

Setting loopback bit to 'one' causes the UART to enter loopback mode. In loopback mode, the TXDn pin is sent to the high level and the transmit buffer register (UTXBUF) is internally connected to the receive buffer register (RBR). This mode is provided for test purposes only.

0 = Normal SIO operation mode

1 = Enable SIO loopback mode (only for testing)

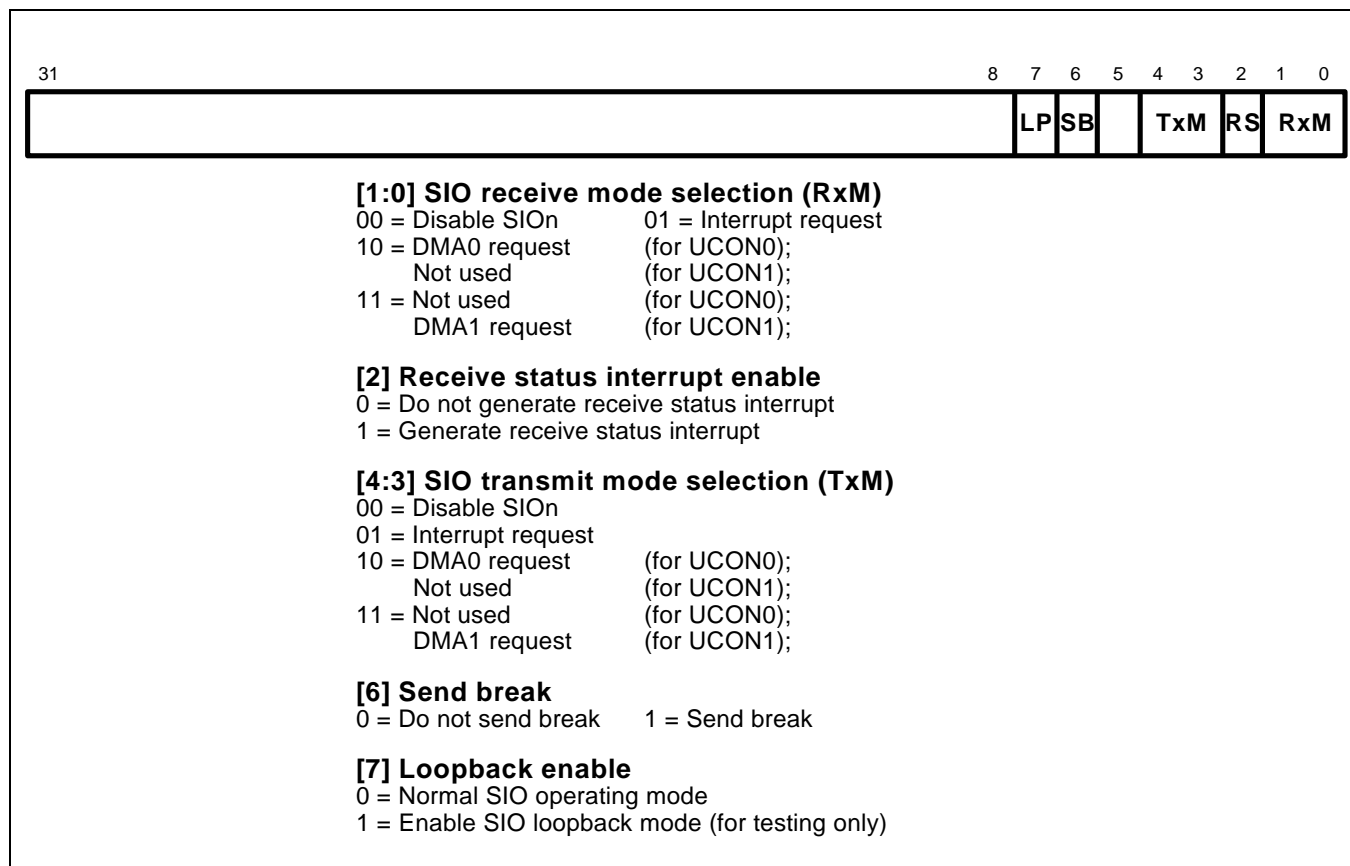


Figure 11-6. UART Control Register (UCON0,1)

UART STATUS REGISTER

There are two identical UART status registers, USTAT0,1, in the UART block, each for a SIO channel. The USTAT is a read only register that is used to monitor the status of SIO.

Register	Offset Address	R/W	Description	Reset Value
USTAT0	0xe008	R/W	UART channel 0 status register	0xc0
USTAT1	0xe808	R/W	UART channel 1 status register	0xc0

[0] Overrun Error

USTAT[0] is automatically set to "1" whenever an overrun error occurs during a serial data receive operation. If the receive-status-interrupt-enable bit CON[2] is "1", and an overrun error occurs, a receive status interrupt will be generated. This bit is automatically cleared to "0" whenever the UART status register (USTAT) is read.

[1] Parity Error

USTAT[1] is automatically set to "1" whenever a parity error occurs during a serial data receive operation. If the receive-status-interrupt-enable bit UCON[2] is "1", and a parity error occurs, a receive status interrupt will be generated. This bit is automatically cleared to "0" whenever the UART status register (USTAT) is read.

[2] Frame Error

USTAT[2] is automatically set to "1" whenever a frame error occurs during a serial data receive operation. If the receive status-interrupt-enable bit UCON[2] is "1", and a frame error occurs, a receive status interrupt will be generated. The frame error bit is automatically cleared to "0" whenever the UART status register (USTAT) is read.

[3] Break Interrupt

USTAT[3] is automatically set to "1" to indicate that a break signal has been received. If the receive status interrupt enable bit UCON[2] is "1", and a break occurs, a receive status interrupt will be generated. The break interrupt bit is automatically cleared to "0" when you read the UART status register.

[5] Receive Data Ready

USTAT[5] is automatically set to "1" whenever the receive data buffer register (RBR) contains valid data received over the serial port. The receive data can then be read from the RBR. When this bit is "0", the RBR does not contain valid data. Depending on the current setting of the SIO receive mode bits, UCON[1:0], an interrupt or a DMA request is generated when USTAT[5] is "1".

[6] Tx Buffer Register Empty

USTAT[6] is automatically set to "1" when the transmit buffer register (THR) does not contain valid data. In this case, the THR can be written with the data to be transmitted. When this bit is "0", the THR contains valid Tx data that has not been copied to the transmit shift register. In this case, the THR cannot be written with new Tx data. Depending on the current setting of the SIO transmit mode bits, UCON[4:3], an interrupt or a DMA request will be generated when USTAT[6] is "1".

[7] Transmitter Empty (T)

USTAT[7] is automatically set to "1" when the transmit buffer register has no valid data to transmit and the Tx shift register is empty. When the transmitter empty bit is "1", it indicates that the transmitter function block is not used and you can manipulate the setting of the transmitter function block.

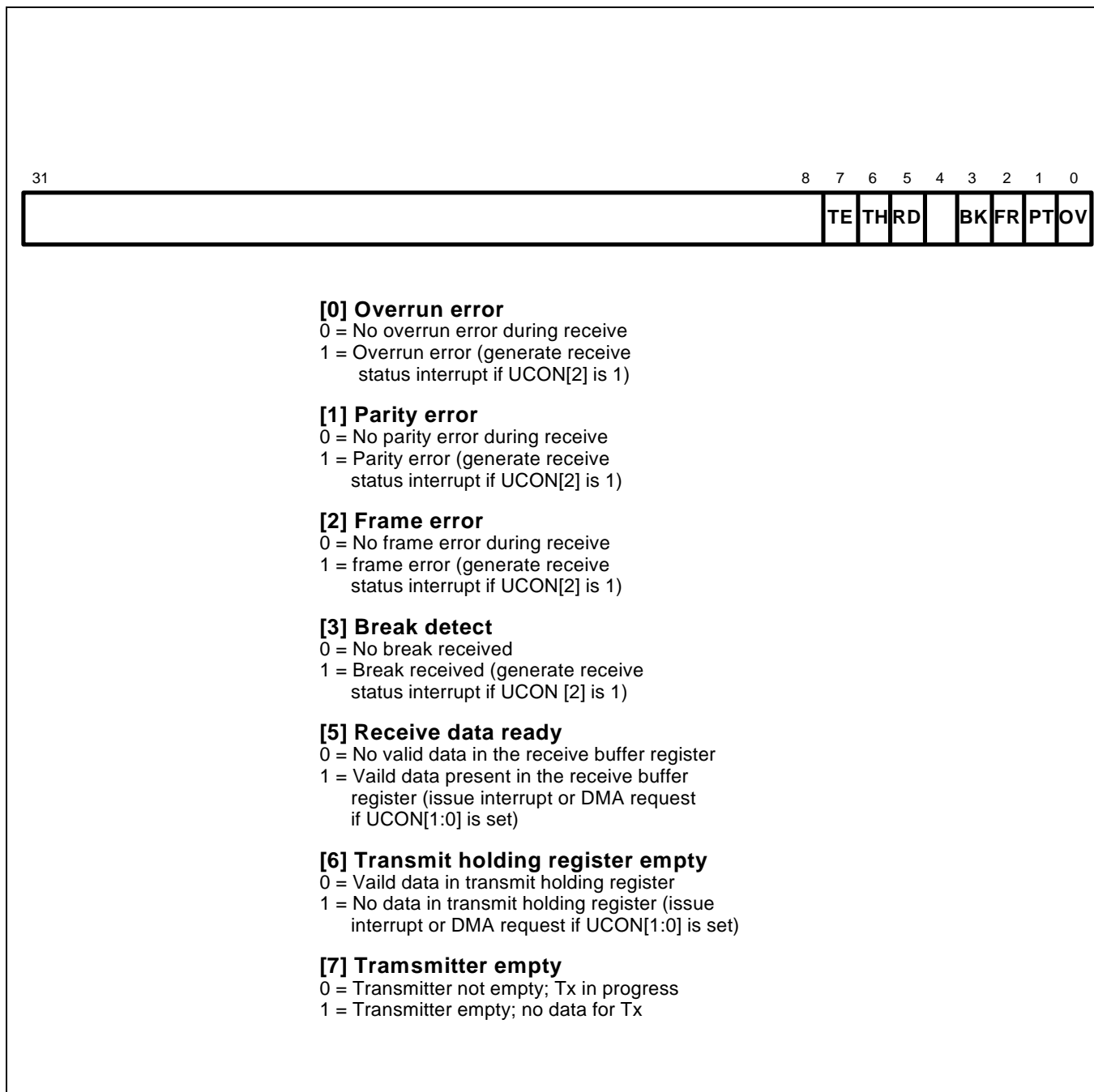


Figure 11-7. UART Status Register (USTAT0,1)

UART TRANSMIT BUFFER REGISTER

There are two identical UART transmit buffer registers, UTXBUF, in the UART block for two SIO channels, each of which contains an 8-bit data value to be transmitted over the SIO channel.

In DMA-based transmit mode, as the destination of the DMA channel, the address of the transmit buffer register should be set to one, into the DMA destination address register.

Register	Offset Address	R/W	Description	Reset Value
UTXBUF0	e00c	W	UART channel 0 transmit buffer register	0xxx
UTXBUF1	e80c	W	UART channel 1 transmit buffer register	0xxx

[7:0] Transmit Data

This field contains the data to be transmitted by the corresponding SIO channel. When this register is written, the transmit buffer register empty bit in the status register, USTAT[6], should be set to "0". This prevents overwriting transmit data that may already be present in the URXBUF. Whenever the UTXBUF is written with new value, the transmit register empty bit, USTAT[6], is automatically cleared to "0".

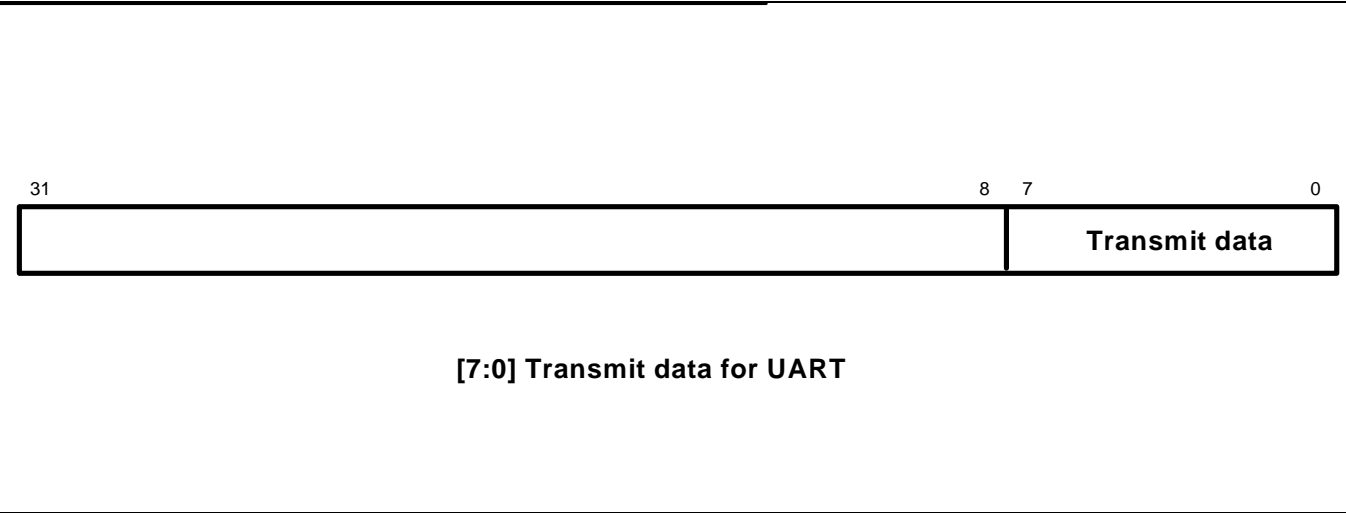


Figure 11-8. UART Transmit Buffer Register (UTXBUF0,1)

UART RECEIVE BUFFER REGISTER

There are two identical UART receive buffer registers, RBR, in the UART block for two SIO channels, each of which contains an 8-bit data value for received serial data.

In DMA-based receive mode, as the source of the DMA channel, the address of the receive buffer register should be set into the DMA source address register.

Register	Offset Address	R/W	Description	Reset Value
URXBUF0	0xe010	R	UART channel 0 receive buffer register	0xxx
URXBUF1	0xe810	R	UART channel 1 receive buffer register	0xxx

[7:0] Receive Data

This field contains the data received from the corresponding SIO channel. When UART finishes receiving a data frame, the receive data ready bit in the UART status register, USTAT[5], should be set to "1". This prevents reading invalid receive data that may already be present in the URXBUF. Whenever the RBR is read, the receive data ready bit, USTAT[5], is automatically cleared to "0".

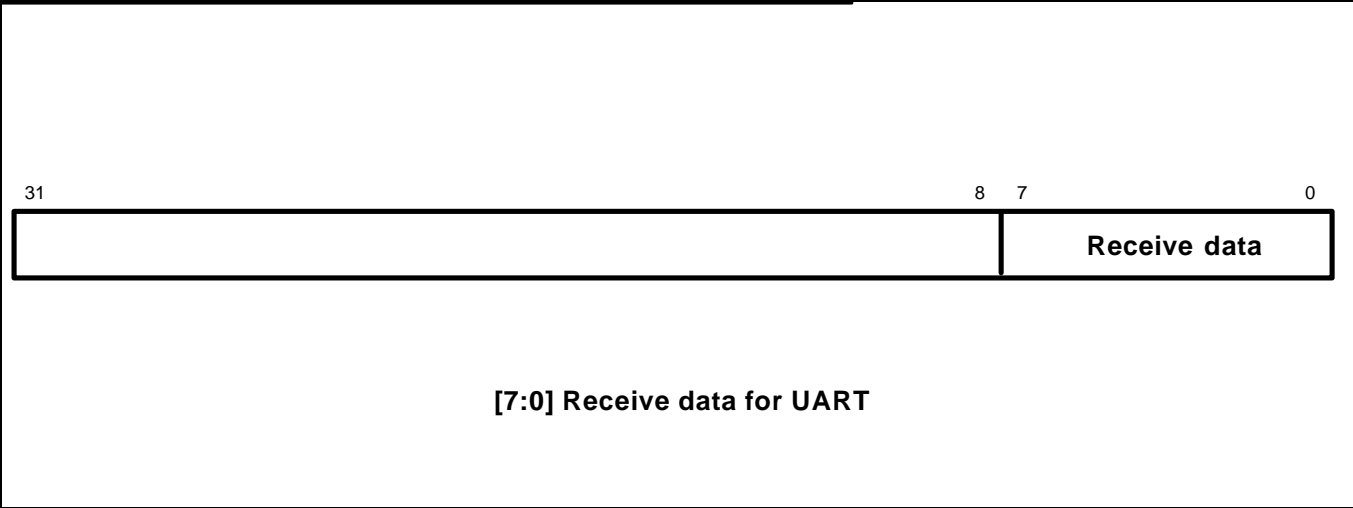


Figure 11-9. UART Receive Buffer Register (URXBUF0,1)

UART BAUD RATE DIVISOR REGISTERS

The value stored in the baud rate divisor register, UBRDIV, is used to determine the serial Tx/Rx clock rate (baud rate) as follows:

$$UBRDIVn = (int)(source_clock / (bps \times 16)) - 1$$

The source_clock is either MCLK (the internal master clock) or UCLK (the external UART clock input) and it is determined by the setting of the serial clock selection bit in the line control register, ULCON[6].

Register	Offset Address	R/W	Description	Reset Value
UBRDIV0	0xe014	R/W	Baud rate divisor register 0	0x00000001
UBRDIV1	0xe814	R/W	Baud rate divisor register 1	0x00000001

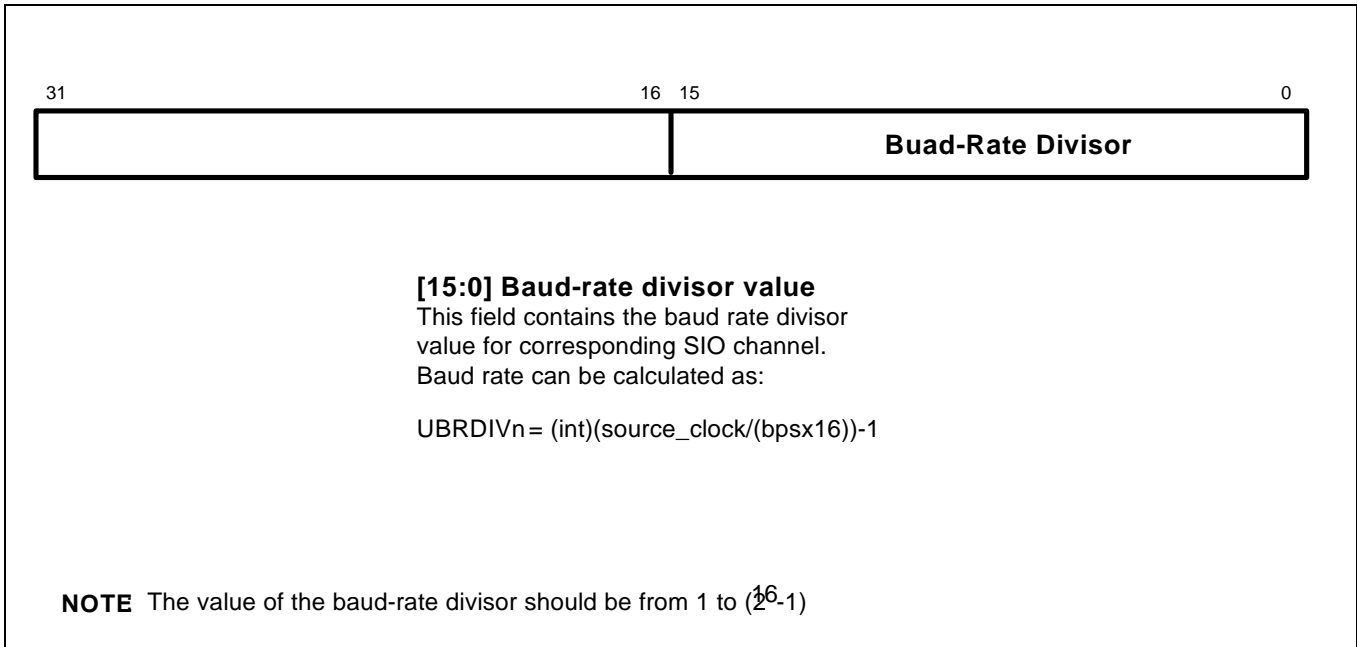


Figure 11-10. UART Baud Rate Divisor Register (UBRDIV0,1)

TIMING DIAGRAMS

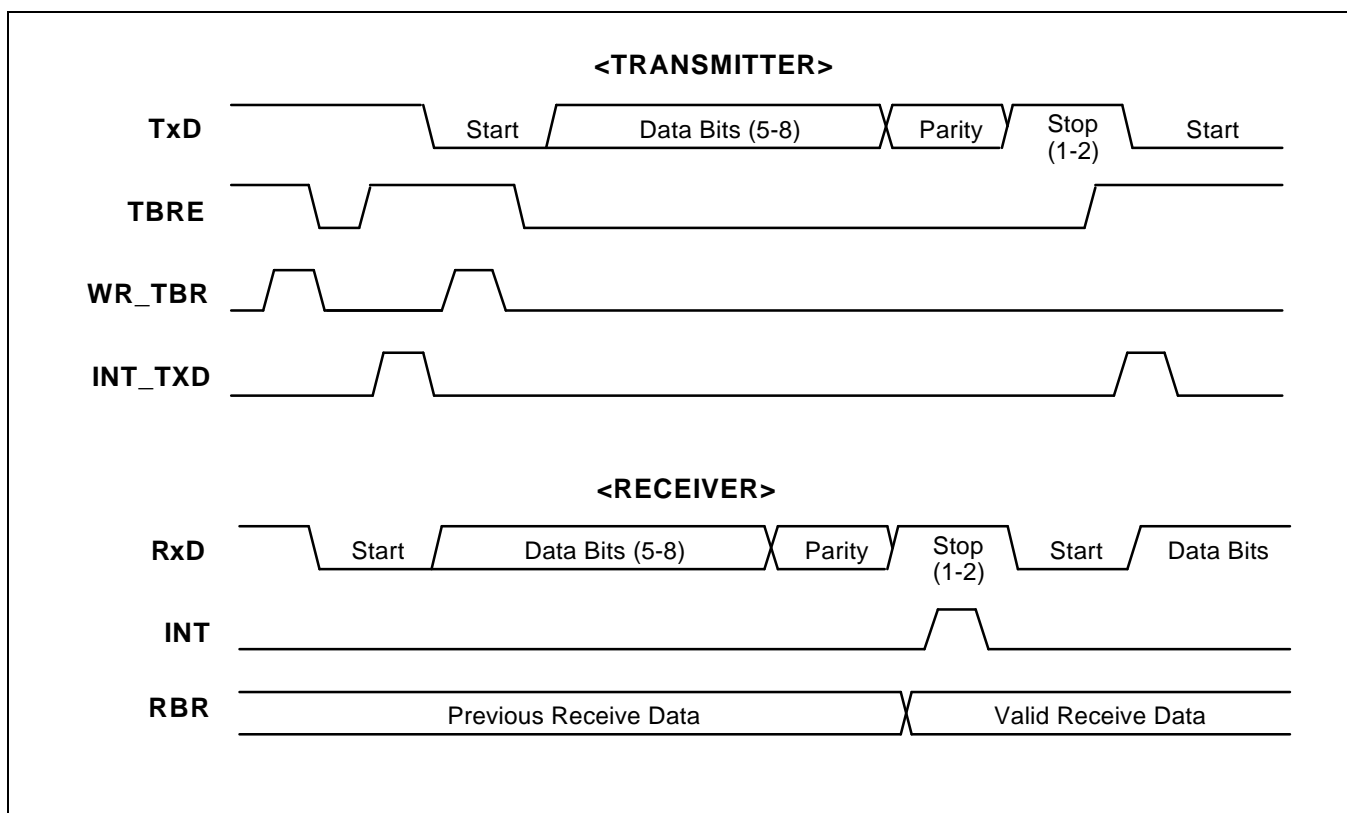


Figure 11-11. Interrupt-based Serial I/O Timing Diagram (Tx and Rx)

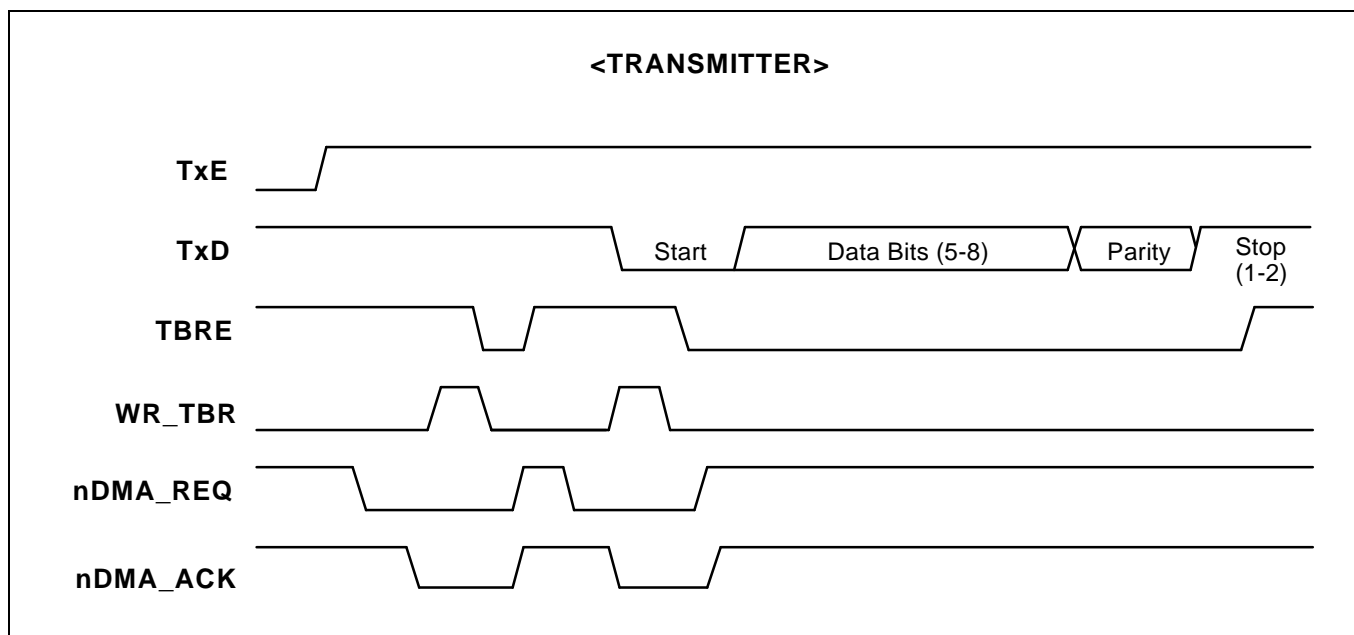


Figure 11-12. DMA-based Serial I/O Timing Diagram (Tx only)

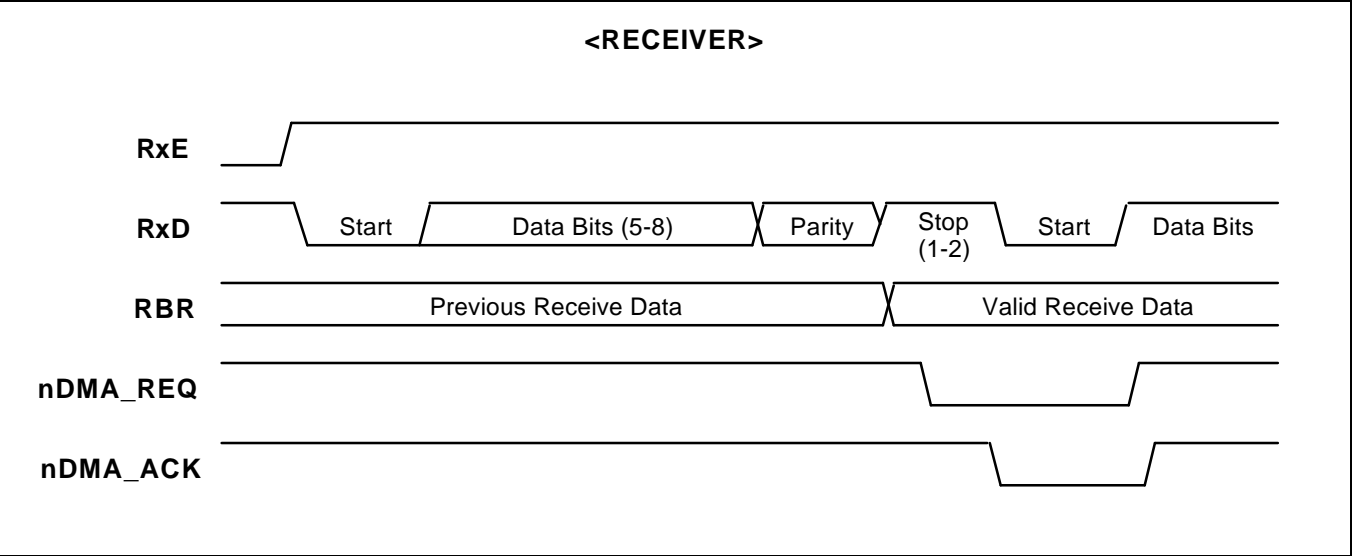


Figure 11-13. DMA-Based Serial I/O Timing Diagram (Rx only)

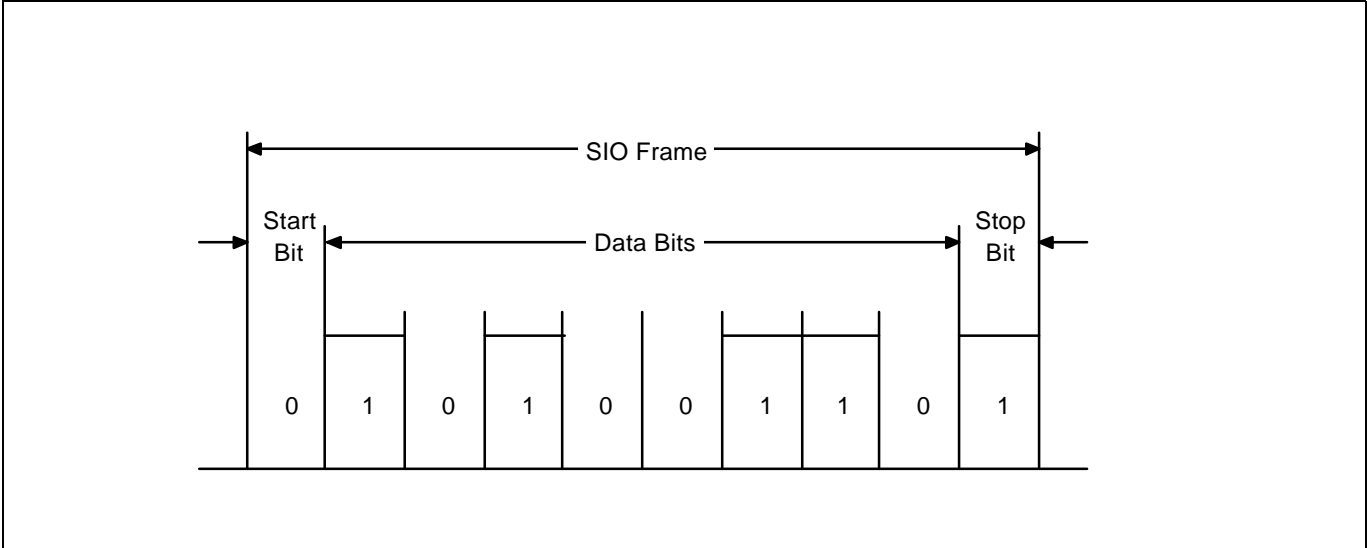


Figure 11-14. Serial I/O Frame Timing Diagram (Normal UART)

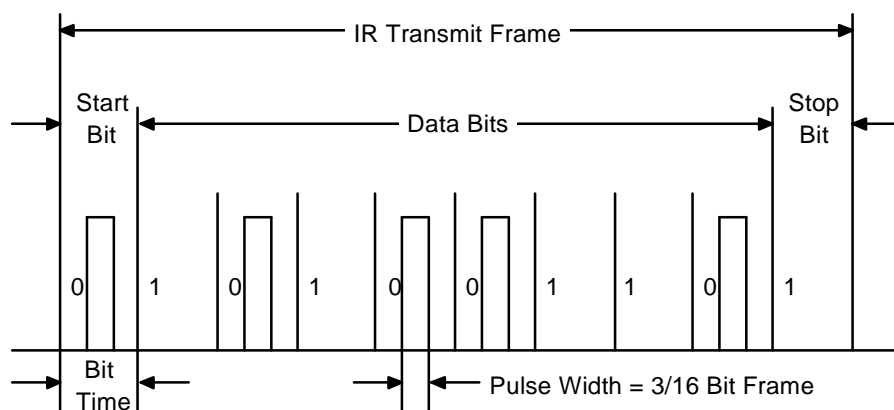


Figure 11-15. Infra-Red Transmit Mode Frame Timing Diagram

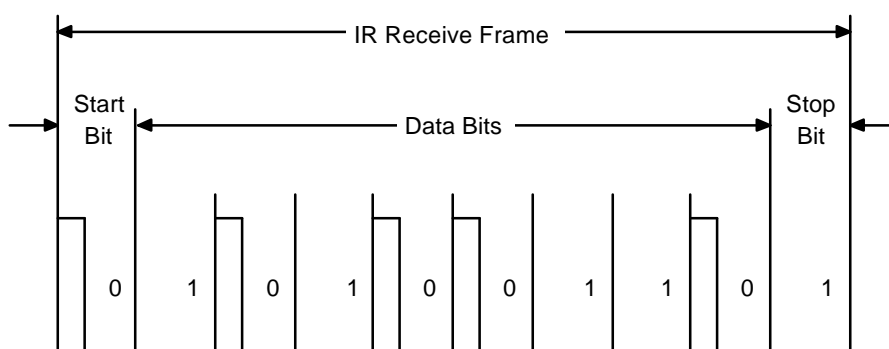


Figure 11-16. Infra-Red Receive Mode Frame Timing Diagram

12

Tone Generator

OVERVIEW

The KS32C6200 Tone Generator provides a programmable tone signal which has a 50% duty cycle. The tone signal can be used to make out 'keyclick' sound. The Tone Generator block has a tone counter which includes an 8-bit programmable divider and a 1/2 divider to make out the 50% duty cycle, and a Tone Data register (TONDATA) which has a tone enable/disable bit and tone count data bits. The 8-bit programmable divider receives MCLK/(prescaler+1)/128 clock signals and divides it depending on the count value of TONDATA [7:0] bits. You can set the prescaler value of TSTCON as shown in Figure 14-3.

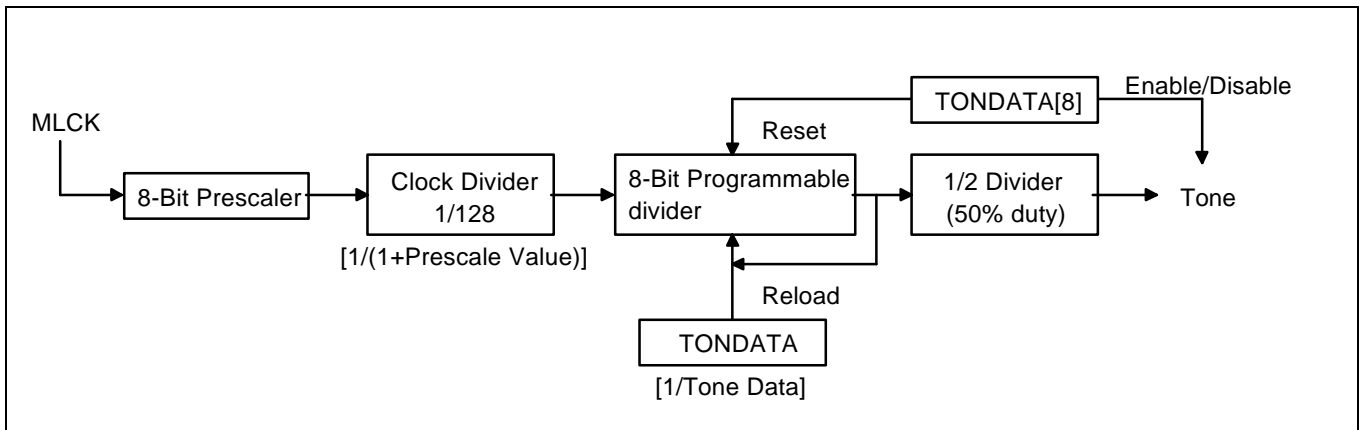


Figure 12-1. Tone Generator Block Diagram

TONDATA[8] bit enables/disables the Tone generator operation. When it is cleared to '0', the tone output is disabled (stopped) and the programmable divider is automatically cleared while the tone data register (TONDATA) retains its value. The initial value of the tone enable bit is '0'.

The input clock to the tone generator is MCLK/(prescaler+1)/128. The divided-by ratio of the tone counter is determined by the tone data register value, ranging from 0 to 255.

You have to load data into the tone data register (TONDATA) before enabling the tone generator to get the correct tone signal. To make out the 50% duty cycle tone signal, the KS32C6200 Tone Generator has a 1/2 divider with a programmable divider. The output of the programmable divider is divided by the 1/2 divider.

The frequency of the tone is calculated as follows:

$$\frac{\text{MCLK}}{(\text{Prescaler}+1) \times 128 \times \text{ToneData} \times 2}$$

Table 12-1. Tone Generator Data Value Setting (MCLK=33 MHz)

TONDATA	Tone Frequency	TONDATA	Tone Frequency
0	No tone (all high)	4	2.470 kHz
1	9.916 kHz
2	4.958 kHz	100	99 Hz
3	3.305 kHz	255	39 Hz

NOTE: The value of prescaler is 0xc.

Tone Generator Data Register (TONDATA)

The tone generator data register (TONDATA) stores an 8-bit value which determines the frequency of the tone generator output. The value in the TONDATA register determines the divided-by ratio of the programmable divider. The divided-by value, therefore, ranges from 0 to 255. The output value of the tone counter is divided by two, producing a 50% duty tone output signal. A reset clears the TONDATA value to '00h'. The tone frequency is therefore calculated, based on the tone data value, as follows:

MCLK

(Prescaler+1) x 128 x ToneData x 2

Register	Offset Address	R/W	Description	Reset Value
TONDATA	0xf004	R/W	Tone generator data register	0x00

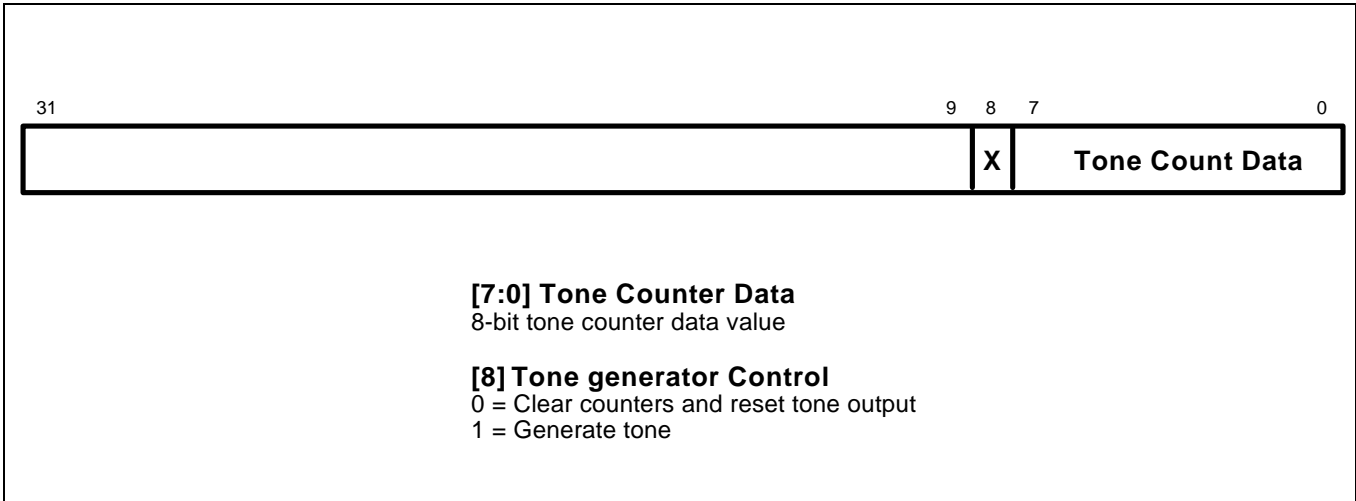


Figure 12-2. Tone Data Register (TONDATA)

13

Watch-Dog Timer

OVERVIEW

The KS32C6200 Watch-Dog Timer is used to resume controller operation when it is disturbed by malfunctions such as noise and system errors. It can be used as a normal interval timer to request interrupt services. You can set the prescaler value (initial value: 0xC) in TSTCON as shown in Figure 14-3.

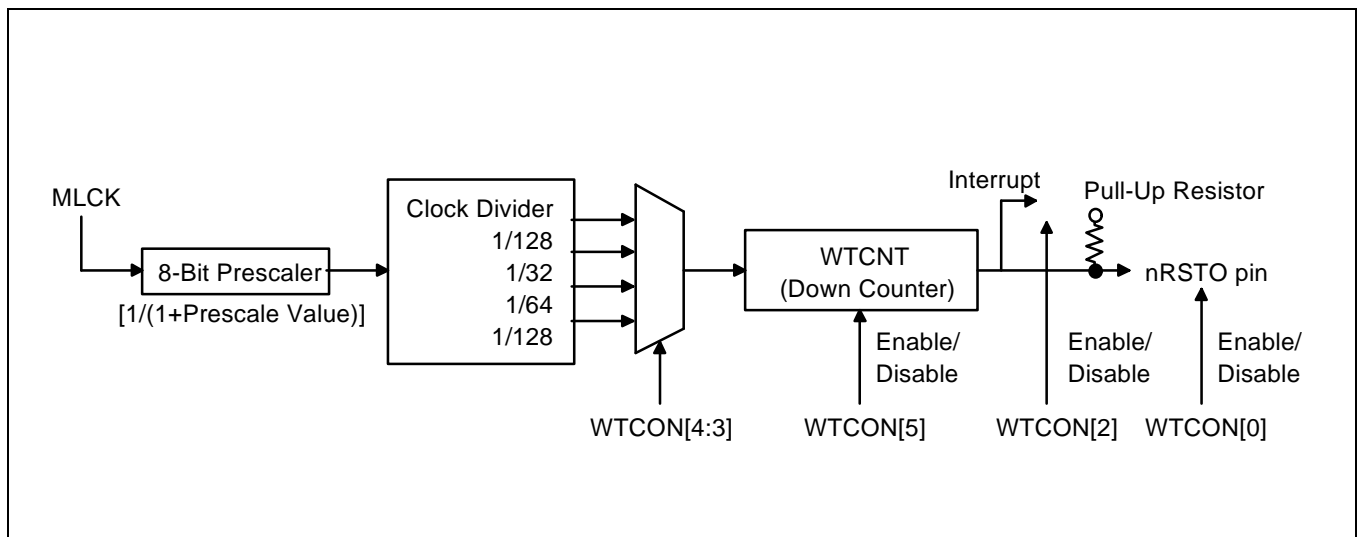


Figure 13-1. Watch-Dog Timer Block Diagram

WATCH-DOG TIMER COUNTER REGISTERS

The watch-dog timer counter register, WTCNT, is used to specify the time out duration. The watch-dog timer enable bit (bit 5, WTCNT) must be '0' before loading a value to this register.

Watch-dog timer clock = MCLK / (prescale value + 1) / division factor

Watch-dog timer duration = count_value x watch-dog timer clock period

Table 13-1. Watch-Dog Timer Counter Setting (MCLK=33 MHz)

Clock Source	Resolution	Maximum Interval	Remark
MCLK/(prescale+1)/16	6.30 μs	413 ms	Default setting
MCLK/(prescale+1)/32	12.6 μs	826 ms	–
MCLK/(prescale+1)/64	25.2 μs	1.651 s	–
MCLK/(prescale+1)/128	50.4 μs	3.305 s	–

NOTE: The value of prescaler is 0xc and WTCNT is 16-bit count.

Register	Offset Address	R/W	Description	Reset Value
WTCNT	0xf804	R/W	Watch-dog timer count register	0x00000003

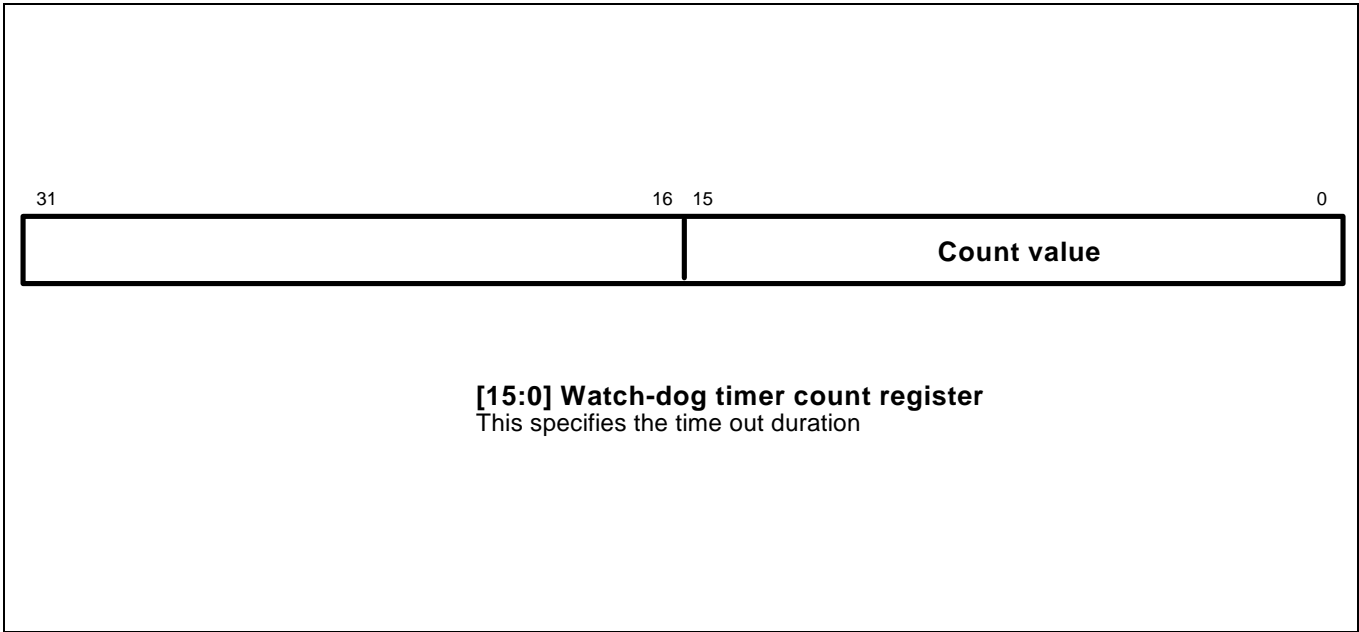


Figure 13-2. Watch-Dog Timer Count Register (WTCNT)

WATCH-DOG TIMER CONTROL REGISTER

Using the Watch-Dog Timer Control register, WTCN, you can enable/disable the watch-dog timer, select the clock signal from 4 different sources, enable/disable interrupts, and enable/disable the watch-dog timer reset output signal through the nRSTO pin. If the counter value of the watch-dog timer is “0,” the WTCN is cleared to 0x0.

Register	Offset Address	R/W	Description	Reset Value
WTCN	0xf800	R/W	Watch-dog timer control register	0x00000021

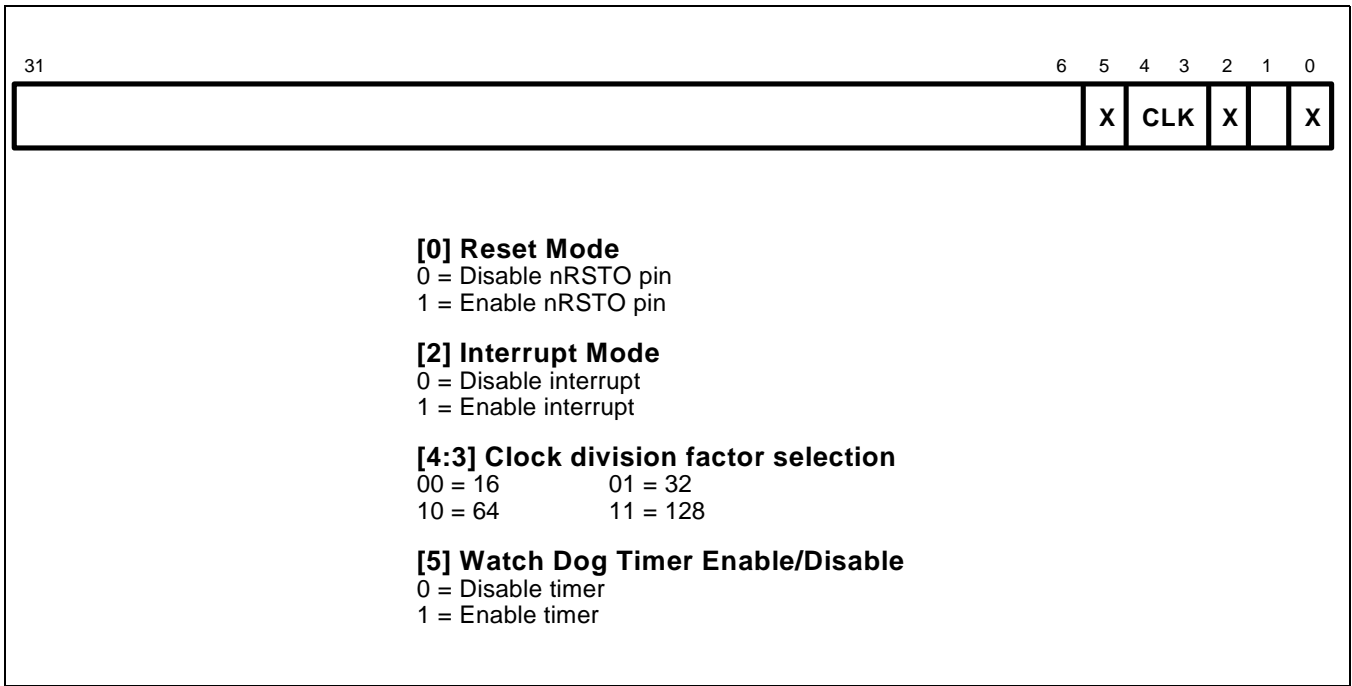


Figure 13-3. Watch-Dog Timer Control Register (WTCN)

Watch-Dog Timer Operation

Before loading a count value into the Watch-Dog Timer Count Register, WTCNT, you have to disable the watch-dog timer by setting the WTCN[5] bit to zero. When WTCN[5] bit set to "1," the watch-dog timer is enabled and the counter starts down-count. The value of the watch-dog counter register is accessible at any time while the watch-dog timer is enabled, because it provides read and write features.

The watch-dog timer provides general timer interrupt as well as system reset features. To enable the watch-dog timer interrupt, the WTCN[2] bit has to be set to "1". When the watch-dog timer interrupt is enabled, the interrupt signal generates one pulse of request signal to CPU. The interrupt pending bit (bit2, INTPNDR) is automatically set to '1' when an underflow occurs.

When WTCN[0] bit is '1', the nRSTO pin is enabled and watch-dog reset signal comes through the nRSTO pin. If watch-dog counter reaches to zero, for some reason, the nRSTO signal is activated during 128 MCLK cycles, and the WTCN will be automatically set to 0x0. To avoid watch-dog timer activating the nRSTO signal, the MCU has to reload the counter value into the watch-dog counter register (WTCNT) periodically.

The nRSTO signal is not connected to the nRESET internally. If nRSTO is connected to the nRESET by an external logic, the KS32C6200 initialization routine will be executed by the nRSTO signal.

NOTE

The pin type of nRST is open-drain output. If you want to use the nRST pin, a pull-up resistor must be installed on the nRST pins.

NOTES

14

I/O Ports

OVERVIEW

The KS32C6200 has 6 input, 13 output, 5 input/output, 10 extra-output ports and serial EERAM control ports. Some ports pins can be multiplexed with other internal device units, such as JTAG, UART, interrupt controller and so on.

I/O Port Special Registers

Two registers, IOPMOD and IOP, control the I/O port configuration. Table 14-1 shows the possible values for the port mode registers. The IOP register contains one bit for each port which reflects the signal level at the respective port pin.

Table 14-1. I/O Port Mode Configuration Settings

I/O Port Pin	I/O Port Mode Configuration Settings	
	Function for one	Function for zero
GIP[0]:RXD0	GIP[0]	RXD0
GIP[1]:RXD1	GIP[1]	RXD1
GIP[2]:nEXT_INT0	GIP[2]	nEXT_INT0
GIP[3]:nEXT_INT1	GIP[3]	nEXT_INT1
GIP[4]:nEXT_DREQ	GIP[4]	nEXT_DREQ
GIP[5]:UCLK	GIP[5]	UCLK
GOP[0]:TXD0	GOP[0]	TXD0
GOP[1]:TXD1	GOP[1]	TXD1
GOP[2]:nEXT_DACK	GOP[2]	nEXT_DACK
GOP[3]:TONE	GOP[3]	TONE
GOP[4]:nRSTO	GOP[4]	nRSTO
GOP[5]:nIOWR1	GOP[5]	nIOWR1
GOP[6]:nIOWR2	GOP[6]	nIOWR2
GOP[7]:nIORD1	GOP[7]	nIORD1
GOP[8]:nIORD2	GOP[8]	nIORD2
GOP[9]:CLKOUT	GOP[9]	CLKOUT
GOP[10]:Reserved	GOP[10]	Reserved
GOP[11]		—
GOP[12]		—

I/O PORT MODE REGISTER

The I/O port mode register, IOPMOD, is used to configure the GIP (general input port), the GOP (general output port), and the GIOP(general in/out port).

Register	Offset Address	R/W	Description	Reset Value
IOPMOD	0x4808	R/W	I/O port mode register	0x00000000

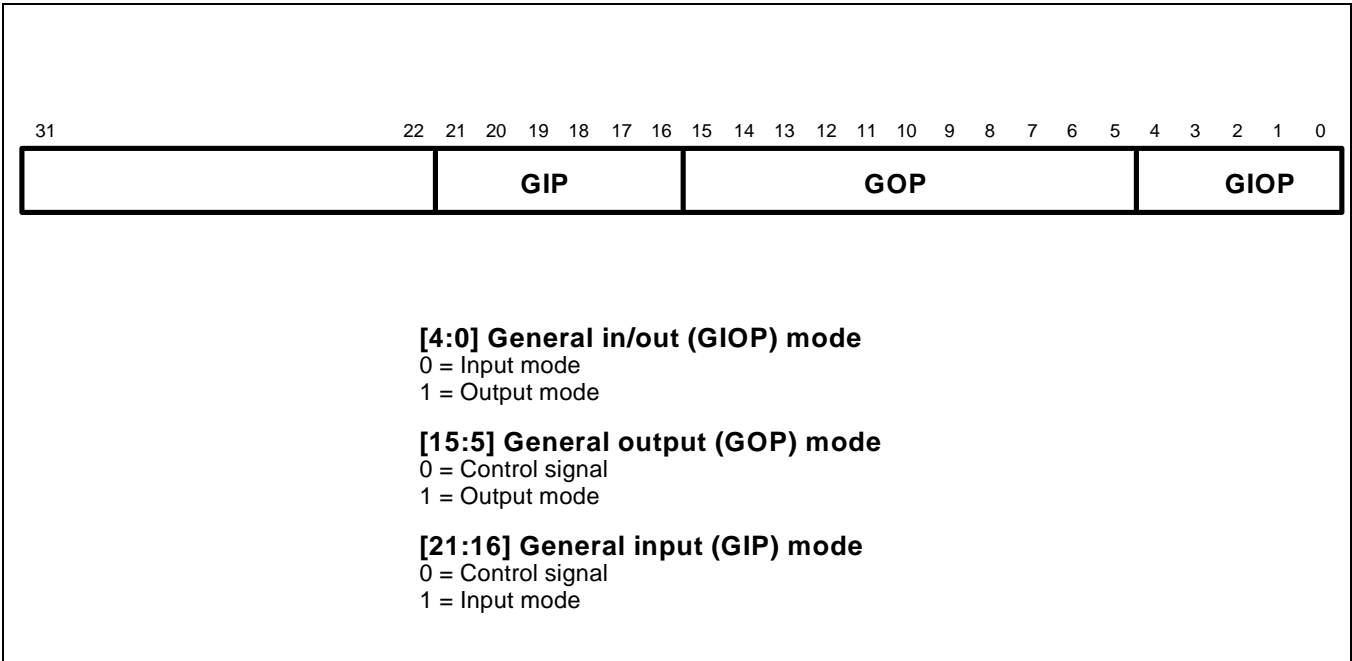


Figure 14-1. I/O Port Mode Register (IOPMOD)

I/O PORT REGISTER

he I/O port data register, IOPDATA, contains one-bit value for I/O ports that are configured to input mode and one-bit write value for ports that are in output mode. You can read/write the ports through the I/O port register, IOPDATA.

Register	Offset Address	R/W	Description	Reset Value
IOPDATA	0x4804	R/W	I/O port data register	Undefined

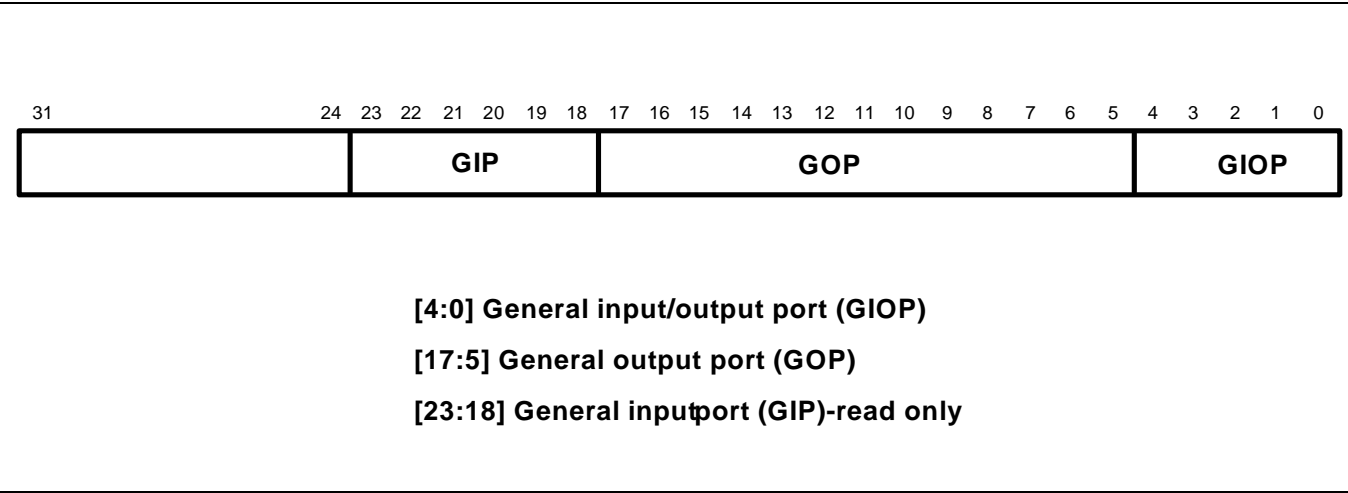


Figure 14-2. I/O Port Data Register (IOPDATA)

TEST CONTROL REGISTER

Test control register (TSTCON) contains 6-bits to test some functions of KS32C6200. These bits are used only for test at fabrication and are not specified in this manual. You can use the other bits as follows:

- CKOUT mode:
The CKOUT mode bit determines whether or not CKOUT output is divided by 2.
0 = MCLK / 2
1 = MCLK
- Prescaler value:
Timer 0, Timer 1, Timer 2, Watch-dog timer, and Tone generator use this prescaler value to divide MCLK.
- Bidirectional control pin

Register	Offset Address	R/W	Description	Reset Value
TSTCON	0x4800	R/W	Test control register	0x00000600

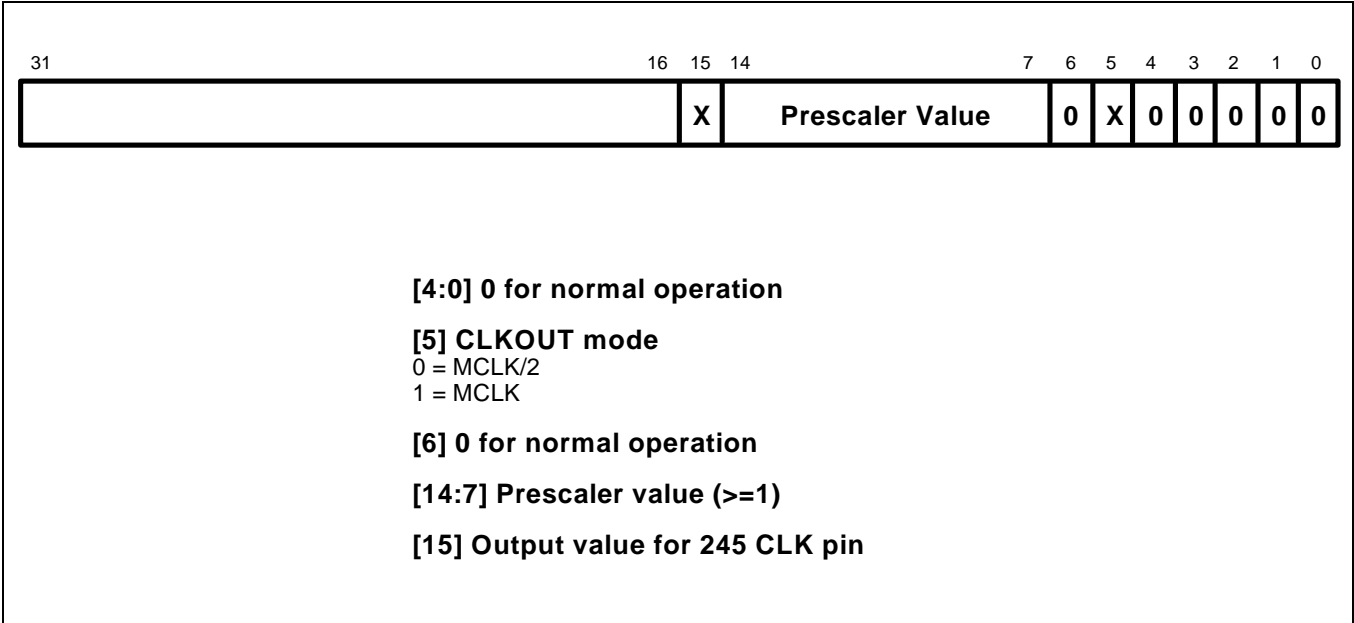


Figure 14-3. Test Control Register (TSTCON)

EERAM CONTROL REGISTER

EERAMCON controls EEDATA and EECLK pins to interface a serial EEPROM. EERAMCON[0] bit responds to the EEDATA pin and EERAMCON[1] bit to the EECLK pin.

Register	Offset Address	R/W	Description	Reset Value
EERAMCON	0x5000	R/W	EERAM control register	0x0000000x

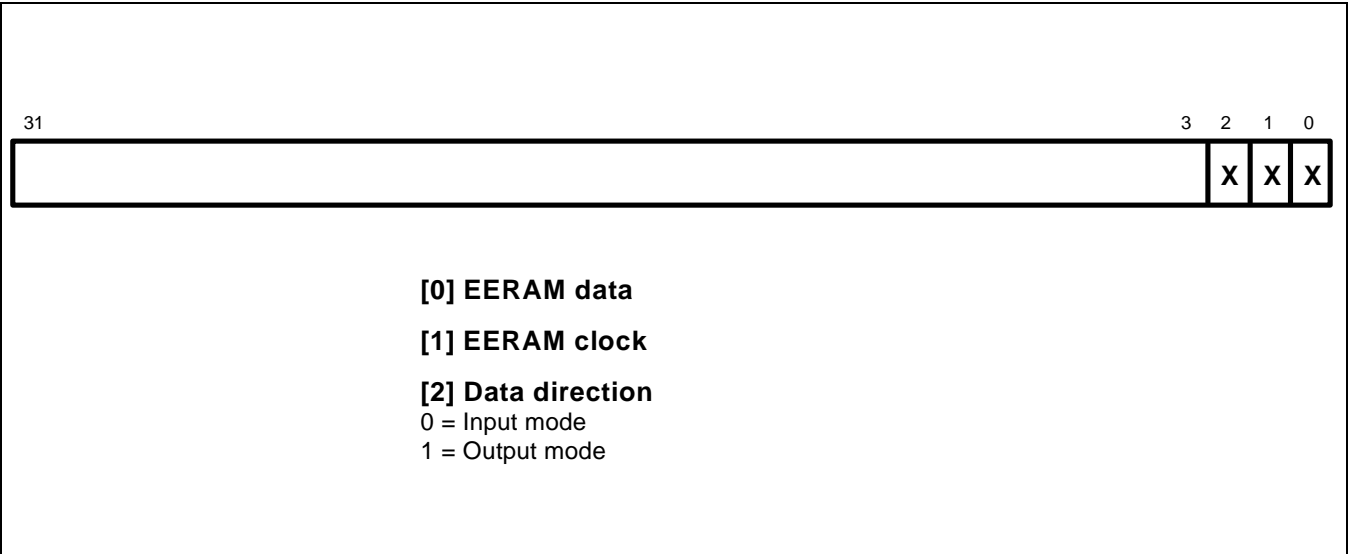


Figure 14-4. EERAM Control Register (EERAMCON)

JTAG Test-Logic Unit

32C6200 has three modes, core test mode, the logic test mode and the MDS mode. These modes are determined by TEST1 and TEST2 pins. MDS mode supports the JTAG test logic unit. When MCU is in the MDS mode, the GIOP pins is used as a TAP (test access port). The MDS mode will be used by the ICE (In-circuit Emulator) supporting the JTAG test logic unit. The core test mode and logic test mode are used only at fabrication.

Table 14-2. MCU Operating Mode Setting

Test2	Test1	MCU State
0	1	Normal operating mode
0	1	Core test mode (only fabrication)
1	0	Logic test mode (only fabrication)
1	1	MDS mode

Table 14-3. Test Access Port Pins (MDS Mode)

TAP Pin	Share Pin	Description
TCK	GIOP[0]	Test clock input
TMS	GIOP[1]	Test mode select input
TDI	GIOP[2]	Test data input
nTRST	GIOP[3]	Test reset input
TDO	GIOP[4]	Test data output

Extra-Output Port

KS32C6200 has 13 output ports and ten additional output pins for the function blocks, which have special functions only for Ink-Jet printers. These usages are somewhat different from GOP because the pins are not designed as a dedicated output port.

To use EOPA[5:0] as an output port:

- Write the data of EOPA[5:0] pins onto the EOPA register.
- Write 0x1800 onto the EOPL (Extra-Output Port Latch Register). EOPA[5:0] pins have had the valid data.
- Do not clear EOPL register after writing 0x1800.

To use EOPB[3:0] as output port:

- Write (EOPB[3:0] << 8) | 0x8000 onto the EOPB register. The bit 14 must be “0.”

EXTRA-OUTPUT PORT A REGISTER

The extra-output port A register, EOPA, contains one-bit write value to configure port to output mode.

Register	Offset Address	R/W	Description	Reset Value
EOPA	0x8004	R/W	Extra-output port A register	0x000003c0

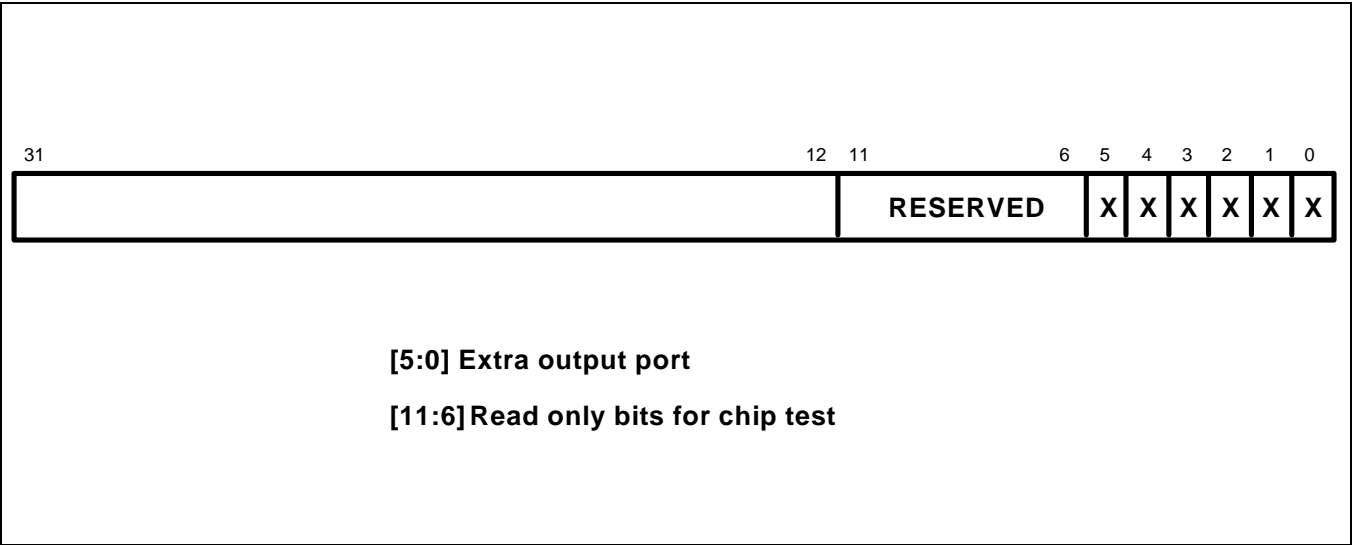


Figure 14-5. Extra-Output Port A Register (EOPA)

EXTRA-OUTPUT PORT LATCH REGISTER

The extra-output port latch register, EOPL, is written by “0x1800.”

Register	Offset Address	R/W	Description	Reset Value
EOPL	0x8000	R/W	Extra-output port latch register	0x800

EXTRA-OUTPUT PORT B REGISTER

The extra-output port B register, EOPB, contains 4-bit values of ports in output mode.

Register	Offset Address	R/W	Description	Reset Value
EOPB	0x9010	R/W	Extra-output port B register	0x0000cf0f

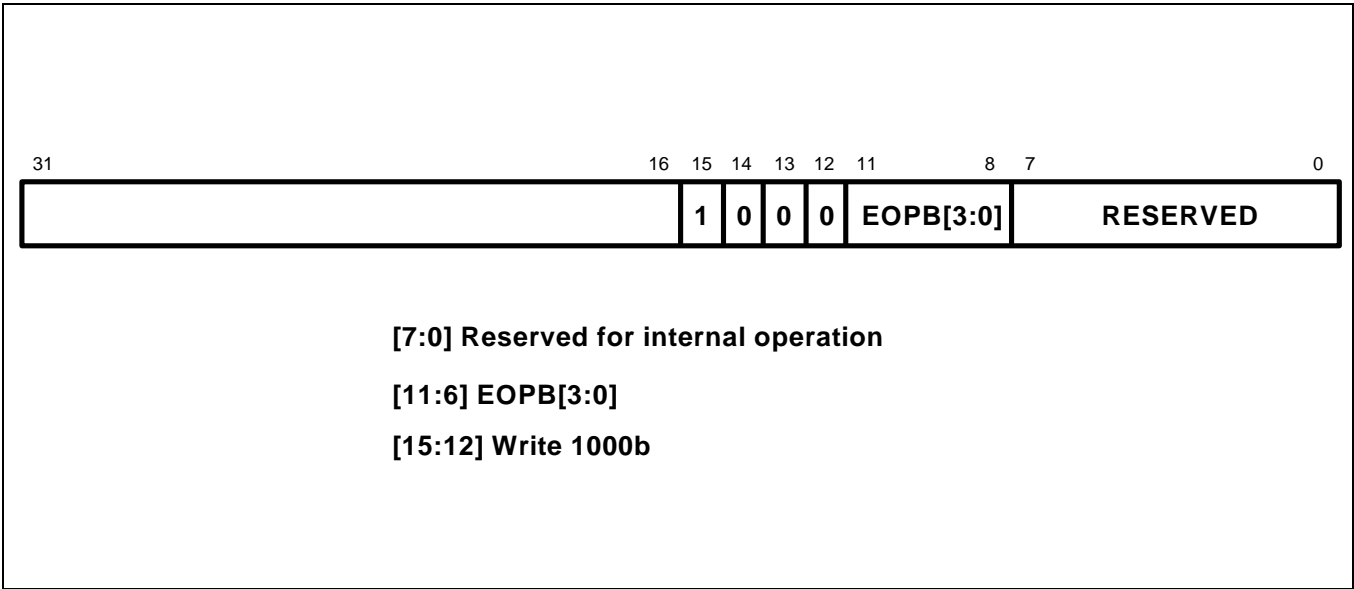


Figure 14-6. Extra-Output Port B Register (EOPB)

NOTES

15

Interrupt Controller

OVERVIEW

The KS32C6200 interrupt structure has a total of 15 interrupt sources. Interrupt requests can be generated by internal function blocks and at external pins. The ARM7T core recognizes two kinds of interrupts, the interrupt request (IRQ) and the fast interrupt request (FIQ). Therefore, all KS32C6200 interrupts can be categorized as either IRQ or FIQ. The KS32C6200 interrupt controller extends the number of multiple interrupt sources that can be serviced by using three special registers, INTMOD, INTPND, and INTMSK:

- **Interrupt mode register**
Defines the interrupt mode, IRQ or FIQ, for each interrupt source.
- **Interrupt pending register**
Indicates that an interrupt requests is pending (that is, when the I-flag or F-flag is set in the program status register, PSR). This status prevents any additional interrupts from being acknowledged. When a pending bit is set, the interrupt service routine starts whenever the I-flag or F-flag is cleared to '0'. The service routine must clear the pending condition by writing a '1' to the appropriate pending bit.
- **Interrupt mask register**
Indicates that the current interrupt has been disabled if the corresponding mask bit is '0'. If an interrupt mask bit is '1', the interrupt will be serviced normally, and if the global mask bit (bit 20) is cleared, all interrupts are not serviced. However, the source's pending bit is set when the interrupt is generated even if the corresponding mask bit is '0'. After the global mask bit is set, the interrupt will be serviced.

Interrupt Sources

In the KS32C6200, the 15 interrupt sources are described as follows:

- [14] Parallel port interrupt
- [13] Timer 2 interrupt
- [12] Timer 1 interrupt
- [11] Timer 0 interrupt
- [10] DMA 0 interrupt
- [9] DMA 1 interrupt
- [8] UART channel 0 error interrupt
- [7] UART channel 1 error interrupt
- [6] UART channel 0 receive interrupt
- [5] UART channel 1 receive interrupt
- [4] UART channel 0 transmit interrupt
- [3] UART channel 1 transmit interrupt
- [2] Watch-dog timer interrupt
- [1] External interrupt 1
- [0] External interrupt 0

INTERRUPT MODE REGISTER

Bits in the interrupt mode register, INTMOD, specify if an interrupt is to be serviced as a fast or normal interrupt.

Register	Offset Address	R/W	Description	Reset Value
INTMOD	0x4000	R/W	Interrupt mode register	0x00000000

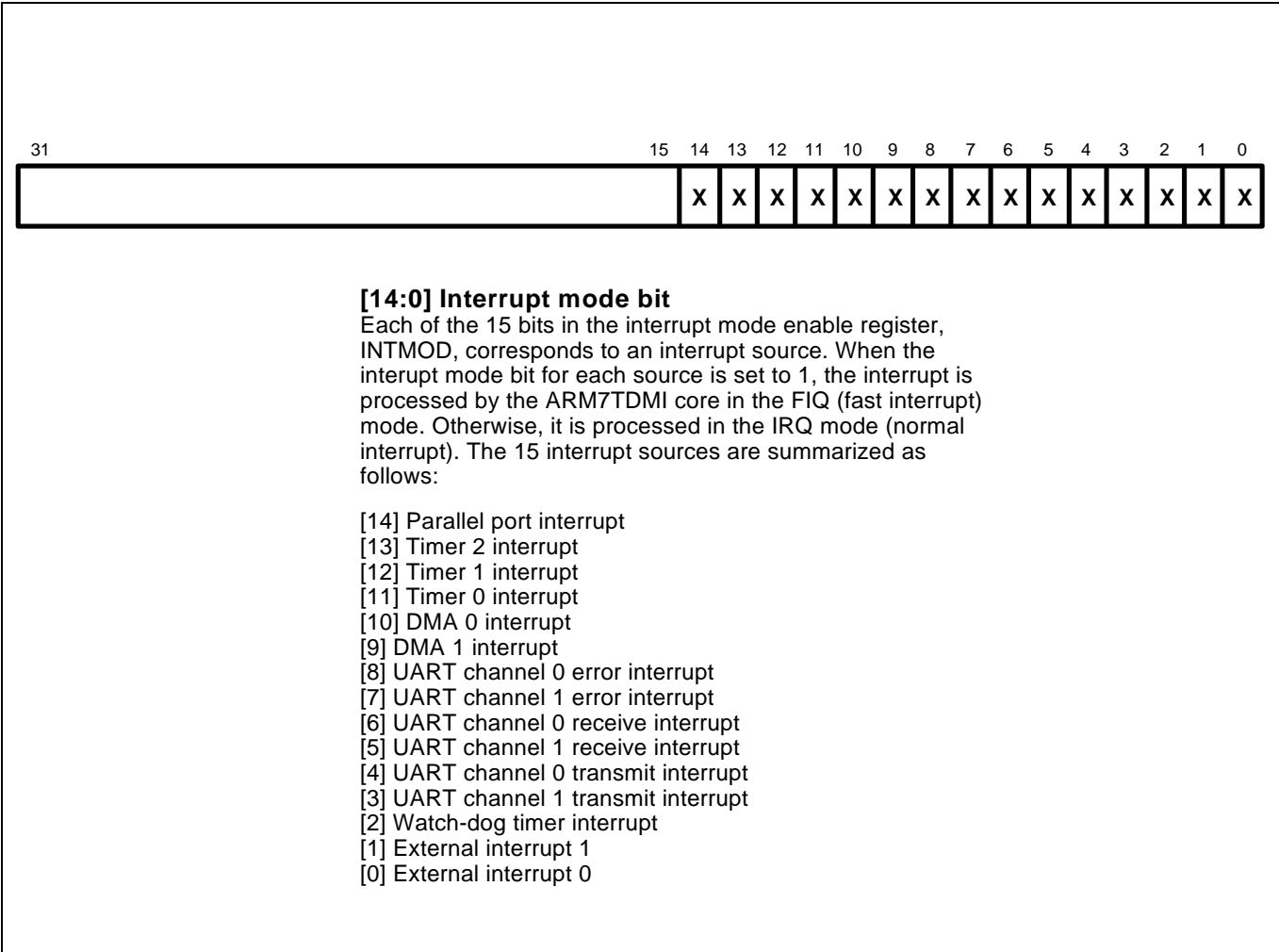


Figure 15-1. Interrupt Mode Register (INTMOD)

INTERRUPT MASK INTERRUPT

The interrupt mask register, INTMSK, contains interrupt mask bits for each interrupt source.

Register	Offset Address	R/W	Description	Reset Value
INTMSK	0x4008	R/W	Interrupt mask register	0x00000000

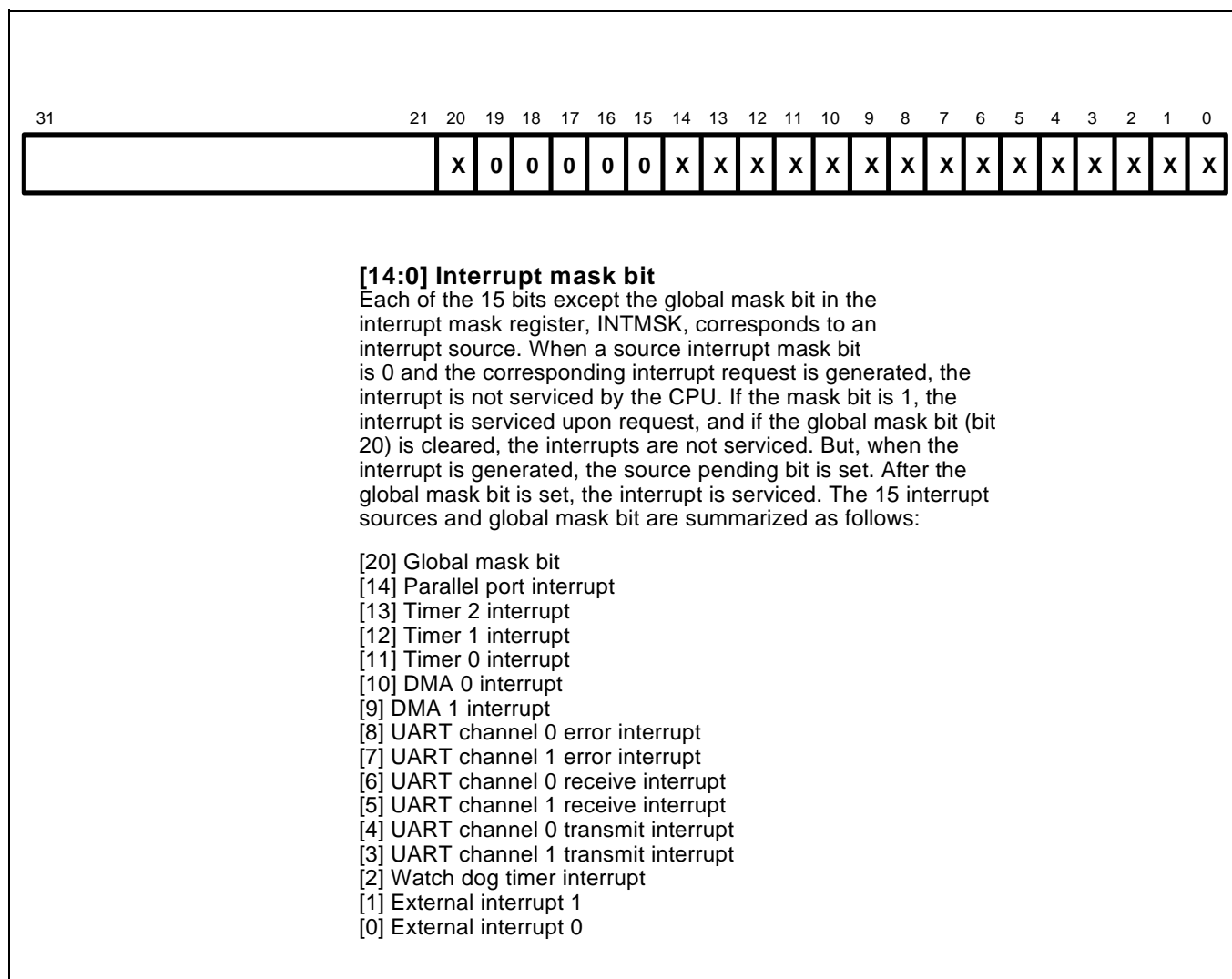


Figure 15-3. Interrupt Mask Register (INTMSK)

Recognitions of the Interrupt Requests

The external interrupt pins (nEINT1, nEINT2) recognize interrupt requests by edge trigger. The KS32C6200 detects the down edge of nEINT1, nEINT2 and checks that the interrupt request pins are low levels during two MCLK period to confirm the interrupt request valid. The low level interval of nEINT1,2 must be longer than 3 MCLK period to confirm the validity of the interrupt request.

16

Electrical Data

ABSOLUTE MAXIMUM RATINGS

Table 16-1. Absolute Maximum Ratings

(T_A = 25 °C)

Parameter	Symbol	Rating	Unit
Supply Voltage	V _{DD}	− 0.3 to + 7.0	V
Input Voltage	V _{IN}	− 0.3 to V _{DD} + 0.3	V
Operating Temperature	T _A	0 to + 70	°C
Storage Temperature	T _{STG}	− 40 to + 125	°C

THERMAL CHARACTERISTICS

Table 16-2. Thermal Characteristics

(T_A = 25 °C)

Parameter	Symbol	Value	Unit
Thermal Impedance—Junction to Ambient Plastic 160-pin TQFP	θ _{JA}	40	°C/W

D.C. ELECTRICAL CHARACTERISTICS

Table 16-3. D.C. Electrical Characteristics

(T_A = 0 °C to +70 °C, V_{DD} = 4.75 V to 5.25 V)

Parameter	Symbol	Conditions	Min	Max	Unit
Input High Voltage	V _{IH}	TTL interface	2.0	–	V
		TTL schmitt trigger	–	2.1	
Input Low Voltage	V _{IL}	TTL interface	–	0.8	V
		TTL schmitt trigger	0.8	–	
Input High Current	I _{IH}	Input buffer, V _{IN} = V _{DD}	– 10	10	μA
Input Low Current	I _{IL}	Input buffer, V _{IN} = V _{SS}	– 10	10	μA
	I _{LL2}	Input buffer with pull-up, V _{IN} = V _{SS}	– 200	– 10	
Output High Voltage	V _{OH}	Type 4, I _{OH} = – 4 mA ⁽²⁾	2.4	–	V
		Type 8, I _{OH} = – 8 mA ⁽²⁾			
Output Low Voltage	V _{OL}	Type 4, I _{OL} = 4 mA ⁽²⁾	–	0.4	V
		Type 8, I _{OL} = 8 mA ⁽²⁾			
Quiescent Supply Current	I _{DD}	V _{IN} = V _{SS} or V _{DD} , 33 MHz	–	180	mA

NOTES

1. We recommend for you to install the bypass capacitors between the V_{DD} and the V_{SS} of KS32C6200.
The bypass capacitor increases the noise immunity of KS32C6200.
2. The pin type of O₁, O₂, O₃, I/O₁, I/O₄, and I/O₅ is type 4, and I/O₃ is type 8.

Table 16-4. A.C. Electrical Characteristics

(T_A = 0 °C to +70 °C, V_{DD} = 4.75 V to 5.25 V)

Parameter	Symbol	Min	Max	Unit
RESET Pulse Width	tRST	69	—	MCLK
CLKOUT Rising Time from MCLK	tCKOUTR	5.0	15.4	ns
CLKOUT Falling Time from MCLK	tCKOUTF	4.8	14.5	ns
nEINT1,2 Setup before MCLK	tINTS	0	—	ns
nEINT1,2 Hold after MCLK	tINTH	5.0	—	ns
nEINT1,2 Pulse Width	tINTW	3.0	—	MCLK
Parallel Port Input Hold after MCLK	tPINH	0	27	ns
Parallel Port Output Valid from MCLK	tPOV	3	17	ns
PPD[7:0] Valid from MCLK	tPDV	11	23	ns
PPD[7:0] High Impedance from MCLK	tPDZ	4	13	ns
nXDREQ Setup Time before MCLK	tXDREQS	6	—	ns
nXDREQ Hold Time before MCLK	tXDREQH	4	—	ns
nXDREQ Pulse Width	tXDREQW	2	—	MCLK
nXDACK Valid from MCLK	tXDACK	4	29	ns
nXDACK Pulse Width	tXDACKW	18	—	MCLK
Address Hold Time	tADDRH	7.1	—	ns
Address Delay time	tADDRD	—	25.1	ns
ROM Bank Chip Select Delay Time	tNRCS	—	20.6	ns
ROM/SRAM/Extra I/O Bank Out Enable Delay	tNROE	—	23.5	ns
SRAM/Extra I/O Bank Write Enable Delay	tNRWE	—	18.2	ns
SRAM/Extra I/O Bank Write Byte Enable Delay	tNWBE	—	18.1	ns
Read Data Hold Time	tRDH	3.0	—	ns
Write Data Delay Time (SRAM/Extra I/O)	tWDD	—	9.8	ns
Write Data Hold Time (SRAM/Extra I/O)	tWDH	26.3	—	ns
DRAM Row Address Strobe Active Delay	tNRASF	—	15.2	ns
DRAM Row Address Strobe Release Delay	tNCASR	—	27.0	ns
DRAM Column Address Strobe Active Delay	tNCASF	—	16.1	ns
DRAM CAS Signal Release Delay Time	tNCASR	—	17.1	ns
DRAM CAS Write Active Delay	tNCASW	—	19.8	ns
DRAM Bank Write Enable Delay Time	tNDWE	—	24.4	ns
DRAM Bank Out Enable Delay Time	tNDOE	—	23.5	ns
External I/O Bank Chip Select Delay Time	tNECS	—	20.6	ns
Special I/O Bank Read Signal Delay Time	tNIORD	—	23.5	ns
Speical I/O Bank Write Signal Delay Time	tNIOWR	—	18.2	ns
DRAM Write Data Delay Time (DRAM)	tWDDD	—	14.2	ns
DRAM Write Data Hold Time (DRAM)	tWDDH	7.4	—	ns

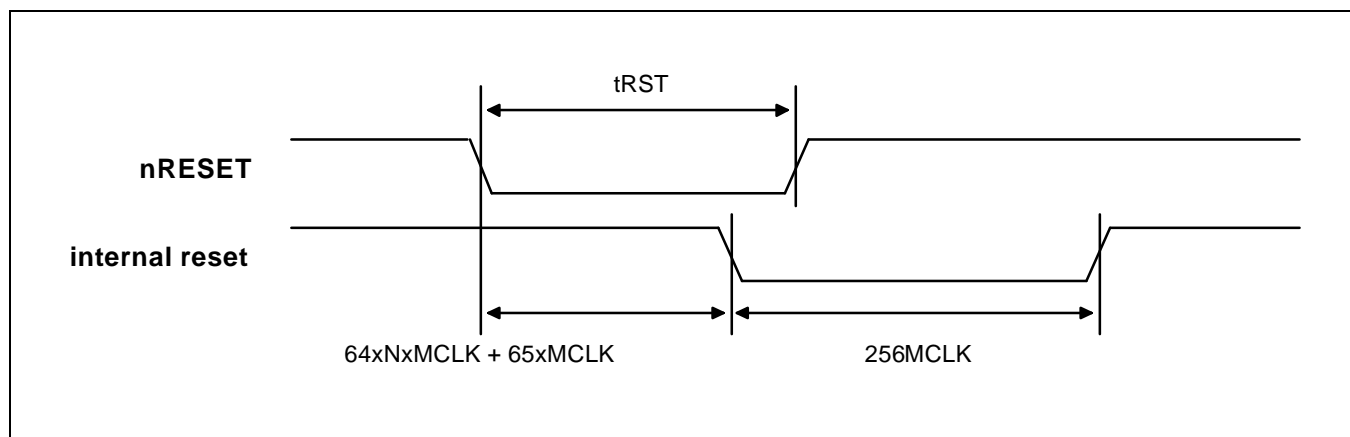


Figure 16-1. Reset Cycles (CLKSET = 0)

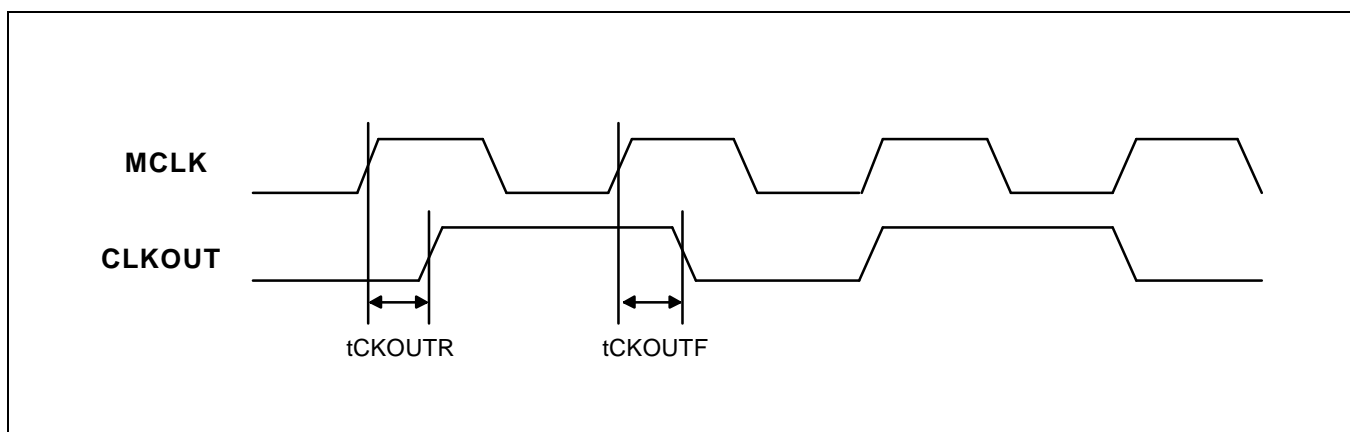


Figure 16-2. CLKOUT Cycle (CLKSEL = 0)

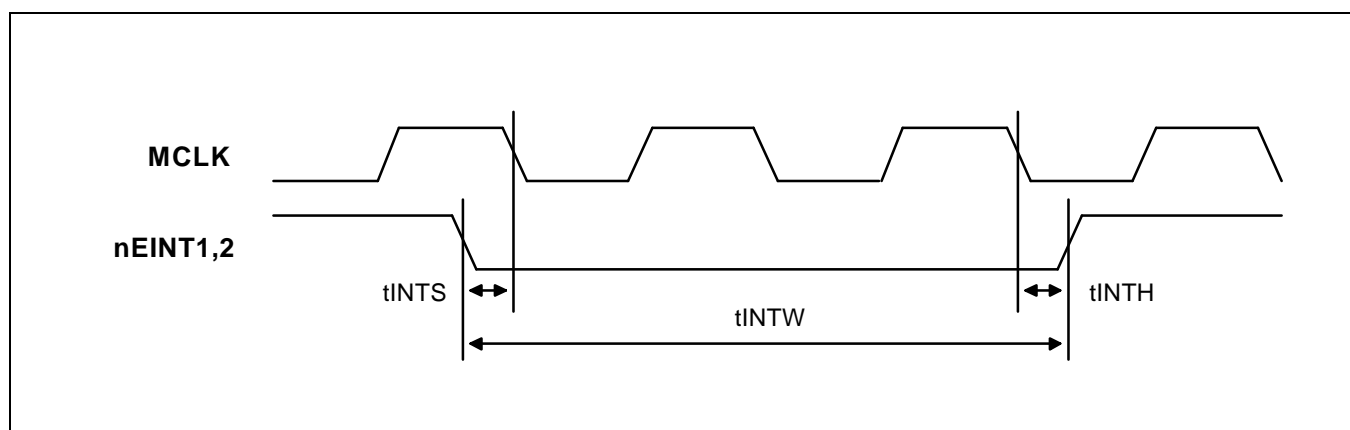


Figure 16-3. External Interrupt Cycle (CLKSEL = 0)

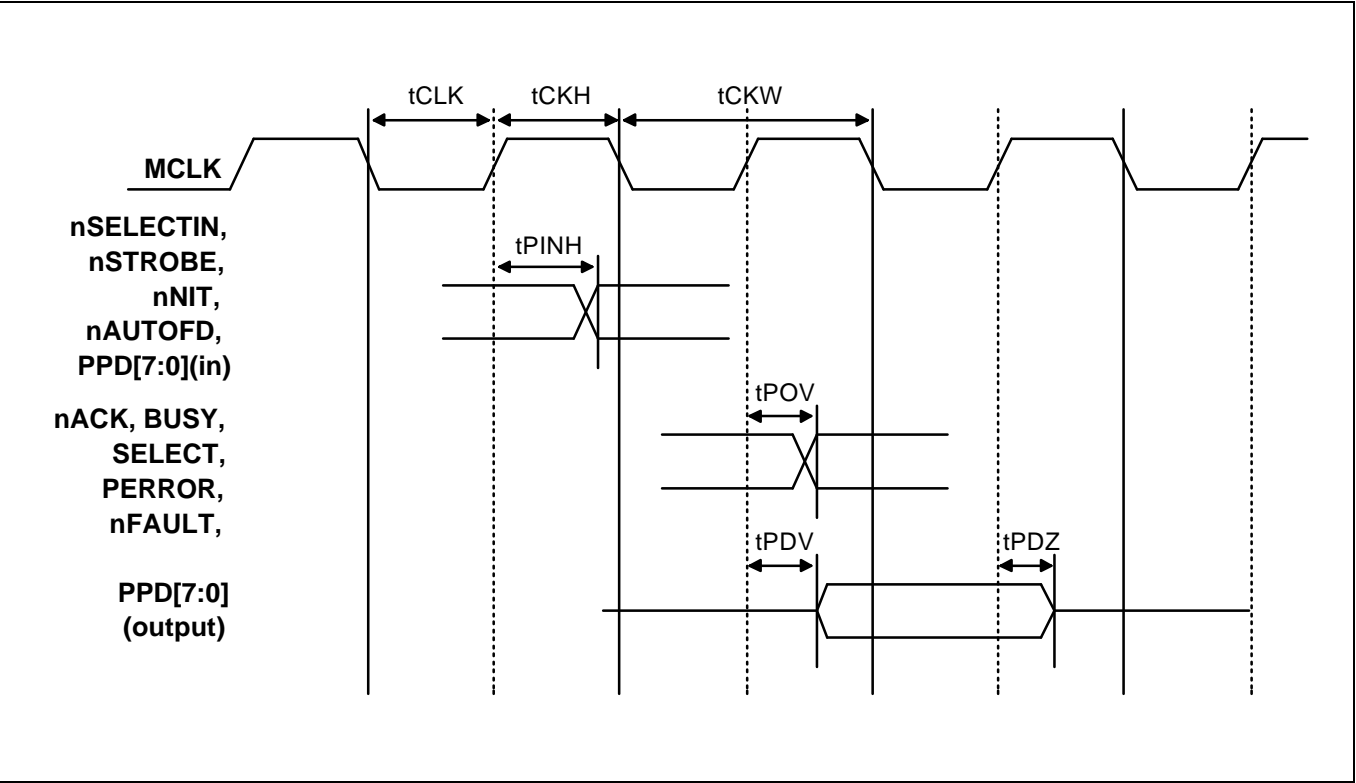


Figure 16-4. Parallel Port Interface Cycle (CLKSEL = 0)

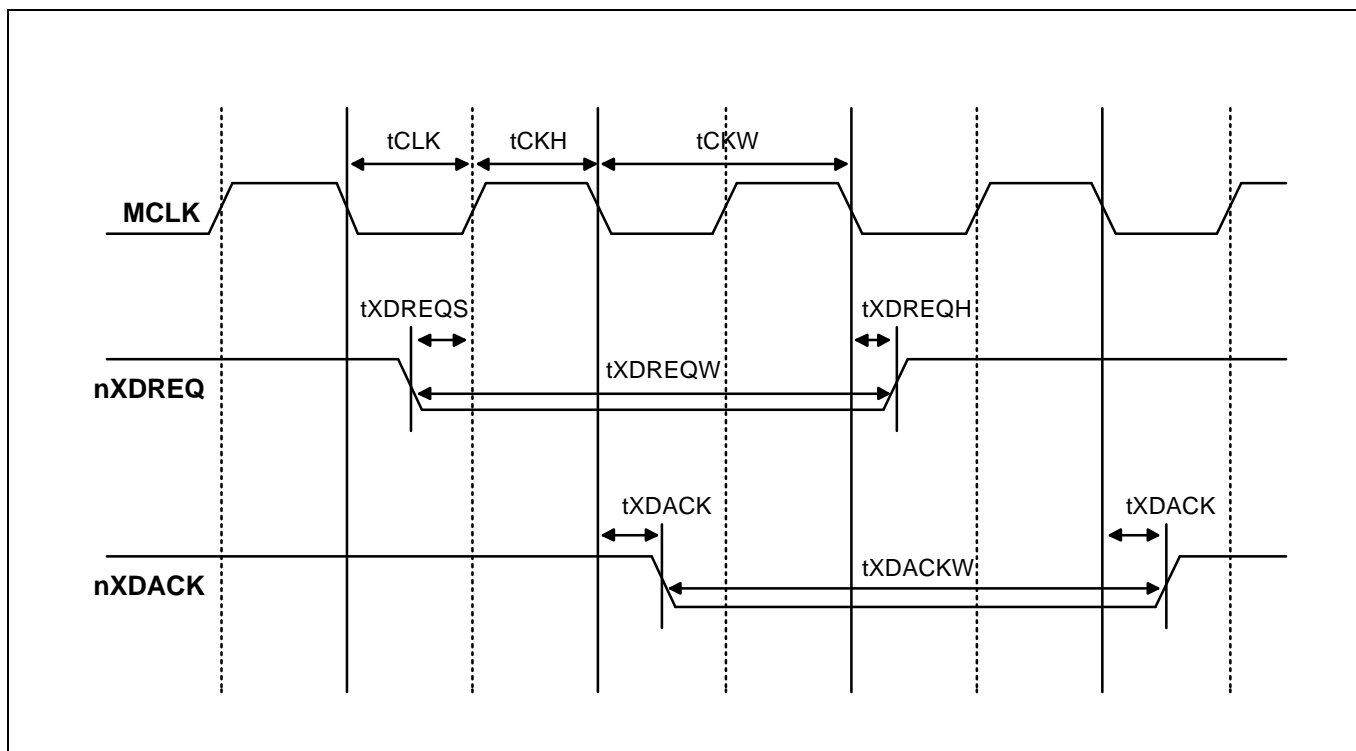


Figure 16-5. External DMA Cycle (CLKSEL = 0)

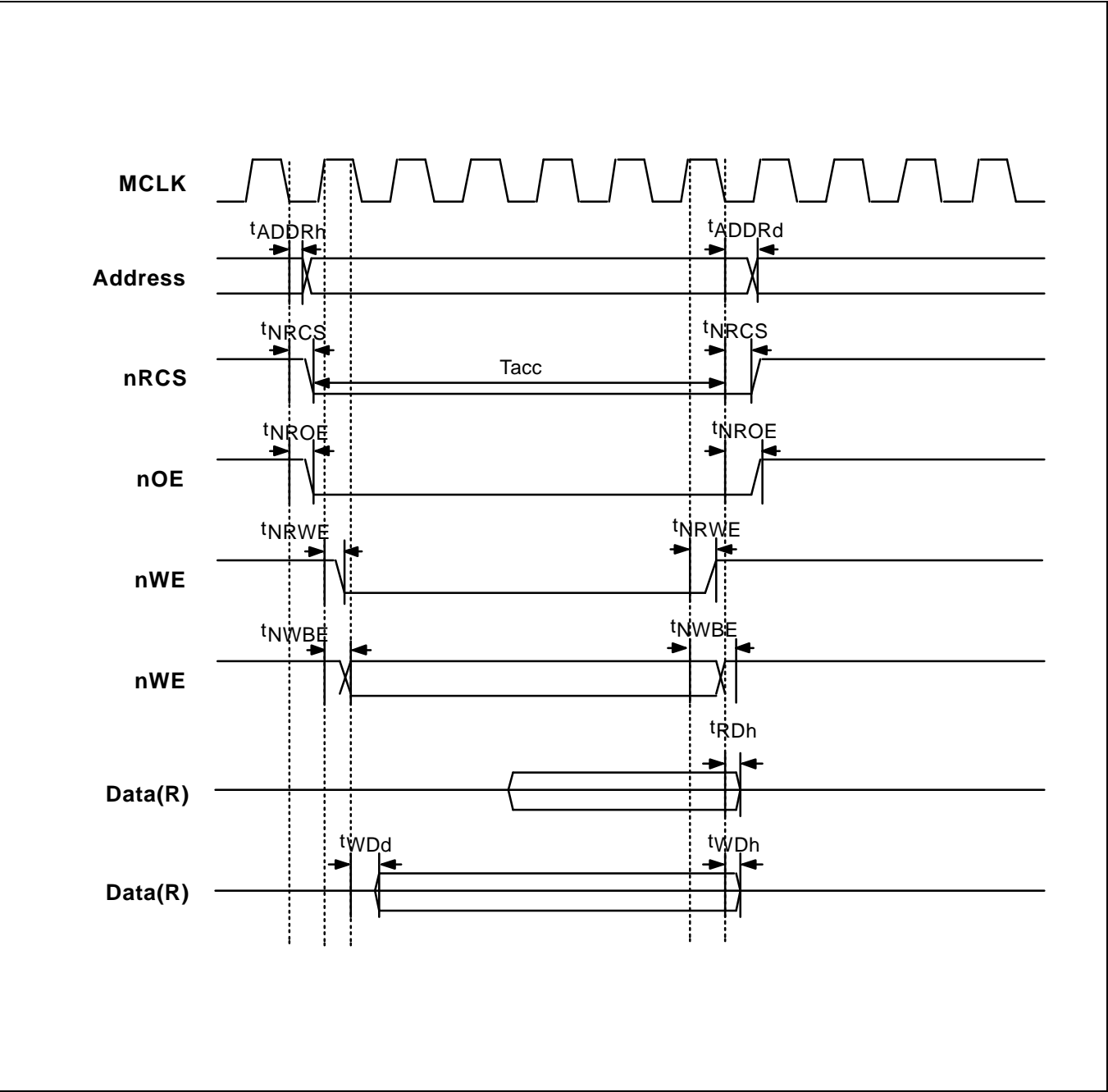


Figure 16-6. ROM/DRAM Access Timing

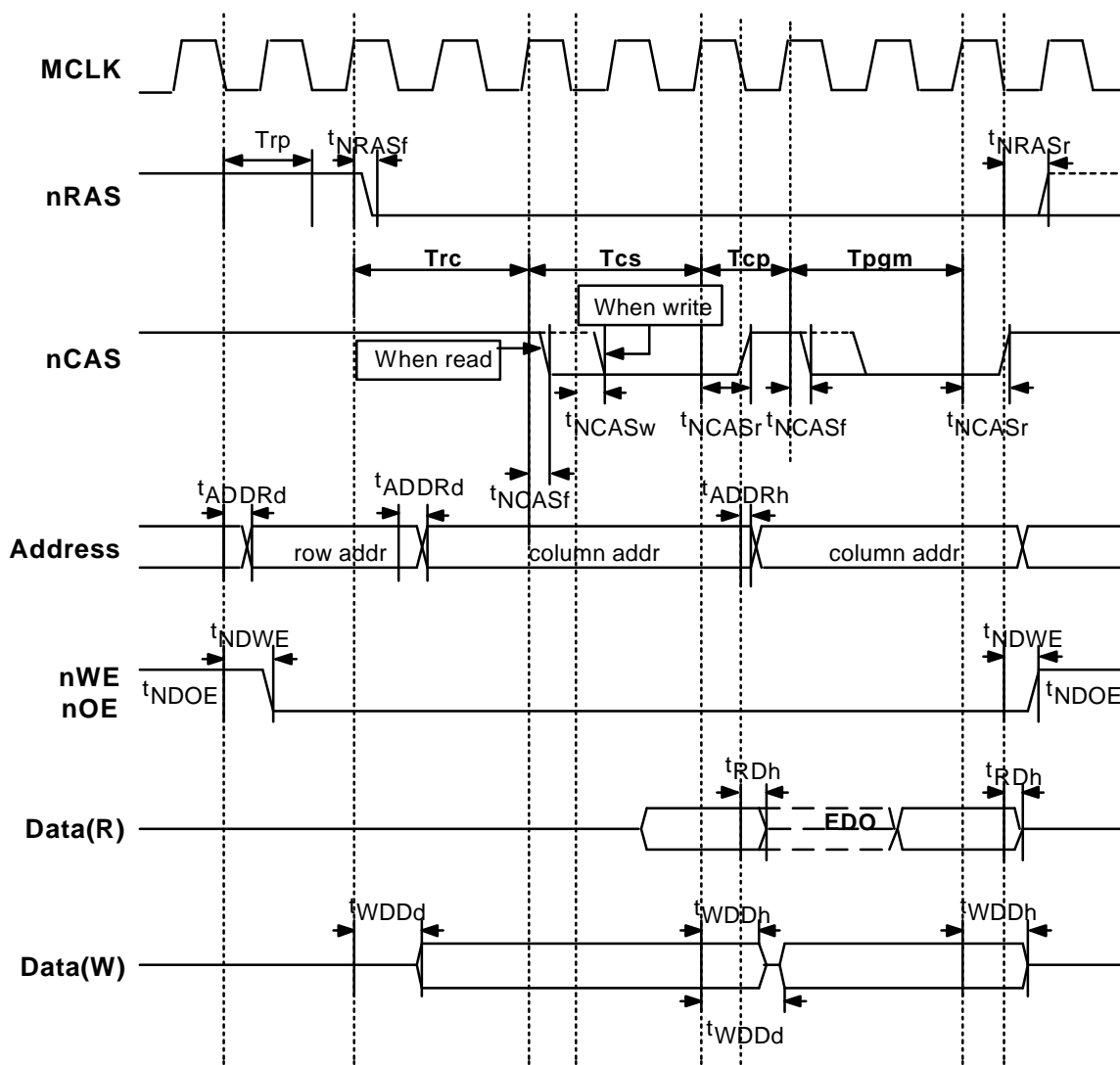


Figure 16-7. DRAM Bank Access Timing

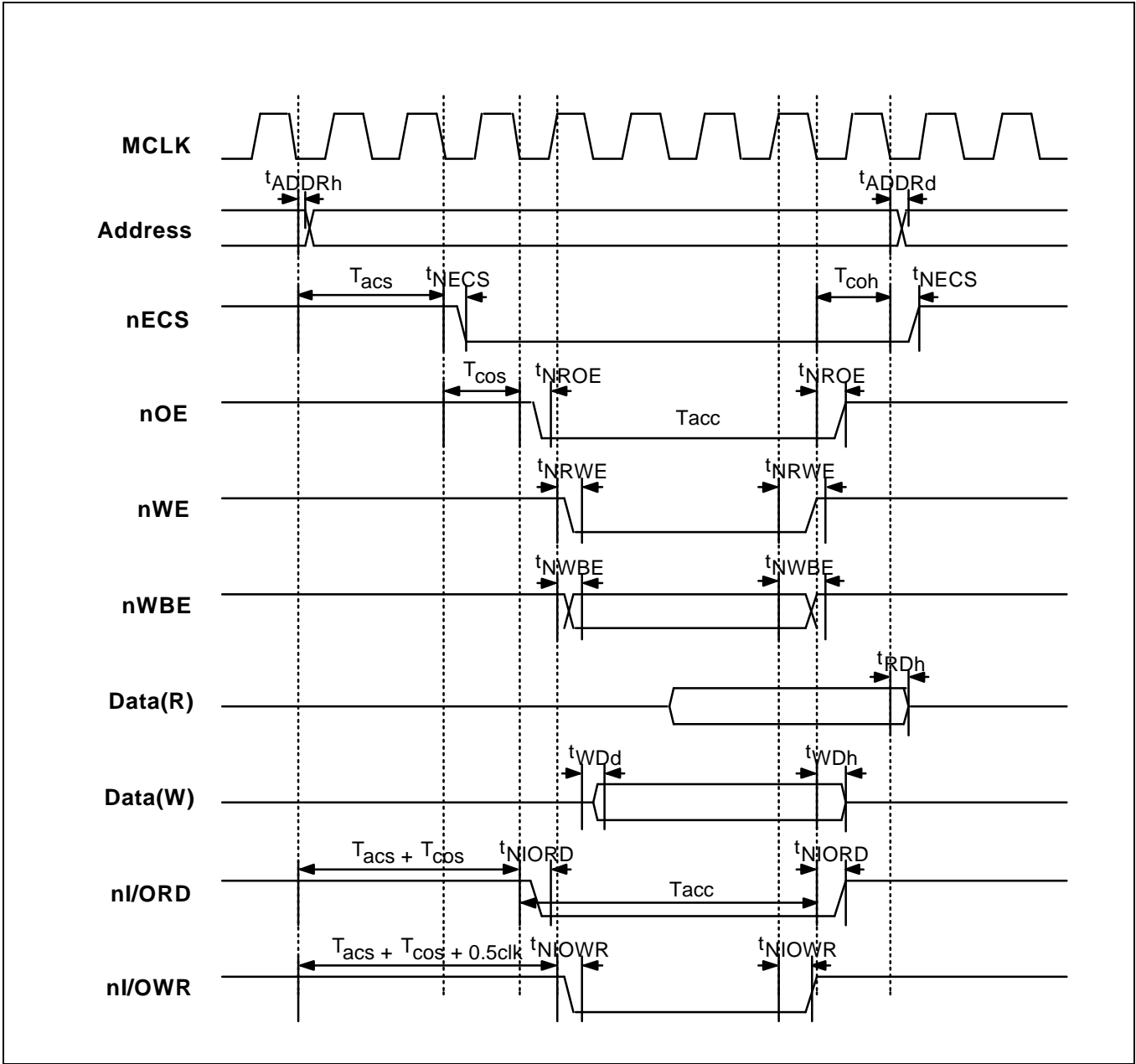


Figure 16-8. Extra-Bank Access Timing

NOTES

17

Mechanical Data

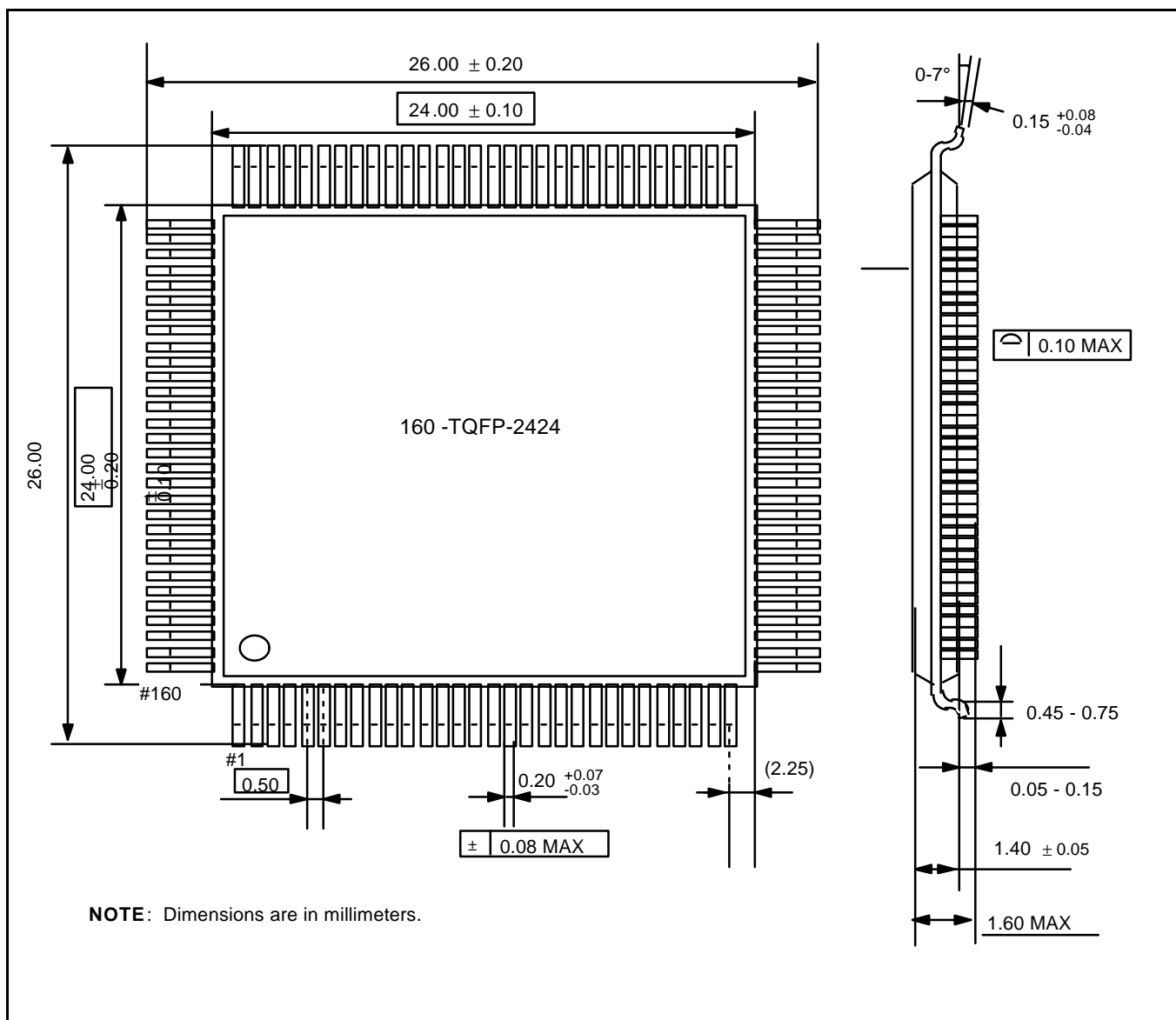


Figure 17-1. 160-TQFP Package Dimensions

KS32C6200 User's Manual Rev. 1.21 ERRATA

Last Revision Date: 9.OCT.1998

It's absolutely recommended to read this sheet.

Caution When Embedded ICE(through JTAG Port) is used and SFR is viewed by debugger, SFR contents may be changed. When the program is download using the ICE, the cache is not updated. So, you have to flush the cache before downloading by clearing the tag RAM or by a reset..

Caution SWP instruction has to manipulate the data in non-cacheable memory area. If SWP instruction manipulates the cachable memory area, the data in cache will be differ the data in memory.

Error Page 1-4,1-5,14-5: 245CLK => 245DIR

Error Page 1-12: TBCNT2 , 581ch => 580ch

Error Page 4-11,Figure 4-8:nRCS0,nRCS1

Because nRCS0 has to be ROM bank, SRAM doesn't use RCS0.
nRCS0 => nRCS1, nRCS1 => nRCS2

Error Page 4-25: Figure 4-21

The waveform of nOE is not correct. It is right that the nOE is 'H' level.

Error Page 9-1: Figure 9-1

GDMA, DMA Chnnel 0 => DMA0, DMA Channel 0

CDMA, DMA Chnnel 0 => DMA1, DMA Channel 1

Error Page 10-7: nACK control

=> Setting this bit to "1" forces the external nACK output to **Low** level by force. This is generally done when hardware handshaking is disable. When this bit is '0',the external nACK is the the internal signal.

Error Page 10-8: SELECT output control bit

;paper error has occurred => ;printer is ready

Caution Page 10-11: [9] DMA request enable

Just after PPCON9 is set, the PPIC will do DMA request for data transfers. So, the DMA has to be configured before PPIC is configured.

Caution If the parallel port event is occurred, the PPIC asserts an interrupt request to the interrupt controller. The PPIC then inactivates the interrupt request automatically though the corresponding bit of PPINTPND is not cleared to '0', Therefore, if the bit 14 of INTPND is cleared but the PPINTPND is not cleared. the interrupt controller retracts the interrupt request to CPU and the interrupt is lost. It's recommended the following sequence to process parallel port interrupts.

1. Read PPINTPND and save the value to memory.
2. Turn off the bit 14 of INTPND
3. Check again PPINTPND. if the PPINTPND is different, go to 1.
4. Turn off the corresponding bit of PPINTPND by the saved value.
5. Process the parallel port interrupt by the saved PPINTPND.

Error Page 11-9:

Register	Offset Address	R/W	Description	Reset Value
UCON0	0xe004	R/W	UART channel 0 control register	0x00
UCON1	0xe804	R/W	UART channel 1 control register	0x00

Error Page 11-11,Figure 11-7: "[6]Transmit holding register empty" item

correctings: transmit holding register => transimt buffer register

Caution Page 13-1,Figure 13-1: nRSTO is open drain output. The external pull-up register have to be installed.

Error Page 14-7: Table 14-2

0 1 Normal operating mode => 0 0 Normal operating mode

Error Page 15-3: Table 15-2

The figure 15-3 is not a right figure. Replace the figure 15-3 as follows:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00		
																	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

[14:0] Interrupt pending bit

Each of the 15 bits in the interrupt pending register, INTPND, corresponds to an interrupt source. When an interrupt request is generated, it will be set by '1'. The interrupt service routine must then clear the pending condition by writing '1' to the appropriate pending bit. Only the bit written with '1' toggles from '1' to '0'. The 15 interrupt sources are summarized as follows:

[14] Parallel port interrupt

[0] External interrupt 0

Error Page 16-2: Table 16-3

Output Low Voltage: Type 8: $I_{OH}=8mA \Rightarrow I_{OL}=8mA$

Error Page 16-7: Figure 16-6 6th wave

nWE => nWBE[1:0], 2nd Drata(R) => Data(W)

Etc

o Table 1-5,1-6,1-7

Table 1.1 KS32C6000 Signal Descriptions. => Table 1.1 KS32C6200 Signal Descriptions.

o Page 1-13, Table 1-3, 16th row, EOP => EOPA

o Page 3-22, 13th line, 0x0000001 => 0x00000014

o Page 3-58, 19th line: Overflow Eetection => Overflow Detection

o Page 4-10, 17th line: Figure 4-20 => Figure 4-15

o Page 4-12, 4th line: 01003010h => 0100301ch

o Page 4-12, the last line: Figure 4-9 => Figure 4-8

o Page 4-21, Figure 4-16: Tocs => Tcos

o Page 4-27, Figure 4-23: nOE => nWE

o Page 7-1, 3rd line: 224 => 208

o Page 9-11, Figure 9-8: CDMA. => DMA.

o Page 10-8, Figure 10-5, The bit10 is ommitted in figure by mistake.

o Page 10-13, [7] Abort bit, bys => bus

o Page 11-1, 9th line: transmit holding register => transmit buffer register

o Page 13-1, Figure 13-1 : 1/128 => 1/16

o Page 14-4, 2nd line : he => The

o Page 10-6, Figure 10-4, 4th line : PPN[7:0] => PPD[7:0]