1 PRODUCT OVERVIEW

OVERVIEW

The S3CB519/FB519 single-chip CMOS microcontroller is designed for high performance using Samsung's new 8-bit CPU core, CalmRISC.

CalmRISC is an 8-bit low power RISC microcontroller. Its basic architecture follows Harvard style, that is, it has separate program memory and data memory. Both instruction and data can be fetched simultaneously without causing a stall, using separate paths for memory access. Represented below is the top block diagram of the CalmRISC microcontroller.

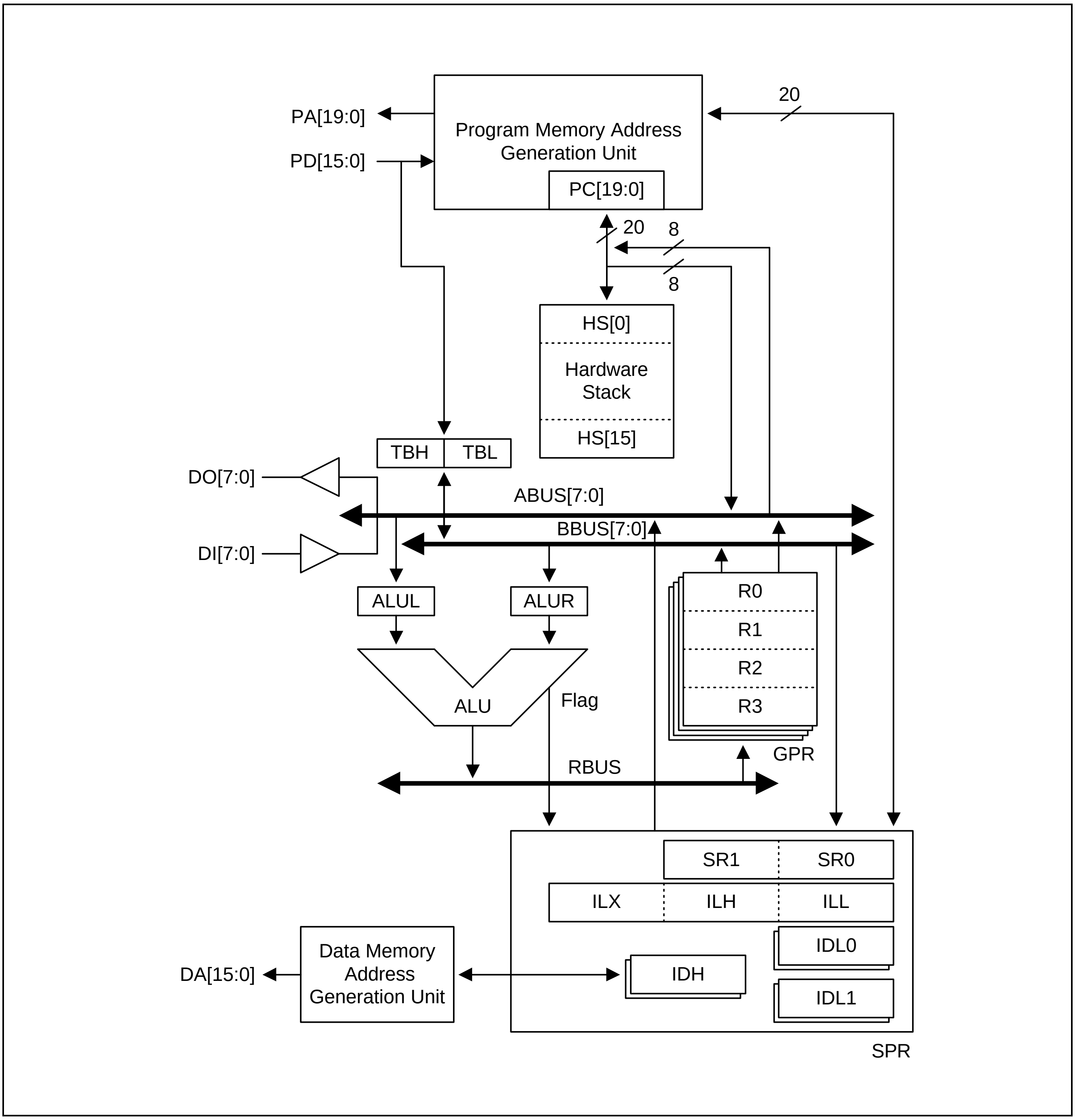


Figure 1-1. Top Block Diagram

The CalmRISC building blocks consist of:

— An 8-bit ALU

— 16 general purpose registers (GPR)

— 11 special purpose registers (SPR)

— 16-level hardware stack

— Program memory address generation unit

— Data memory address generation unit

Sixteen GPRs are grouped into four banks (Bank0 to Bank3), and each bank has four 8-bit registers (R0, R1, R2, and R3). SPRs, designed for special purposes, include status registers, link registers for branch-link instructions, and data memory index registers. The data memory address generation unit provides the data memory address (denoted as *DA[15:0]* in the top block diagram) for a data memory access instruction. Data memory contents are accessed through *DI[7:0]* for read operations and *DO[7:0]* for write operations. The program memory address generation unit contains a program counter, PC[19:0], and supplies the program memory address through *PA[19:0]* and fetches the corresponding instruction through *PD[15:0]* as the result of the program memory access. CalmRISC has a 16-level hardware stack for low power stack operations as well as a temporary storage area.

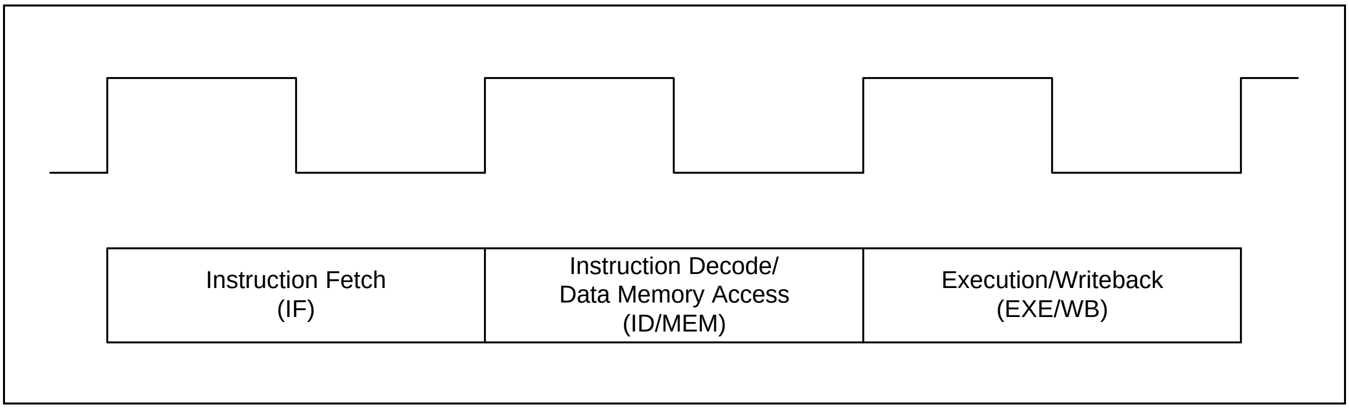


Figure 1-2. CalmRISC Pipeline Diagram

CalmRISC has a 3-stage pipeline as described below:

As can be seen in the pipeline scheme, CalmRISC adopts a register-memory instruction set. In other words, data memory where *R* is a GPR can be one operand of an ALU instruction as shown below:

The first stage (or cycle) is the Instruction fetch stage (IF for short), where the instruction pointed by the program counter, PC[19:0] , is read into the Instruction Register (IR for short). The second stage is the Instruction Decode and Data Memory Access stage (ID/MEM for short), where the fetched instruction (stored in IR) is decoded and data memory access is performed, if necessary. The final stage is the Execute and Write-back stage (EXE/WB), where the required ALU operation is executed and the result is written back into the destination registers.

Since CalmRISC instructions are pipelined, the next instruction fetch is not postponed until the current instruction is completely finished but is performed immediately after completing the current instruction fetch. The pipeline stream of instructions is illustrated in the following diagram.

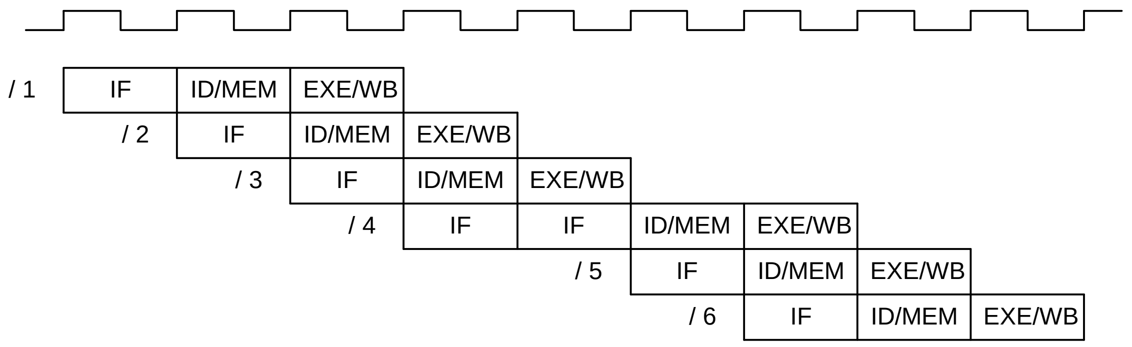


Figure 1-3. CalmRISC Pipeline Stream Diagram

Most CalmRISC instructions are 1-word instructions, while same branch instructions such as long “call” and “jp” instructions are 2-word instructions. In Figure 1-3, the instruction, *I4*, is a long branch instruction, and it takes two clock cycles to fetch the instruction. As indicated in the pipeline stream, the number of clocks per instruction (CPI) is 1 except for long branches, which take 2 clock cycles per instruction.

FEATURES

CPU

- 8-bit CalmRISC

Coprocessor

- MAC 816
- 8 × 16, 16 × 16 multiply and accumulation
- Arithmetic operation

Memory

- ROM: 16K-word
- RAM: 3K-byte
  2048 (X-memory)
  1024 (Y-memory)

I/O Pins

- 11 I/O: not include COM/SEG
- 35 I/O: include COM/SEG

Power-Down

- Idle mode: only CPU clock stops
- Stop mode: main system oscillator stops
- Sub-system clock stop mode

ROM Option

- Basic timer counter clock source selection reset value
- Watchdog timer enable/disable selection

8-Bit Basic Timer

- Programmable interval timer
- 8 kinds of clock source

Watchdog Timer

- System reset

Watch Timer

- Real time clock or interval time measurement
- Buzzer function (0.5/1/2/4 kHz at 4.19 MHz OSC)

Timer/Counters

- One 8-bit timer with PWM/Capture
- One 16-bit general-purpose timer/counter

LCD Controller/Driver

- 56 SEG × 16 COM terminals
- 8, 12 and 16 COM selectable
- 16-level contrast control
- Key strobe output function

Battery Level Detector

- 2.4, 2.7, 3.0, 3.3, 4.0, 4.5 V detectable
- Internal level and/or external level selectable

8-Bit Serial I/O Interface

- 8-bit transmit/receive mode
- 8-bit receive mode
- LSB-first or MSB-first transmission selectable

A/D Converter

- Sigma delta ADC
- Linear 14-bit data (16-bit format)
- 256X over sampling
- Operation voltage: V_{DD} = 3.0 V–5.5 V

D/A Converter

- 8-bit resolution
- Regulated output voltage
- Operation voltage: V_{DD} = 2.4 V–5.5 V

Oscillation Sources

- Crystal, ceramic, RC for main system clock
- Crystal or external oscillator for subsystem clock
- Main system clock frequency: Max 8.2 MHz
- Subsystem clock frequency: 32.768 kHz

Operating Voltage

- 2.2 V to 5.5 V

Operating Temperature Range

- – 40 °C to 85 °C

Package Type

- 100 QFP-1420C

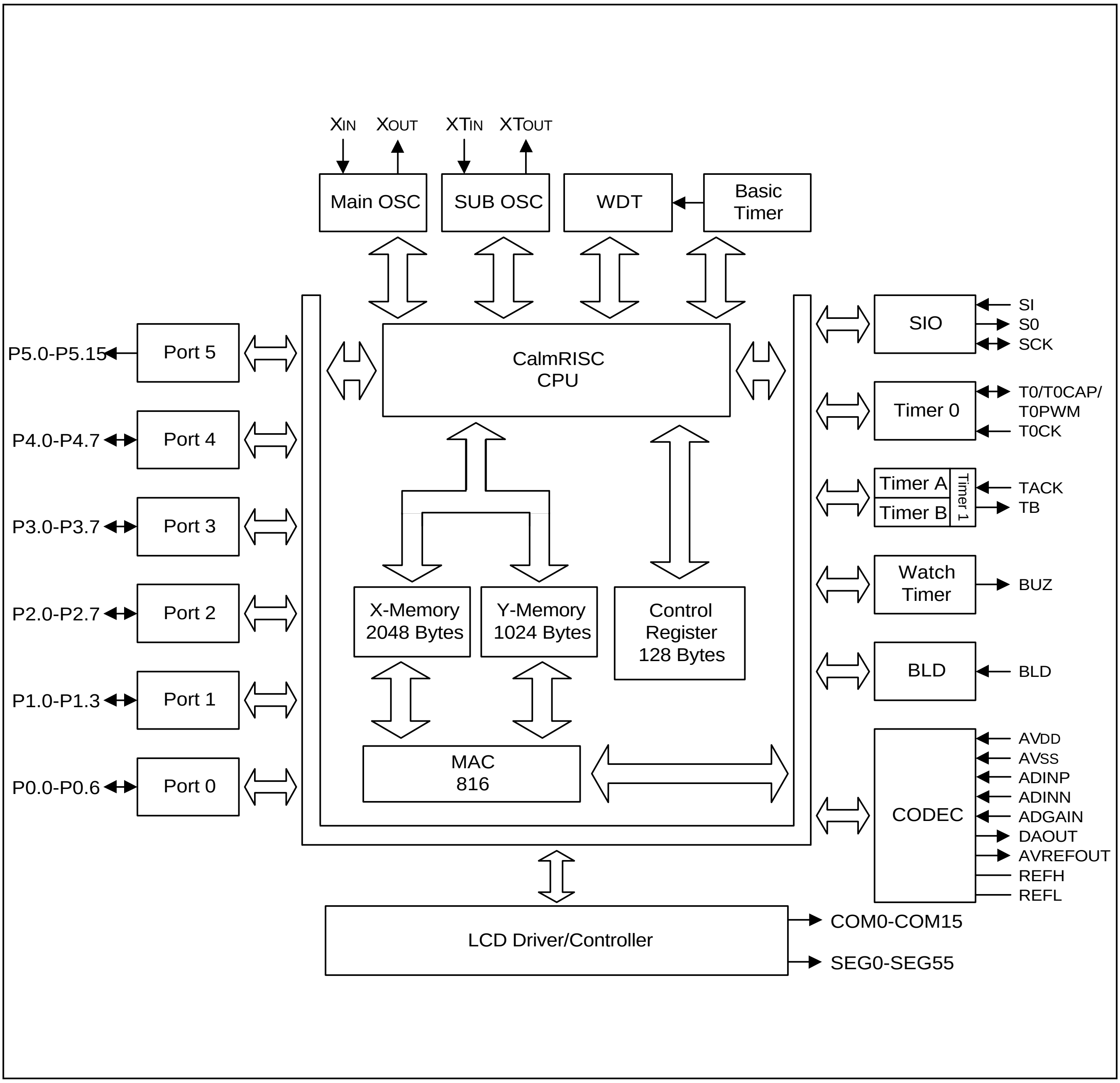


Figure 1-4. S3CB519/FB519 Block Diagram

PIN ASSIGNMENT

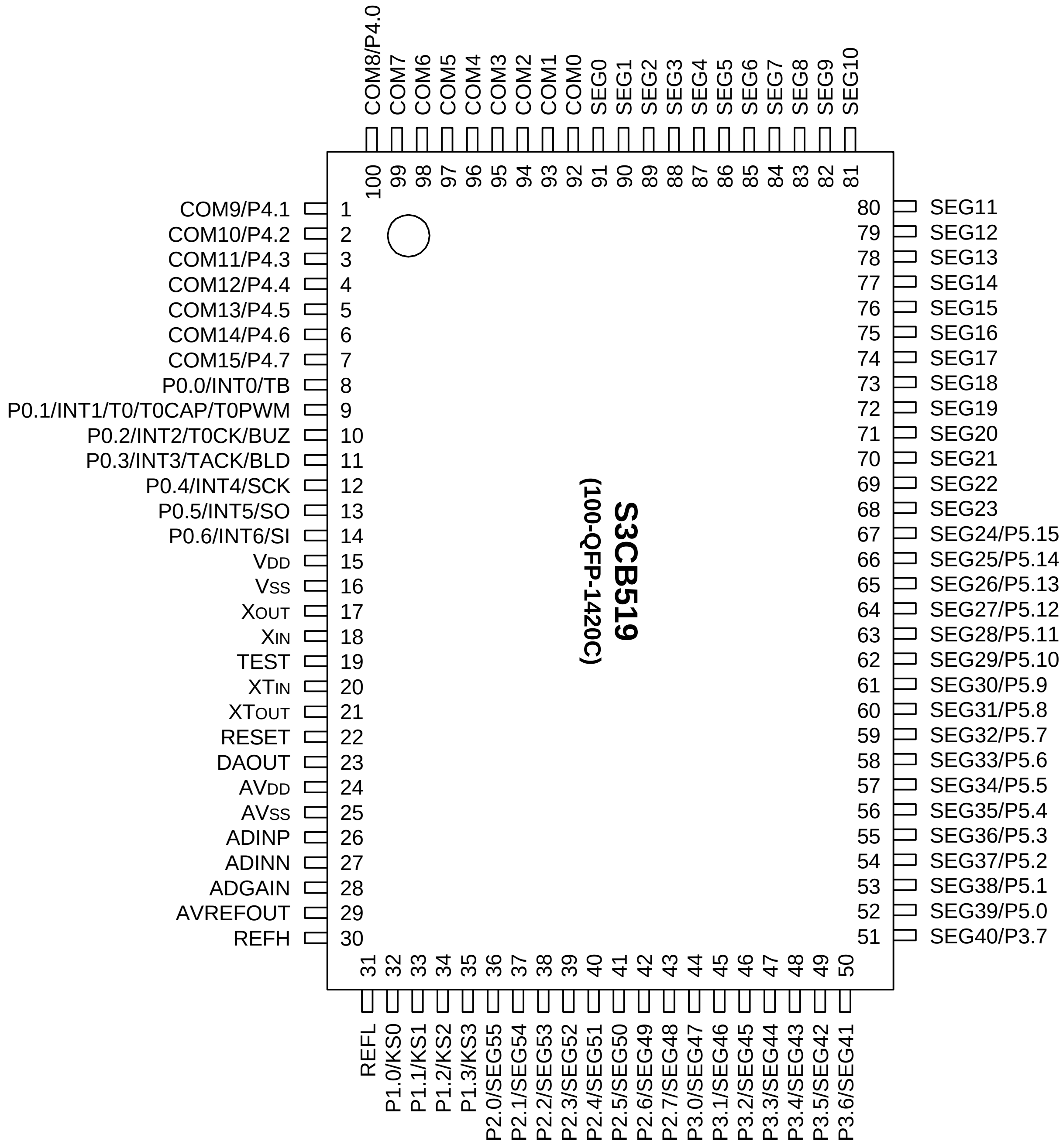


Figure 1-5. S3CB519 Pin Assignment Diagram (100-QFP)

2 ADDRESS SPACE

OVERVIEW

CalmRISC has 20-bit program address lines, *PA[19:0]*, which support up to 1 Mwords of program memory. The 1 Mword program memory space is divided into 256 pages, and each page is 4 Kwords long as shown on the next page. The upper 8 bits of the program counter, PC[19:12], points to a specific page, and the lower 12 bits, PC[11:0], specify the offset address of the page.

CalmRISC also has 16-bit data memory address lines, DA[15:0], which support up to 64K-byte of data memory. The 64K-byte data memory space is divided into 256 pages, and each page has 256 bytes. The upper 8 bits of the data address, DA[15:8], points to a specific page, and the lower 8 bits, DA[7:0], specify the offset address of the page.

PROGRAM MEMORY (ROM)

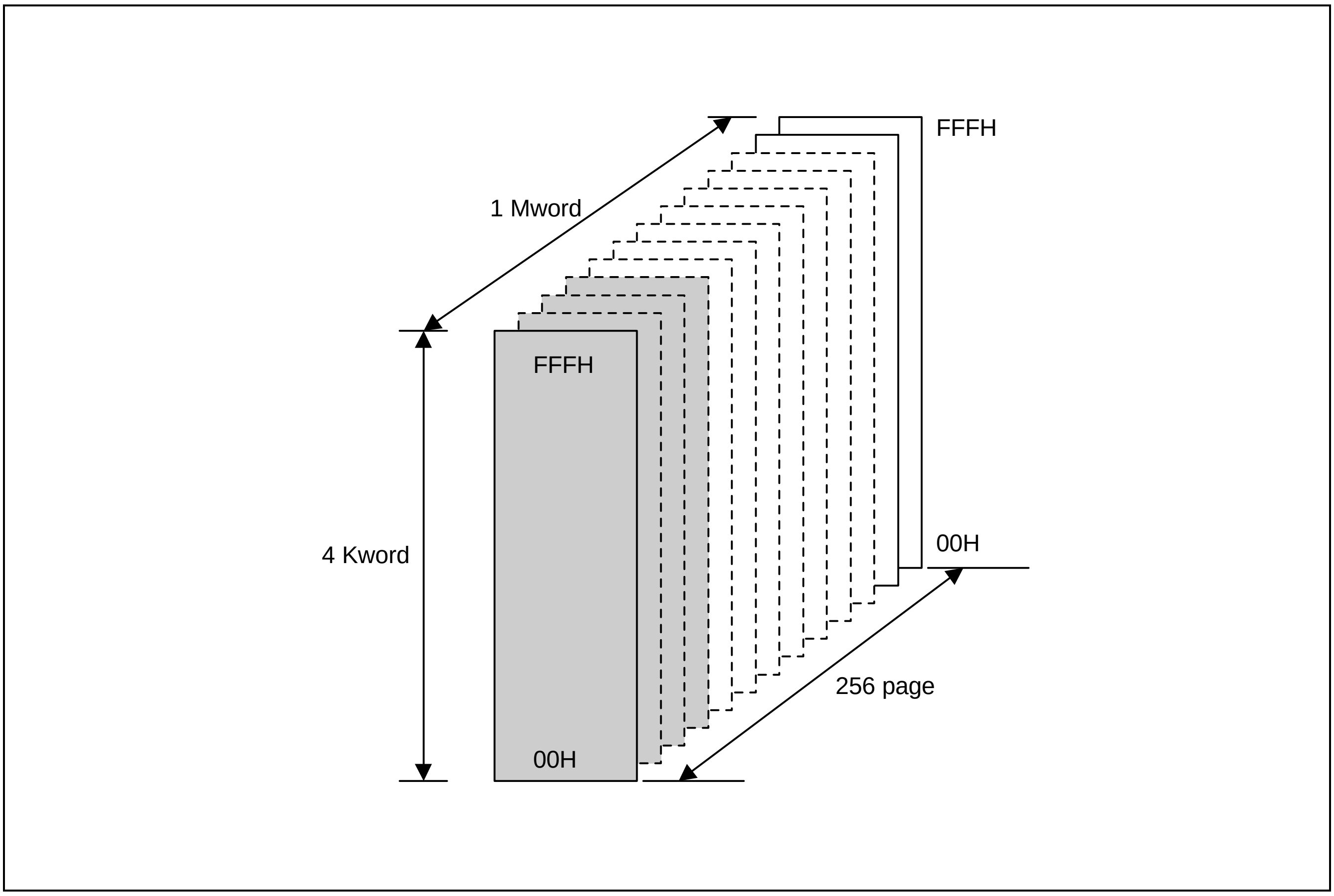


Figure 2-1. Program Memory Organization

For example, if PC[19:0] = 5F79AH, the page index pointed to by PC is 5FH, and the offset in the page is 79AH. If the current PC[19:0] = 5EFFFH and the instruction pointed to by the current PC (i.e., the instruction at the address 5EFFFH is *not* a branch instruction), the next PC becomes 5E000H, *not* 5F000H. In other words, the instruction sequence wraps around at the page boundary, unless the instruction at the boundary (in the above example, at 5EFFFH) is a long branch instruction. The only way to change the program page is by long branches (CALL, LNK, and JP), where the absolute branch target address is specified. For example, if the current PC[19:0] = 047ACH (the page index is 04H and the offset is 7ACH) and the instruction pointed to by the current PC (i.e., the instruction at the address 047ACH), is “JP A507FH” (jump to the program address A507FH) , then the next PC[19:0] = A507FH, which means that the page and the offset are changed to A5H and 07FH, respectively. On the other hand, the short branch instructions cannot change the page indices.

Suppose the current PC is 6FFFEH and its instruction is “JR 5H” (jump to the program address PC + 5H), then the next instruction address is 6F003H, not 70003H. In other words, the branch target address calculation also wraps around with respect to a page boundary. This situation is illustrated below:

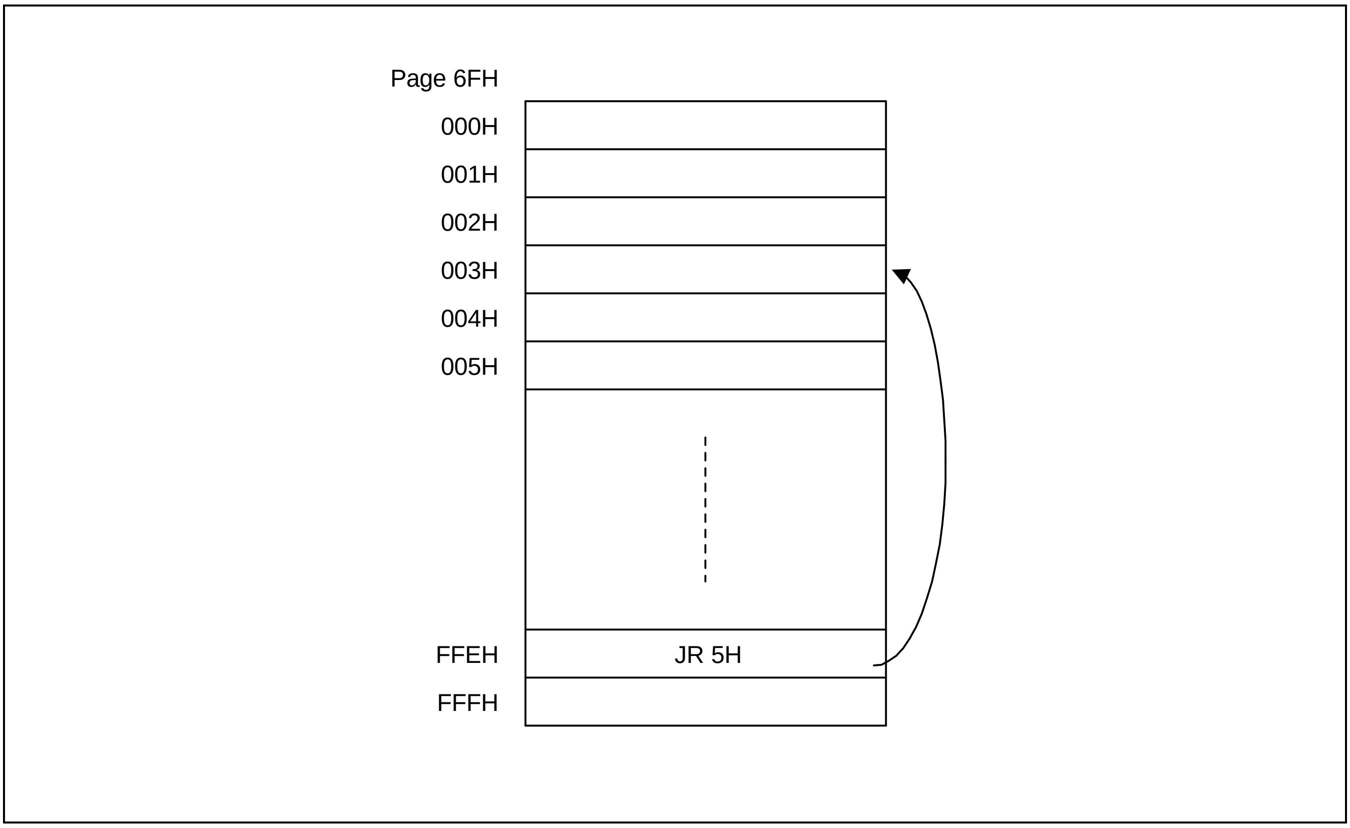
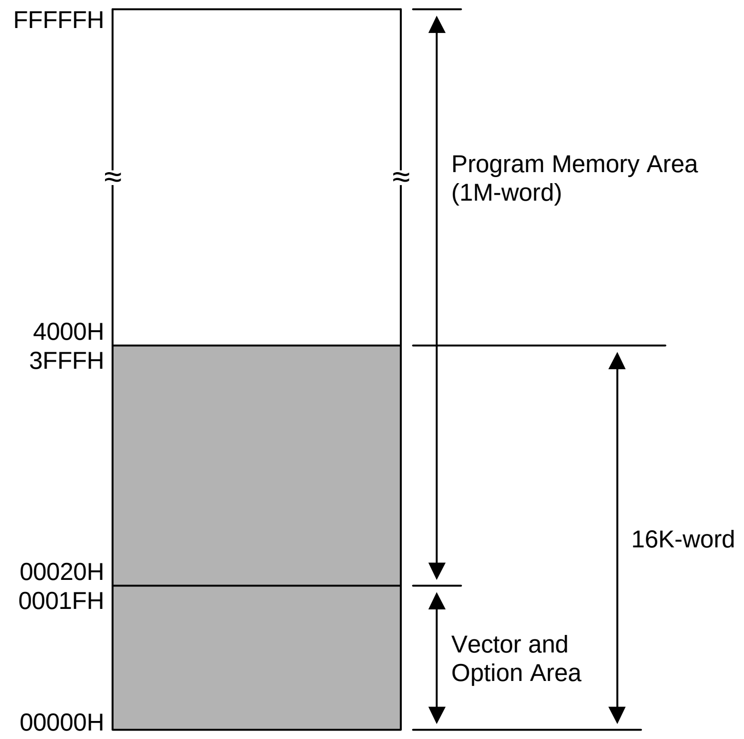


Figure 2-2. Relative Jump Around Page Boundary

Programmers do not have to manually calculate the offset and insert extra instructions for a jump instruction across page boundaries. The compiler and the assembler for CalmRISC are in charge of producing appropriate codes for them.



NOTE: S3CB519/FB519 has totally 16K-word (32K-byte) program memory area.

Figure 2-3. Program Memory Layout

From 00000H to 00004H addresses are used for vector addresses of exceptions, and 0001EH and 0001FH are used for only the option. Aside from these addresses others are reserved in the vector and option area. Program memory area from the address 00020H to FFFFFH can be used for normal programs.

Because the S3CB519/FB519's program memory is 16 Kword (32K-byte), the block of addresses from 00020H to 3FFFH is the program memory area.

ROM CODE OPTION (RCOD_OPT)

Just after power on, the ROM data located at 0001EH and 0001FH is used as the ROM code option. S3CB519/FB519 has ROM code options like the Reset value of Basic timer and Watchdog timer enable.

For example, if you program as below:

```
opt_sec     section     CODE, abs 0001FH

            opt_sec

            dw          3FFH
```

— fxx/32 is used as Reset value of basic timer (by bit.14, 13, 12)

— Watch-dog timer is enabled (by bit.11)

If you don’t program any values in these option areas, then the default value is “1”.

In these cases, the address 0001EH would be the value of “FFFFH”.

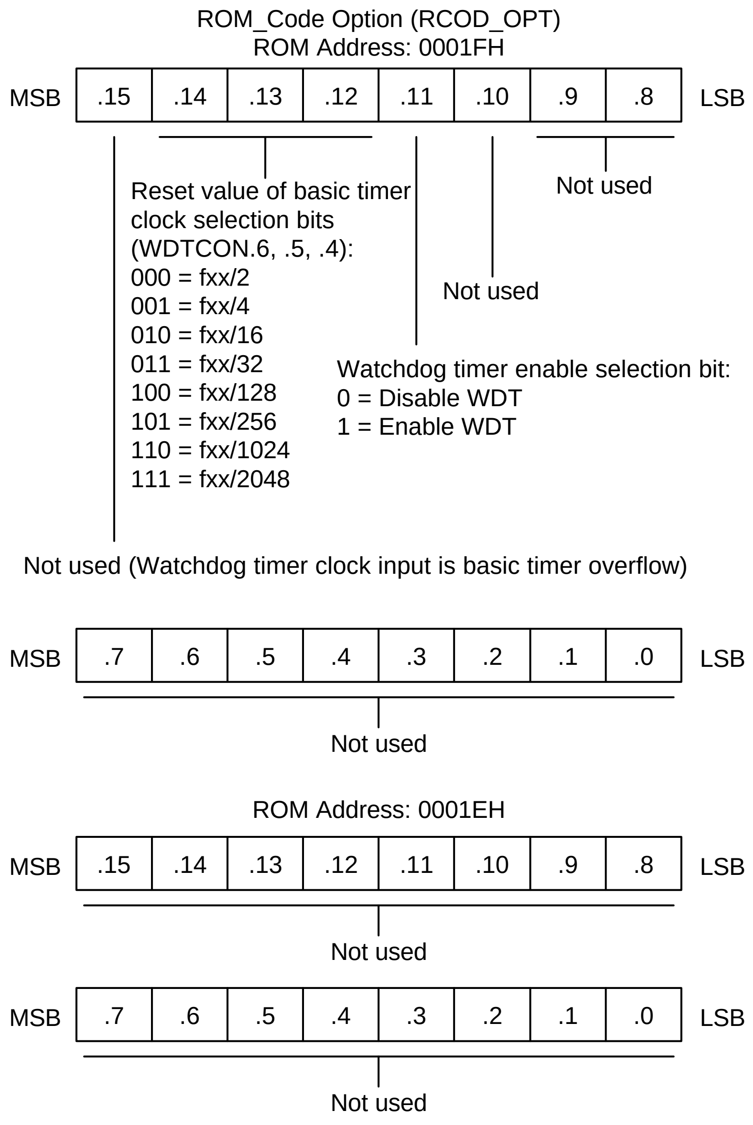


Figure 2-4. ROM Code Option (RCOD_OPT)

DATA MEMORY ORGANIZATION

The total data memory address space is 64K-byte, addressed by *DA[15:0],* and divided into 256 pages, Each page consists of 256 bytes as shown below.

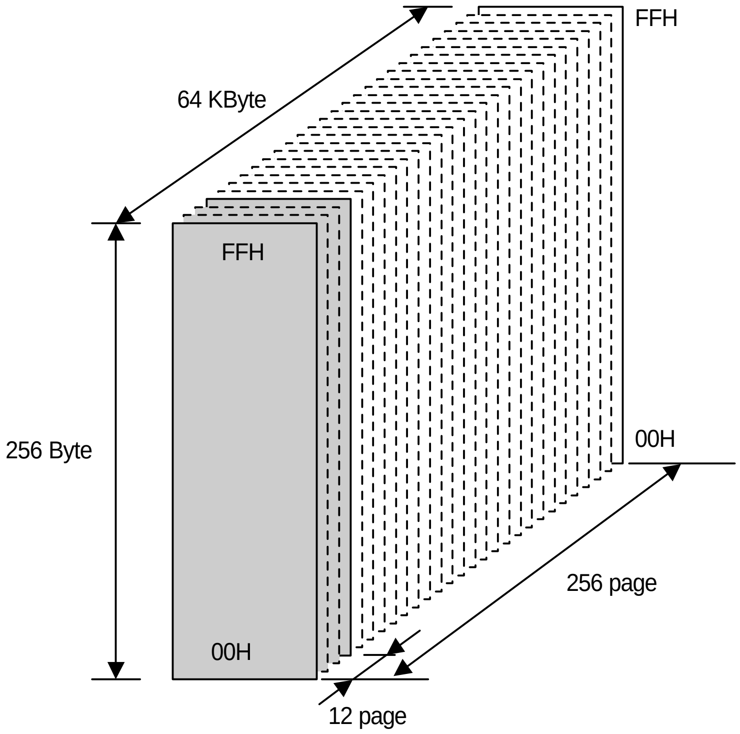


Figure 2-5. Data Memory Map

The data memory page is indexed by SPR and IDH. In data memory index addressing mode, 16-bit data memory address is composed of two 8-bit SPRs, IDH[7:0] and IDL0[7:0] (or IDH[7:0] and IDL1[7:0]). IDH[7:0] points to a page index, and IDL0[7:0] (or IDL1[7:0]) represents the page offset. In data memory direct addressing mode, an 8-bit direct address, adr[7:0], specifies the offset of the page pointed to by IDH[7:0] (See the details for direct addressing mode in the instruction sections). Unlike the program memory organization, data memory address does *not* wrap around. In other words, data memory index addressing with modification performs an addition or a subtraction operation on the whole 16-bit address of IDH[7:0] and IDL0[7:0] (or IDL1[7:0]) and updates IDH[7:0] and IDL0[7:0] (or IDL1[7:0]) accordingly. Suppose IDH[7:0] is 0FH and IDL0[7:0] is FCH and the modification on the index registers, IDH[7:0] and IDL0[7:0], is increment by 5H, then, after the modification (i.e., 0FFCH + 5 = 1001H), IDH[7:0] and IDL0[7:0] become 10H and 01H, respectively.

As for the MAC816 coprocessor, the data memory is a word unit (16-bit wide) and is divided to X-memory and Y-memory for DSP instruction.
The address 0080H in CalmRISC, for example, is viewed as 0040H by MAC816.

The S3CB519/FB519 has a total of 3072 bytes of data register address from 0080H to 0C7FH.
The area from 0000H to 007FH is for peripheral control, and LCD RAM area is from 0C80H to 0CEFH.
The MAC816 views the peripheral control register area as being from 0000H to 003FH, and X-memory from 0040H to 043FH, and Y-memory from 0440H to 063FH.

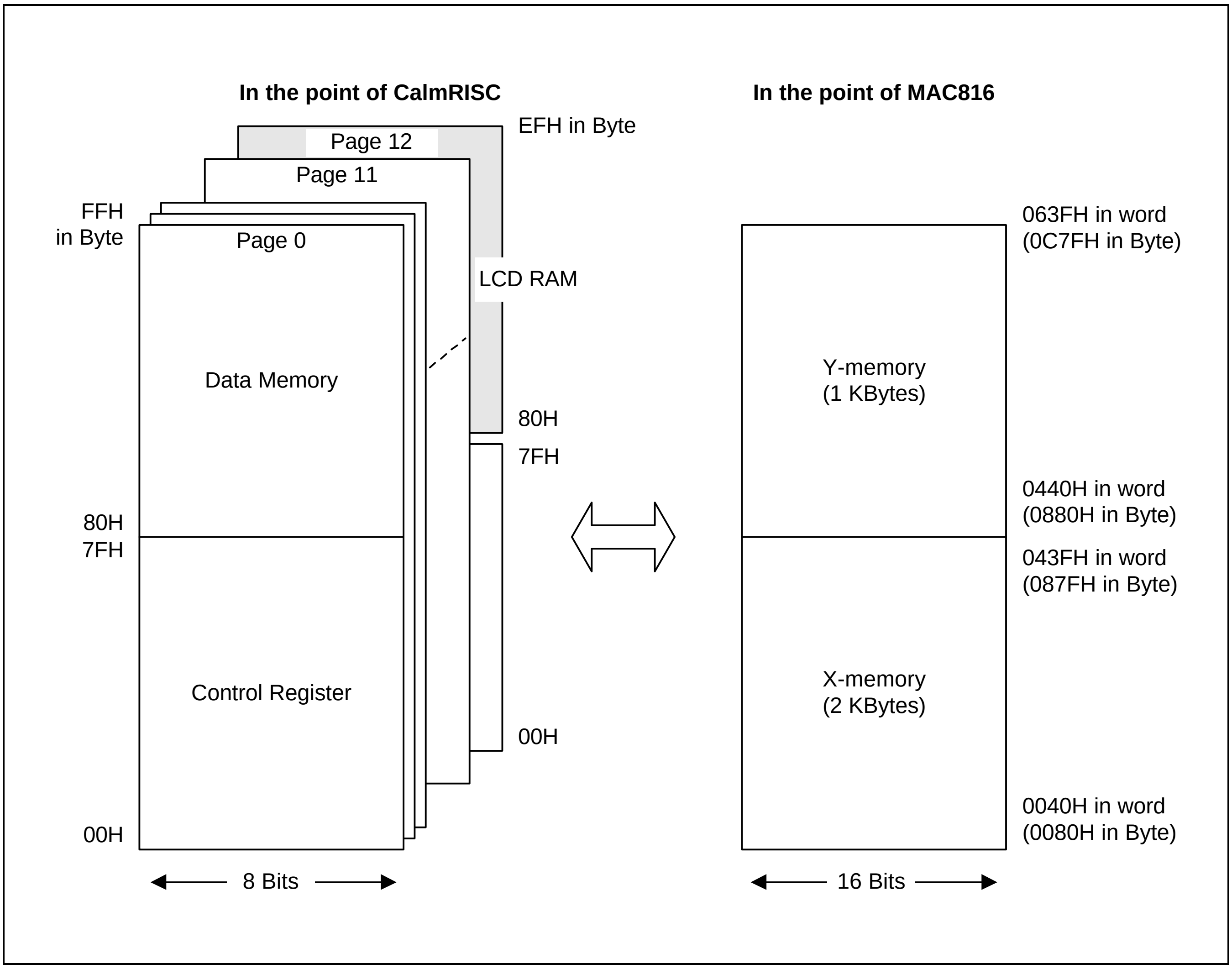


Figure 2-6. S3CB519/FB519 Data Memory Map

I/O PIN DESCRIPTION

Table 1-1. S3CB519 Pin Descriptions

Pin Name	Pin Type	Pin Description	Circuit Type	Share Pins
P0.0 P0.1 P0.2 P0.3 P0.4 P0.5 P0.6	I/O	I/O port with bit programmable pins; Input and output modes can be selected by software; Software assignable pull-up. Alternately, P0.0–P0.6 can be used as INT0–INT6, TB, T0, T0CAP, T0PWM, T0CK, BUZ, TACK, BLD, SCK, SO, SI.	D-2 D-2 D-2 F-10 D-2 D-2 D-2	INT0/TB INT1/T0/T0CAP/ T0PWM INT2/T0CK/BUZ INT3/TACK/BLD INT4/SCK INT5/SO INT6/SI
P1.0–P1.3	I/O	I/O port with bit programmable pins; Input and output modes can be selected by software; Software assignable pull-up. Alternately, P1.0–P1.3 can be used as KS0–KS3.	D-2	KS0–KS3
P2.0–P2.7	I/O	I/O port with 4-bit programmable pins; Input and output modes can be selected by software; Software assignable pull-up. Alternately, P2.0–P2.7 can be used as SEG55–SEG48.	H-35	SEG55–SEG48
P3.0–P3.7	I/O	I/O port with 4-bit programmable pins; Input and output modes can be selected by software; Software assignable pull-up. Alternately, P3.0–P3.7 can be used as SEG47–SEG40.	H-35	SEG47–SEG40
P4.0–P4.7	I/O	I/O port with 4-bit programmable pins; Input and output modes can be selected by software; Software assignable pull-up. Alternately, P4.0–P4.7 can be used as COM8–COM15.	H-35	COM8–COM15
P5.0–P5.15	O	Key strobe output port with 4-bit programmable pins. Push-pull and open-drain modes can be selected by software. Alternately, P5.0–P5.15 can be used as SEG39–SEG24.	H-34	SEG39–SEG24
SEG0–SEG23	O	LCD segment signal output.	H-29	–
SEG24–SEG39	O		H-34	P5.0–P5.15
SEG40–SEG55	I/O		H-35	P3.0–P3.7 P2.0–P2.7
COM0–COM7	O	LCD common signal output.	H-29	–
COM8–COM15	O		H-35	P4.0–P4.7
KS0–KS3	I/O	Key interrupt and/or external interrupt inputs.	D-2	P1.0–P1.3
INT0–INT2, INT4–INT6	I/O	External interrupt input.	D-2	P0.0–P0.2, P0.4–P0.6
INT3	I/O		F-10	P0.3

Table 1-1. S3CB519 Pin Descriptions (Continued)

Pin Name	Pin Type	Pin Description	Circuit Type	Share Pins
TB	I/O	Timer B clock output.	D-2	P0.0
T0	I/O	Timer 0 clock output.	D-2	P0.1
T0CAP	I/O	Timer 0 capture input.	D-2	P0.1
T0PWM	I/O	Timer 0 PWM output.	D-2	P0.1
T0CK	I/O	Timer 0 clock input.	D-2	P0.2
BUZ	I/O	Buzzer output.	D-2	P0.2
TACK	I/O	Timer A clock input.	F-10	P0.3
BLD	I/O	Battery level detector input.	F-10	P0.3
SCK	I/O	Serial I/O interface clock signal.	D-2	P0.4
SO	I/O	Serial data output.	D-2	P0.5
SI	I/O	Serial data input.	D-2	P0.6
DAOUT	O	DAC analog output.	–	–
AV_{DD}	–	Analog power.	–	–
AV_{SS}	–	Analog ground.	–	–
ADINP	I	Analog input positive.	–	–
ADINN	I	Analog input negative.	–	–
ADGAIN	–	Analog input gain control.	–	–
AVREFOUT	O	Analog reference voltage output.	–	–
REFH	–	Analog reference power.	–	–
REFL	–	Analog reference ground.	–	–
V_{DD}	–	Main power supply.	–	–
V_{SS}	–	Ground	–	–
X_{IN}, X_{OUT}	–	Crystal, Ceramic or RC oscillator pins for system clock.	–	–
XT_{IN}, XT_{OUT}	–	Crystal oscillator pins for subsystem clock.	–	–
TEST	I	Chip test input pin. Hold GND with the device is operating.	–	–
RESET	I	Reset signal	B	–

PIN CIRCUIT DIAGRAMS

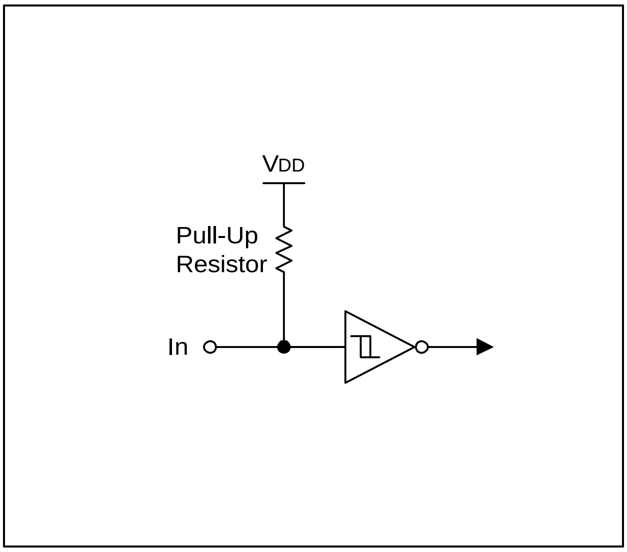


Figure 1-7. Pin Circuit Type B

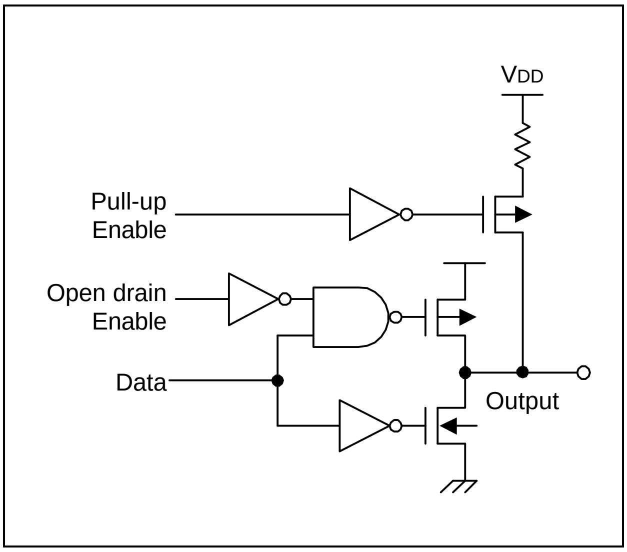


Figure 1-9. Pin Circuit Type E-2

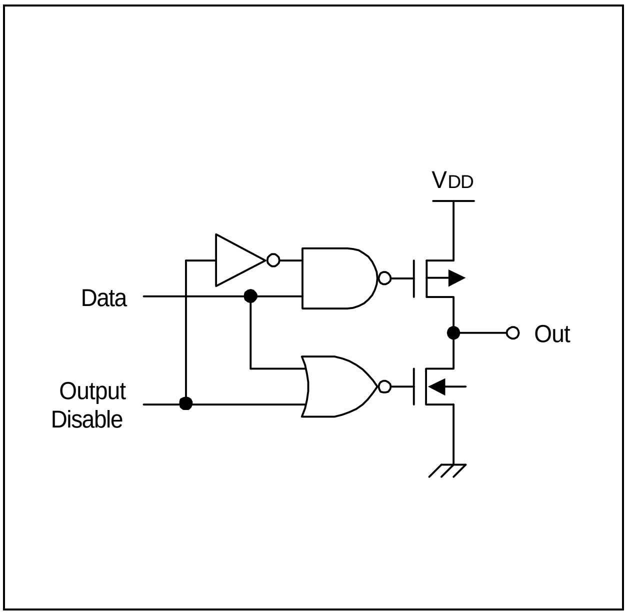


Figure 1-8. Pin Circuit Type C

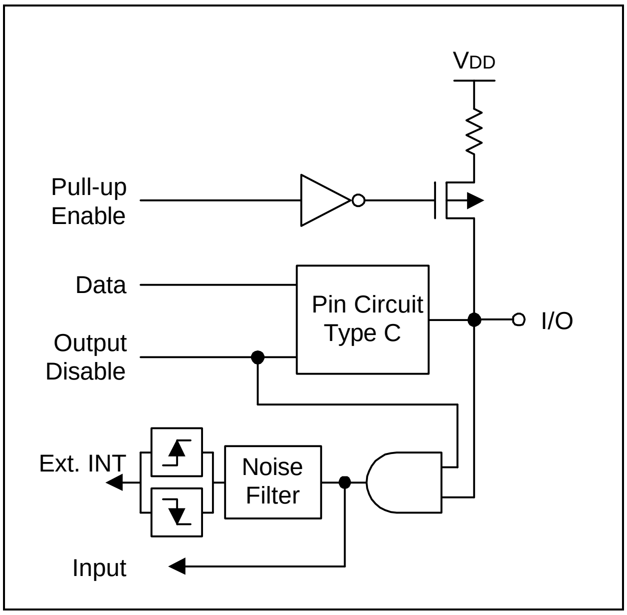


Figure 1-10. Pin Circuit Type D-2

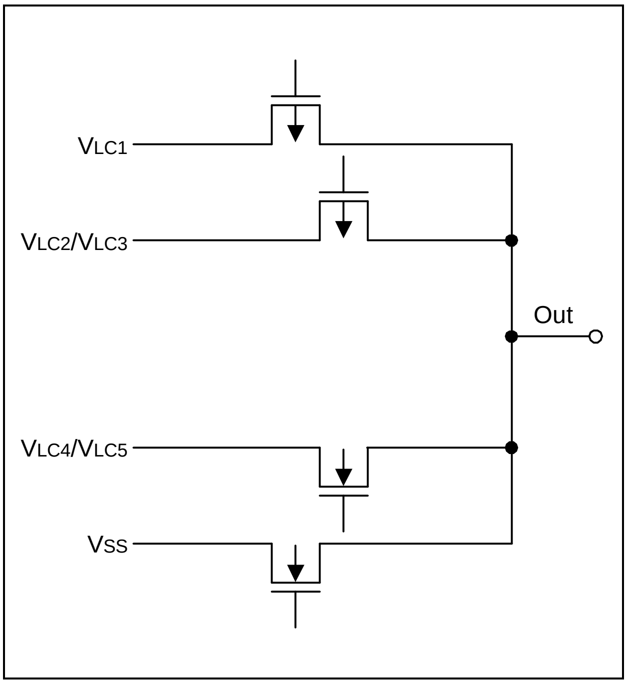


Figure 1-11. Pin Circuit Type H-29

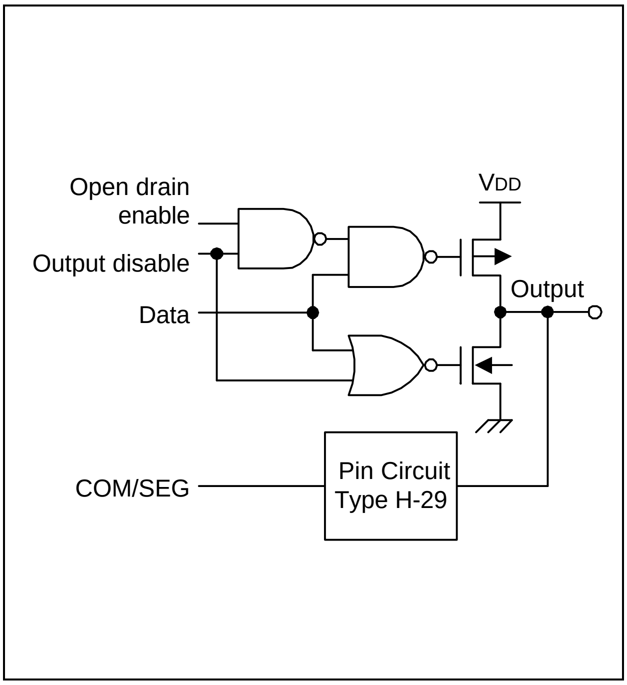


Figure 1-13. Pin Circuit Type H-34

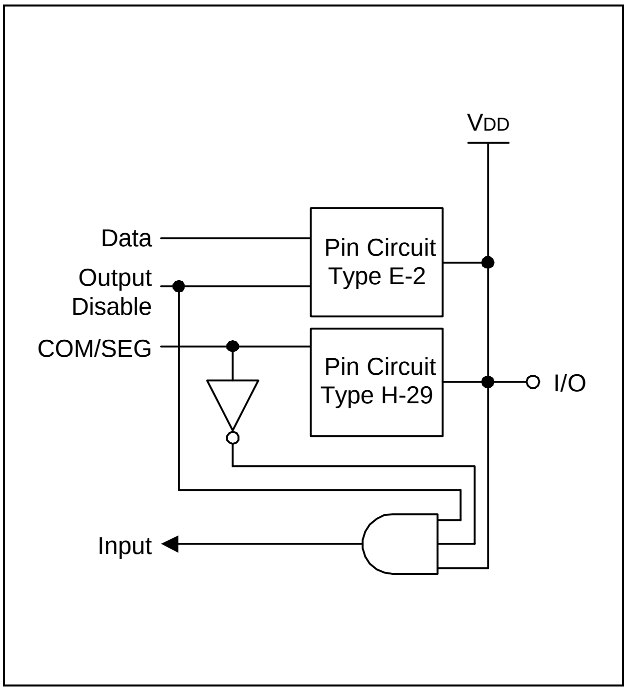


Figure 1-12. Pin Circuit Type H-35

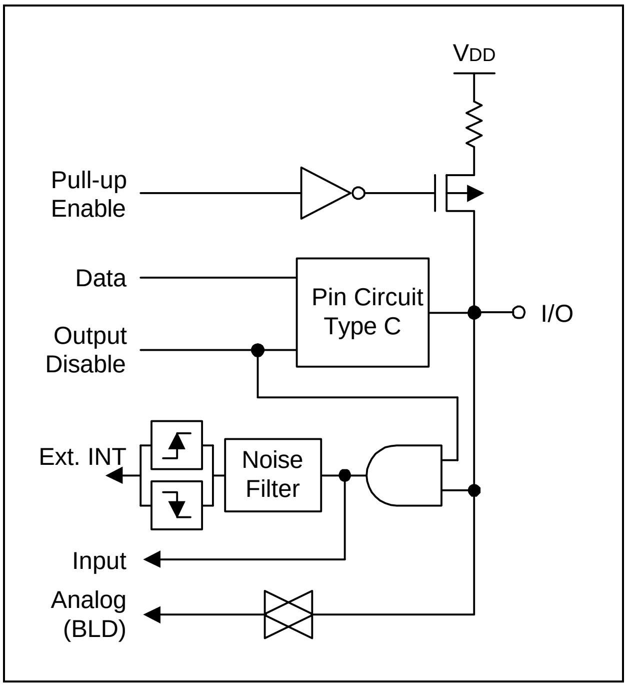


Figure 1-14. Pin Circuit Type F-10

NOTES

3 REGISTERS

OVERVIEW

The registers of CalmRISC are grouped into 2 types: general purpose registers and special purpose registers.

Table 3-1. General and Special Purpose Registers

Registers		Mnemonics	Description	Reset Value
General Purpose Registers (GPR)		**R0**	General Register 0	Unknown
		R1	General Register 1	Unknown
		R2	General Register 2	Unknown
		R3	General Register 3	Unknown
Special Purpose Registers (SPR)	Group 0 (SPR0)	**IDL0**	Lower Byte of Index Register 0	Unknown
		IDL1	Lower Byte of Index Register 1	Unknown
		IDH	Higher Byte of Index Register	Unknown
		SR0	Status Register 0	00H
	Group 1 (SPR1)	**ILX**	Instruction Pointer Link Register for Extended Byte	Unknown
		ILH	Instruction Pointer Link Register for Higher Byte	Unknown
		ILL	Instruction Pointer Link Register for Lower Byte	Unknown
		SR1	Status Register 1	Unknown

GPRs can be used in most instructions such as ALU instructions, stack instructions, load instructions, *etc* .(See the instruction set sections). From the programming standpoint, they have almost no restriction whatsoever. CalmRISC has 4 banks of GPRs, and each bank has 4 registers, R0, R1, R2, and R3. Hence, 16 GPRs in total are available. The GPR bank can be switched by setting an appropriate value in SR0[4:3] (See SR0 for details). The ALU operations between GPRs from different banks are *not* allowed.

SPRs are designed for their own dedicated purposes. They have some restrictions in terms of instructions that can access them. For example, direct ALU operations cannot be performed on SPRs. However, data transfers between a GPR and an SPR are allowed, and stack operations with SPRs are also possible (See the instruction sections for details).

INDEX REGISTERS: IDH, IDL0 AND IDL1

IDH in concatenation with IDL0 (or IDL1) forms a 16-bit data memory address. Note that CalmRISC's data memory address space is 64 Kbyte (addressable by 16-bit addresses). Basically, IDH points to a page index, and IDL0 (or IDL1) corresponds to an offset of the page. Like GPRs, the index registers are 2-way banked. There are 2 banks in total, each of which has its own index registers, IDH, IDL0 and IDL1. The banks of index registers can be switched by setting an appropriate value in SR0[2] (See SR0 for details). Normally, programmers can reserve an index register pair, IDH and IDL0 (or IDL1), for software stack operations.

LINK REGISTERS: ILX, ILH AND ILL

The link registers are specially designed for link-and-branch instructions (See LNK and LRET instructions in the instruction sections for details). When an LNK instruction is executed, the current PC[19:0] is saved into ILX, ILH and ILL registers (i.e., PC[19:16] into ILX[3:0], PC[15:8] into ILH [7:0]) and PC[7:0] into ILL[7:0], respectively. When an LRET instruction is executed, the PC value returned is recovered from ILX, ILH, and ILL (i.e., ILX[3:0] into PC[19:16], ILH[7:0] into PC[15:8] and ILL[7:0] into PC[7:0], respectively). These registers are used to access program memory by LDC instructions. When an LDC instruction is executed, the (code) data residing at the program address specified by ILX:ILH:ILL will be read into TBH:TBL.

There is a special core input pin signal, *nP64KW*, which is reserved for indicating that the program memory address space is only 64 Kword. By grounding the signal pin to zero, the upper 4 bits of PC, PC[19:16], is deactivated and therefore its program memory address signals from CalmRISC core are also deactivated. This, in turn, totally eliminates the power consumption due to manipulating the upper 4 bits of PC (See the core pin description section for details). From the programmer's standpoint, when *nP64KW* is tied to the ground level, then PC[19:16] is *not* saved into ILX for LNK instructions and ILX is *not* read back into PC[19:16] for LRET instructions. Therefore, ILX is totally unused in LNK and LRET instructions when *nP64KW* = 0.

STATUS REGISTER 0: SR0

SR0 is mainly reserved for system control functions, and each bit of SR0 has its own dedicated function.

Table 3-2. Status Register 0 Configuration

Flag Name	Bit	Description
eid	0	Data memory page selection in direct addressing
ie	1	Global interrupt enable
idb	2	Index register banking selection
grb[1:0]	4,3	GPR bank selection
exe	5	Stack overflow/underflow exception enable
ie0	6	Interrupt 0 enable
ie1	7	Interrupt 1 enable

SR0[0] (or eid) selects which page index is to be used in direct addressing. If eid = 0, then page 0 (page index = 0) is used. Otherwise (eid = 1), IDH of the current index register bank is used for the page index. SR0[1] (or ie) is the global interrupt enable flag. As explained in the interrupt/exception section, CalmRISC has 3 interrupt sources (non-maskable interrupt, interrupt 0, and interrupt 1) and 1 stack exception. Both interrupt 0 and interrupt 1 are masked by setting SR0[1] to 0 (i.e., ie = 0). When an interrupt is serviced, the global interrupt enable flag ie is automatically cleared. The execution of an IRET instruction (return from an interrupt service routine) automatically sets ie = 1. SR0[2] (or idb) and SR0[4:3] (or grb[1:0]) selects an appropriate bank for index registers and GPRs, respectively, as shown below:

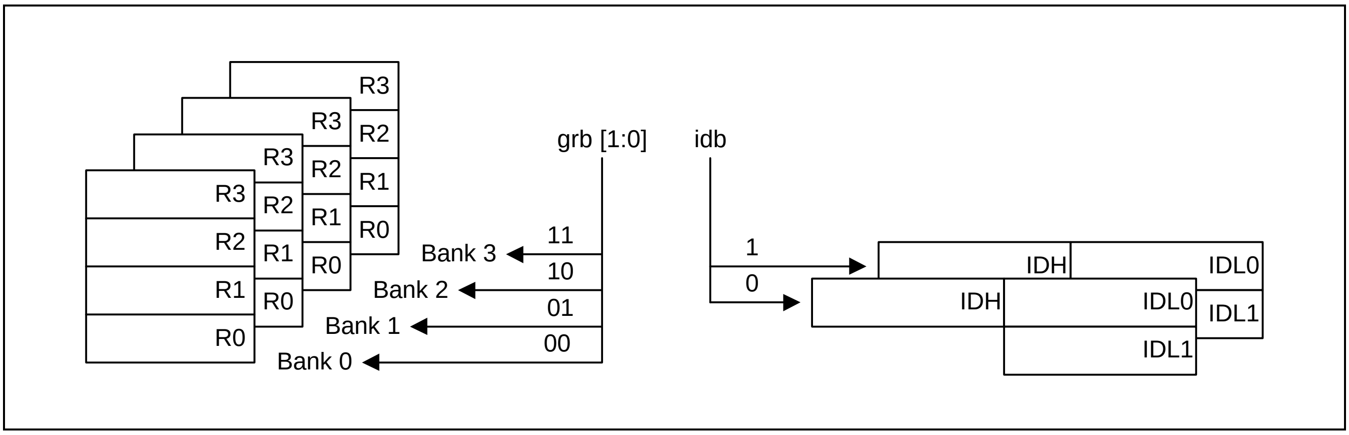


Figure 3-1. Bank Selection by Setting of GRB Bits and IDB Bit

SR0[5] (or exe) enables the stack exception, that is, the stack overflow/underflow exception. If exe = 0, the stack exception is disabled. The stack exception can be used for program debugging in the software development stage. SR0[6] (or ie0) and SR0[7] (or ie1) are enabled, by setting them to 1. Even though ie0 or ie1 are enabled, the interrupts are ignored (not serviced) if the global interrupt enable flag ie is set to 0.

STATUS REGISTER 1: SR1

SR1 is the register for status flags such as ALU execution flag and stack full flag.

Table 3-3. Status Register 1: SR1

Flag Name	Bit	Description
C	0	Carry flag
V	1	Overflow flag
Z	2	Zero flag
N	3	Negative flag
SF	4	Stack Full flag
–	5, 6, 7	Reserved

SR1[0] (or C) is the carry flag of ALU executions. SR1[1] (or V) is the overflow flag of ALU executions. It is set to 1 if and only if the carry-in into the 8-th bit position of addition/subtraction differs from the carry-out from the 8-th bit position. SR1[2] (or Z) is the zero flag, which is set to 1 if and only if the ALU result is zero. SR1[3] (or N) is the negative flag. Basically, the most significant bit (MSB) of ALU results becomes the N flag. Note, a load instruction into a GPR is considered an ALU instruction. However, if an ALU instruction touches the overflow flag (V) like ADD, SUB, CP, *etc*, N flag is updated as exclusive-OR of V and the MSB of the ALU result. This implies that even if an ALU operation results in an overflow, N flag is still valid. SR1[4] (or SF) is the stack overflow flag. It is set when the hardware stack is overflowed or underflowed. Programmers can check if the hardware stack has any abnormalities through the stack exception or testing if SF is set (See the hardware stack section for more details).

NOTE

When an interrupt occurs, the hardware does not save SR0 and SR1, so the software must save the SR1 register values.

4 MEMORY MAP

OVERVIEW

To support the control of peripheral hardware, the addresses of peripheral control registers are memory-mapped to page 0 of the RAM. Memory mapping lets you use a mnemonic as the operand of an instruction at a specific memory location.
In this section, detailed descriptions of the S3CB519/FB519 control registers are presented in an easy-to-read format.
You can use this section as a quick-reference source when writing application programs.

This memory area can be accessed with the whole method of data memory access.

— If SR0 bit 0 is “0” then the accessed register area is always page 0.

— If SR0 bit 0 is “1” then the accessed register page is controlled by the proper value of the IDH register.

So if you want to access the memory map area, clear the SR0.0 and use the direct addressing mode.
This method is used for most cases.
The control register is divided into five areas. Here, the system control register area is same in every device.

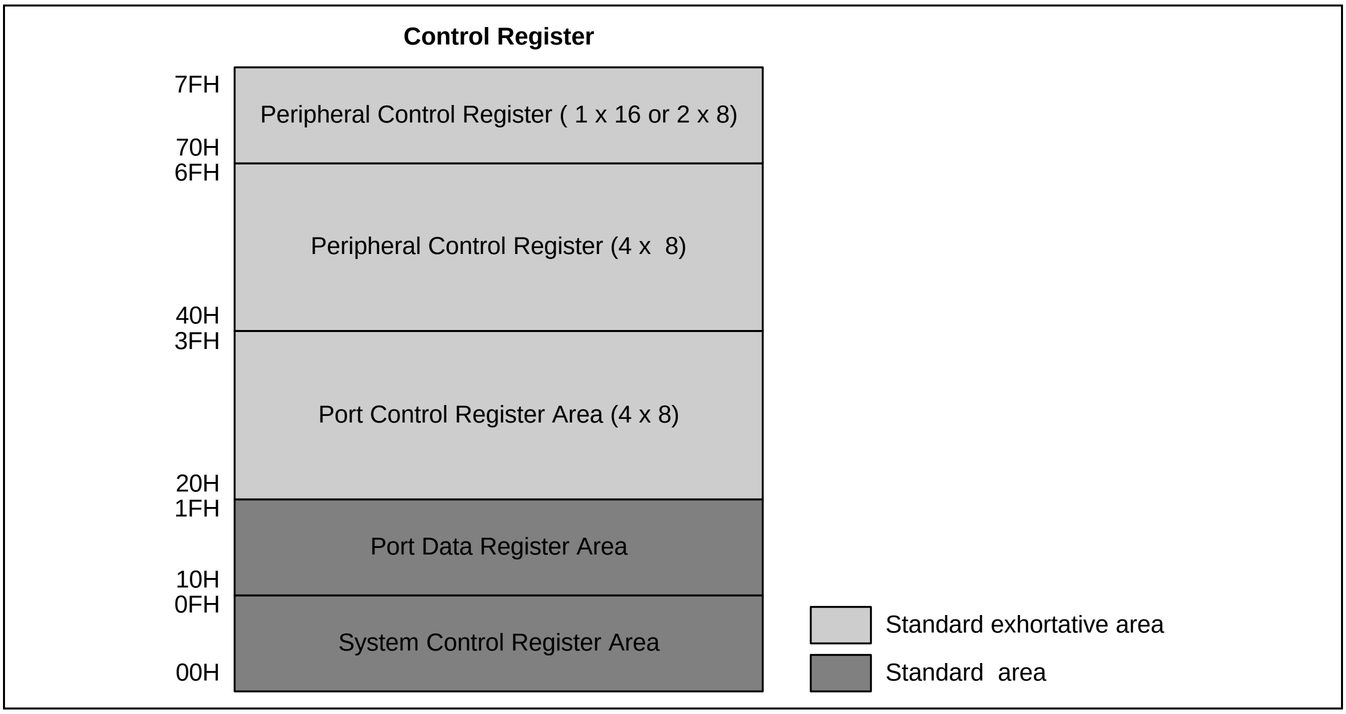

Control Register
7FH
Peripheral Control Register (1 x 16 or 2 x 8)
70H
6FH
Peripheral Control Register (4 x 8)
40H
3FH
Port Control Register Area (4 x 8)
20H
1FH
Port Data Register Area
10H
0FH
System Control Register Area
00H
Standard exhortative area
Standard area

Figure 4-1. Control Register Area

Table 4-1. Control Registers

Register Name	Mnemonic	Decimal	Hex	Reset	R/W
Location 16H–1FH is not mapped					
Port 5 data register	P5	21	15H	**0FH**	R
Port 4 data register	P4	20	14H	00H	R/W
Port 3 data register	P3	19	13H	00H	R/W
Port 2 data register	P2	18	12H	00H	R/W
Port 1 data register	P1	17	11H	00H	R/W
Port 0 data register	P0	16	10H	00H	R/W
Locations 0EH and 0FH are not mapped					
Watchdog timer control register	WDTCON	13	0DH	X0H	R/W
Basic timer counter	BTCNT	12	0CH	00H	R
Interrupt ID register 1	IIR1	11	0BH	–	R/W
Interrupt priority register 1	IPR1	10	0AH	–	R/W
Interrupt mask register 1	IMR1	9	09H	00H	R/W
Interrupt request register 1	IRQ1	8	08H	–	R
Interrupt ID register 00	IIR00	7	07H	–	R/W
Interrupt priority register 00	IPR00	6	06H	–	R/W
Interrupt mask register 00	IMR00	5	05H	00H	R/W
Interrupt request register 00	IRQ00	4	04H	–	R
Oscillator control register	OSCCON	3	03H	00H	R/W
Power control register	PCON	2	02H	**04H**	R/W
Locations 00H and 01H are not mapped					

NOTES

1. All the unused and unmapped registers and bits read “0”.
2. “–“ means undefined.

Table 4-1. Control Registers (Continued)

Register Name	Mnemonic	Decimal	Hex	Reset	R/W
Locations 35H–3FH are not mapped					
Port 5 control register	P5CON	52	34H	00H	R/W
Locations 31H–33H are not mapped					
Port 4 control register	P4CON	48	30H	00H	R/W
Locations 2DH–2FH are not mapped					
Port 3 control register	P3CON	44	2CH	00H	R/W
Locations 29H–2BH are not mapped					
Port 2 control register	P2CON	40	28H	00H	R/W
Locations 26H–27H are not mapped					
Port 1 interrupt control register	P1INT	37	25H	00H	R/W
Port 1 control register	P1CON	36	24H	00H	R/W
Port 0 interrupt edge control register	P0EDGE	35	23H	00H	R/W
Port 0 interrupt control register	P0INT	34	22H	00H	R/W
Port 0 control register low	P0CONL	33	21H	00H	R/W
Port 0 control register high	P0CONH	32	20H	00H	R/W

NOTE: All unused and unmapped registers and bits read “0”.

Table 4-1. Control Registers (Concluded)

Register Name	Mnemonic	Decimal	Hex	Reset	R/W
Locations 74H–7FH are not mapped					
D/A converter data register	DADATA	115	73H	00H	R/W
D/A converter control register	DACON	114	72H	00H	R/W
Battery level detector register	BLDCON	113	71H	**40H**	R/W
Watch timer control register	WTCON	112	70H	00H	R/W
Locations 5FH–6FH are not mapped					
LCD contrast register	LCNST	94	5EH	00H	R/W
LCD mode register	LMOD	93	5DH	00H	R/W
LCD control register	LCON	92	5CH	00H	R/W
Interrupt ID register 01	IIR01	91	5BH	–	R/W
Interrupt priority register 01	IPR01	90	5AH	–	R/W
Interrupt mask register 01	IMR01	89	59H	00H	R/W
Interrupt request register 01	IRQ01	88	58H	–	R
Locations 53H–57H are not mapped					
Timer 0 counter	T0CNT	82	52H	–	R
Timer 0 data register	T0DATA	81	51H	**FFH**	R/W
Timer 0 control register	T0CON	80	50H	00H	R/W
Location 4FH is not mapped					
A/D Converter data register, Low byte	ADATAL	78	4EH	00H	R
A/D Converter data register, High byte	ADATAH	77	4DH	00H	R
A/D Converter control register	ADCON	76	4CH	00H	R/W
Location 4BH is not mapped					
Serial I/O data register	SIODATA	74	4AH	00H	R/W
Serial I/O pre-scale register	SIOPS	73	49H	00H	R/W
Serial I/O control register	SIOCON	72	48H	00H	R/W
Location 47H is not mapped					
Timer B counter	TBCNT	70	46H	–	R
Timer B data register	TBDATA	69	45H	**FFH**	R/W
Timer B control register	TBCON	68	44H	00H	R/W
Location 43H is not mapped					
Timer A counter	TACNT	66	42H	–	R
Timer A data register	TADATA	65	41H	**FFH**	R/W
Timer A control register	TACON	64	40H	00H	R/W

NOTES

1. All unused and unmapped registers and bits read “0”.
2. “–“ means undefined.

5 HARDWARE STACK

OVERVIEW

The hardware stack in CalmRISC has two usages:

— To save and restore the return PC[19:0] on CALL, CALLS, RET, and IRET instructions.

— Temporary storage space for registers on PUSH and POP instructions.

When PC[19:0] is saved into or restored from the hardware stack, the access should be 20 bits wide. On the other hand, when a register is pushed into or popped from the hardware stack, the access should be 8 bits wide. Hence, to maximize the efficiency of the stack usage, the hardware stack is divided into 3 parts: the extended stack bank (XSTACK, 4-bits wide), the odd bank (8-bits wide), and the even bank (8-bits wide).

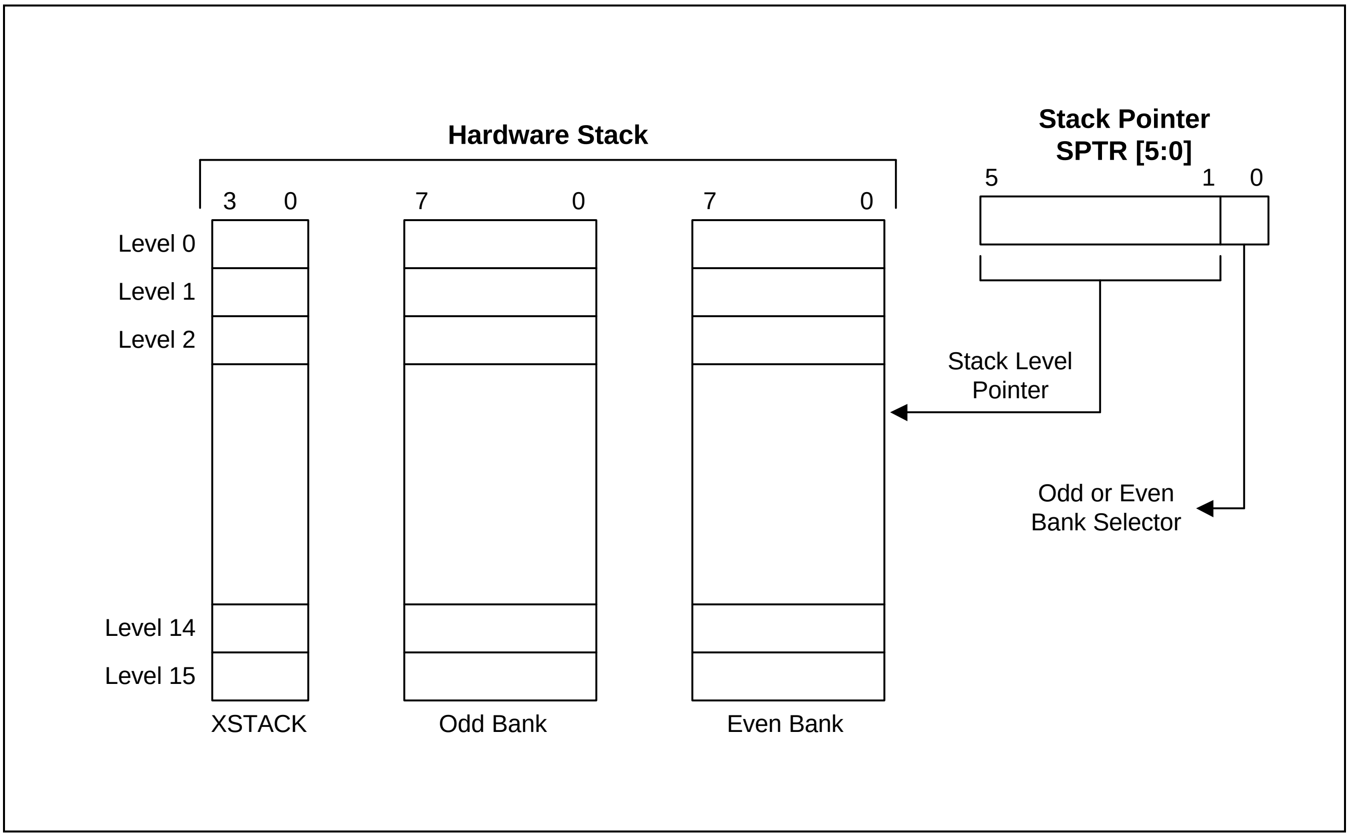


Figure 5-1. Hardware Stack

The top of the stack (TOS) is pointed to by a stack pointer, called **sptr[5:0]**. The upper 5 bits of the stack pointer, sptr[5:1], points to the stack level into which either PC[19:0] or a register is saved. For example, if sptr[5:1] is 5H or TOS is 5, then level 5 of XSTACK is empty and either level 5 of the odd bank or level 5 of the even bank is empty. In fact, sptr[0], the stack bank selection bit, indicates which bank(s) is empty. If sptr[0] = 0, both level 5 of the even and the odd banks are empty. On the other hand, if sptr[0] = 1, level 5 of the odd bank is empty, but level 5 of the even bank is occupied. This situation is well illustrated in the figure below.

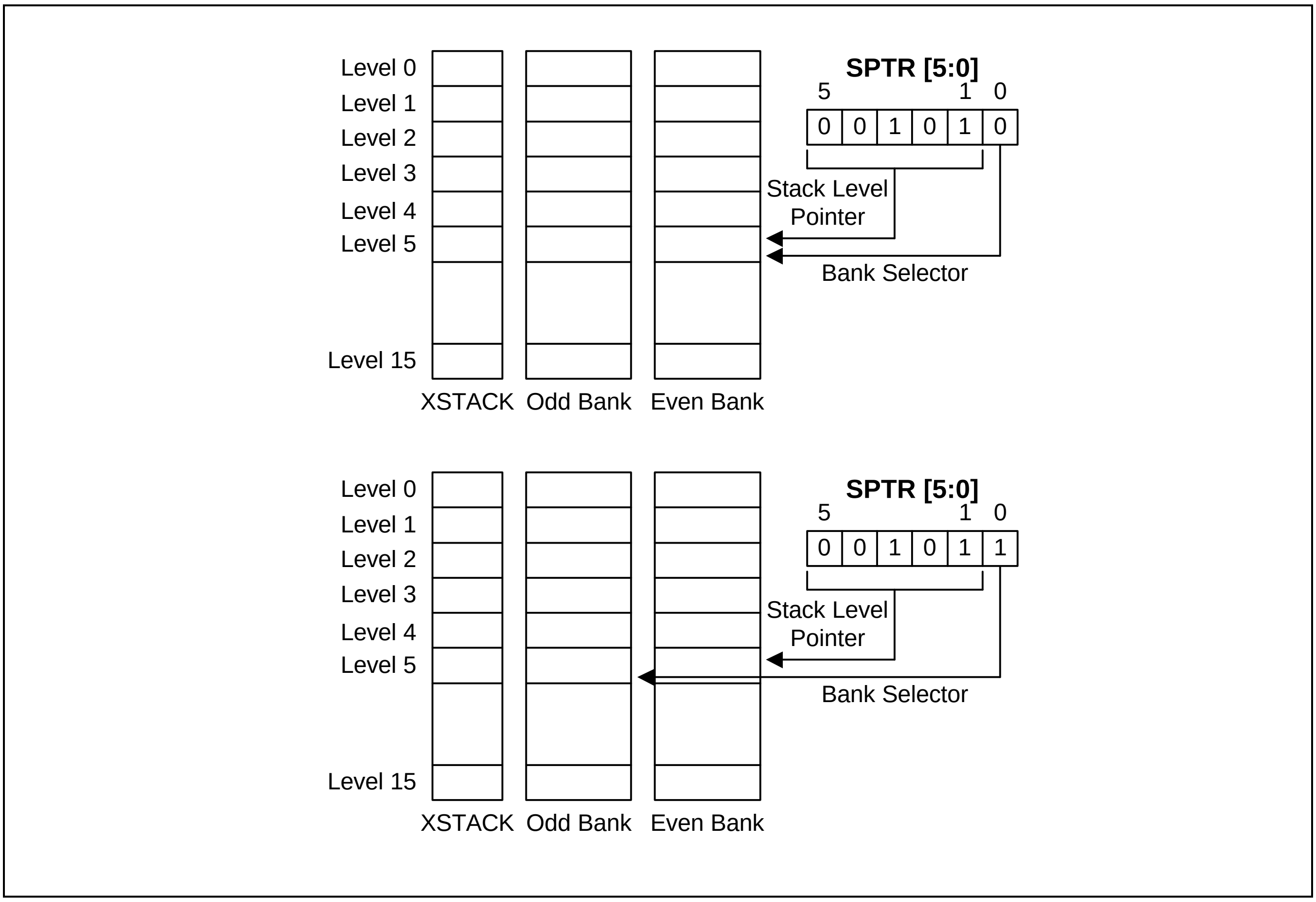


Figure 5-2. Even and Odd Bank Selection Example

As can be seen in the above example, sptr[5:1] is used as the hardware stack pointer when PC[19:0] is pushed or popped and sptr[5:0] as the hardware stack pointer when a register is pushed or popped. Note that XSTACK is used only for storing and retrieving PC[19:16]. Let us consider the cases where PC[19:0] is pushed into the hardware stack (by executing CALL/CALLS instructions or by interrupts/exceptions being served) or is retrieved from the hardware stack (by executing RET/IRET instructions). Regardless of the stack bank selection bit (sptr[0]), TOS of the even bank and the odd bank stores or returns PC[7:0] or PC[15:8], respectively. This is illustrated in the following figures.

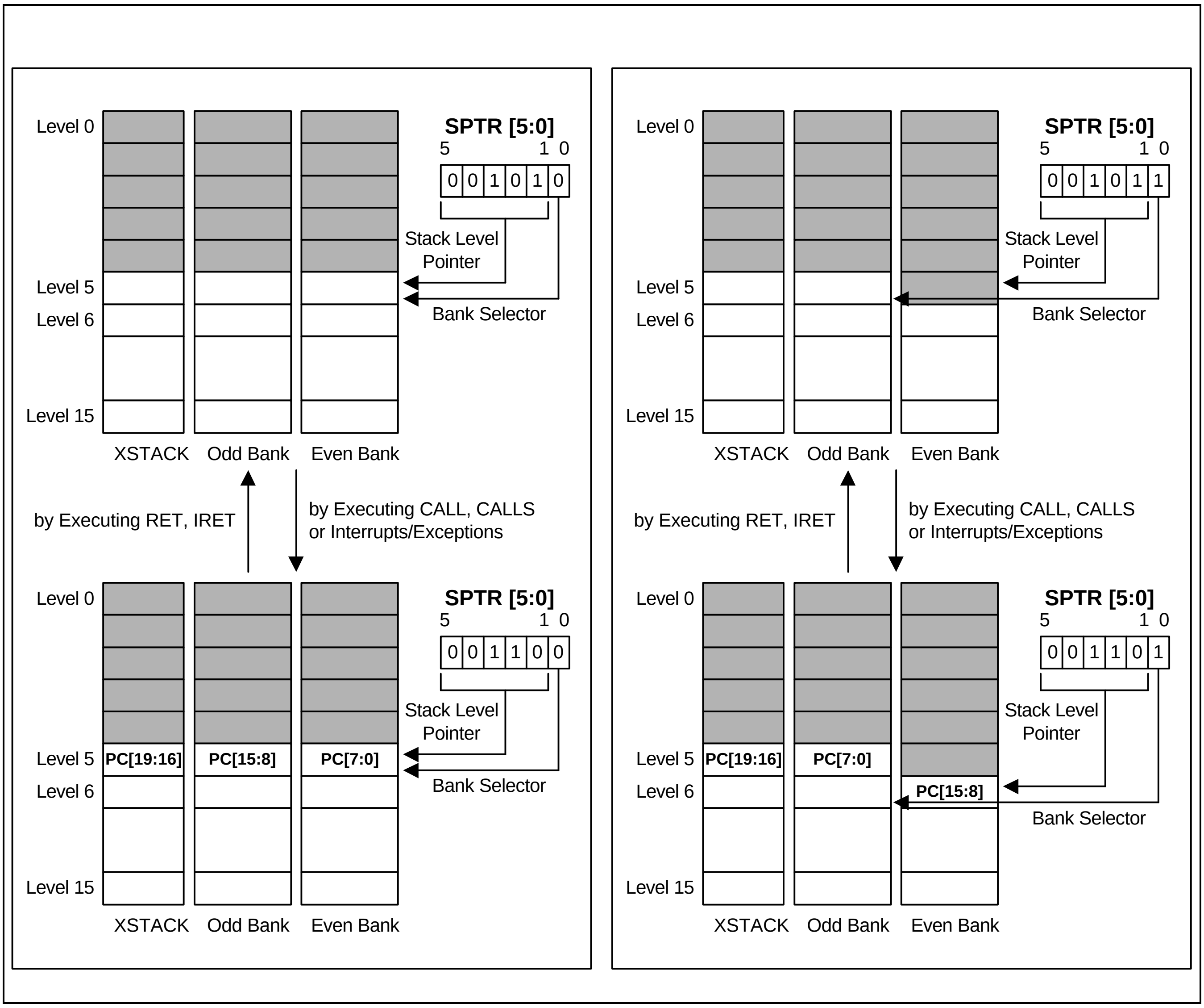


Figure 5-3. Stack Operation with PC [19:0]

As can be seen in the figures, when stack operations with PC[19:0] are performed, the stack level pointer sptr[5:1] (*not* sptr[5:0]) is either incremented by 1 (when PC[19:0] is pushed into the stack) or decremented by 1 (when PC[19:0] is popped from the stack). The stack bank selection bit (sptr[0]) is unchanged. If a CalmRISC core input signal *nP64KW* is 0, which signifies that only PC[15:0] is meaningful, then any access to XSTACK is totally deactivated from the stack operations with PC. Therefore, XSTACK has no meaning when the input pin signal, *nP64KW*, is tied to 0. In that case, XSTACK doesn't have to even exist. As a matter of fact, XSTACK is not included in CalmRISC core itself and it is interfaced through some specially reserved core pin signals (*nPUSH, nSTACK*, *XHSI[3:0]*, *XSHO[3:0]*), if the program address space is more than 64K words (See the core pin signal section for details).

With regards to stack operations with registers, a similar argument can be made.The only difference is that the data written into or read from the stack are a byte. Hence, the even bank and the odd bank are accessed alternately as shown below.

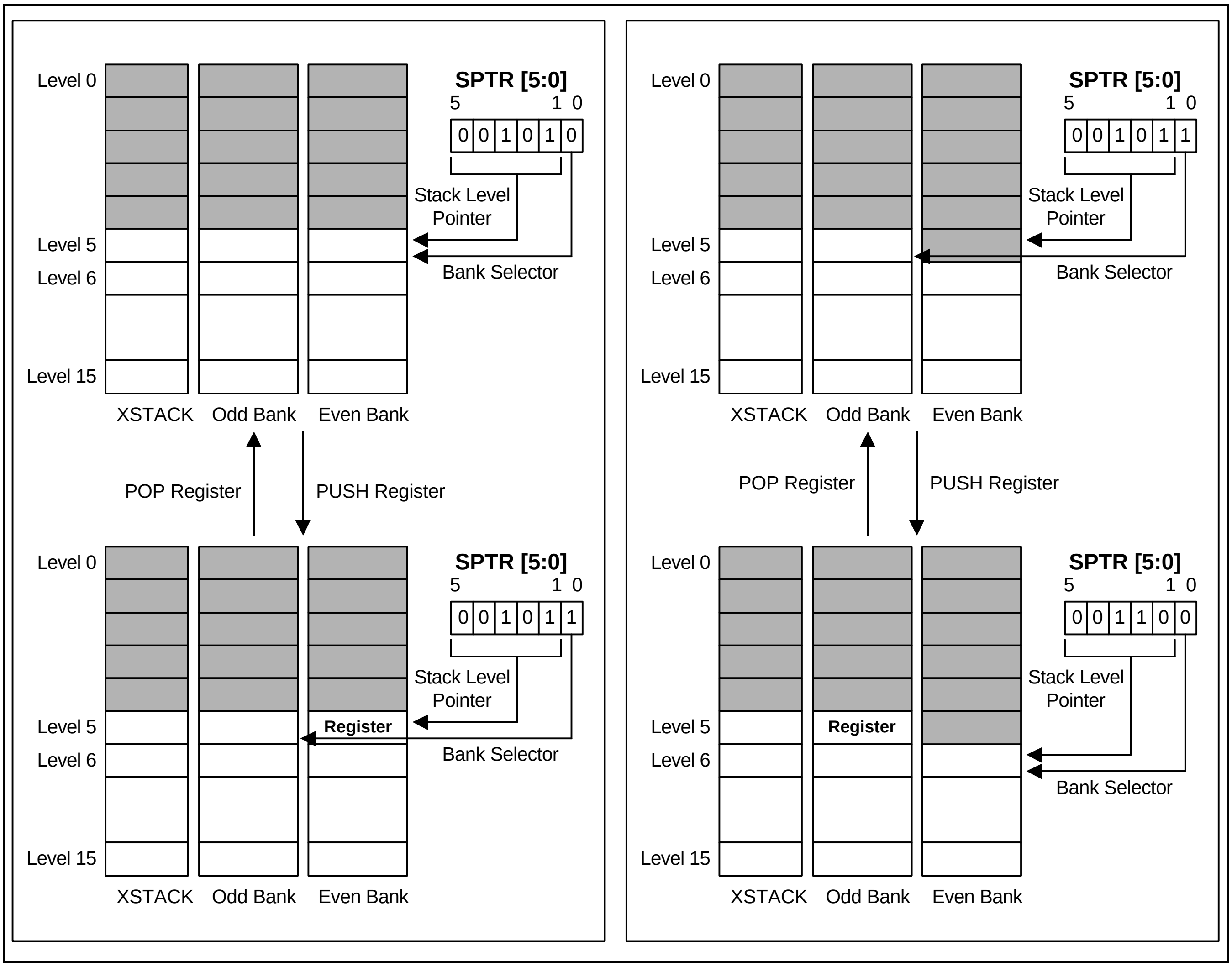


Figure 5-4. Stack Operation with Registers

When the bank selection bit (sptr[0]) is 0, then the register is pushed into the even bank and the bank selection bit is set to 1. In this case, the stack level pointer is unchanged. When the bank selection bit (sptr[0]) is 1, then the register is pushed into the odd bank, the bank selection bit is set to 0, and the stack level pointer is incremented by 1. Unlike the push operations of PC[19:0], any data are not written into XSTACK in the register push operations. This is illustrated in the example figures. When a register is pushed into the stack, sptr[5:0] is incremented by 1 (*not* the stack level pointer sptr[5:1]). The register pop operations are the reverse processes of the register push operations. When a register is popped out of the stack, sptr[5:0] is decremented by 1 (*not* the stack level pointer sptr[5:1]).

Hardware stack overflow/underflow happens when the MSB of the stack level pointer, sptr[5], is 1. This is obvious from the fact that the hardware stack has only 16 levels and the following relationship holds for the stack level pointer in a normal case.

Suppose the stack level pointer sptr[5:1] = 15 (or 01111B in binary format) and the bank selection bit sptr[0] = 1. Here if either PC[19:0] or a register is pushed, the stack level pointer is incremented by 1. Therefore, sptr[5:1] = 16 (or 10000B in binary format) and sptr[5] = 1, which implies that the stack is overflowed. The situation is depicted in the following figure.

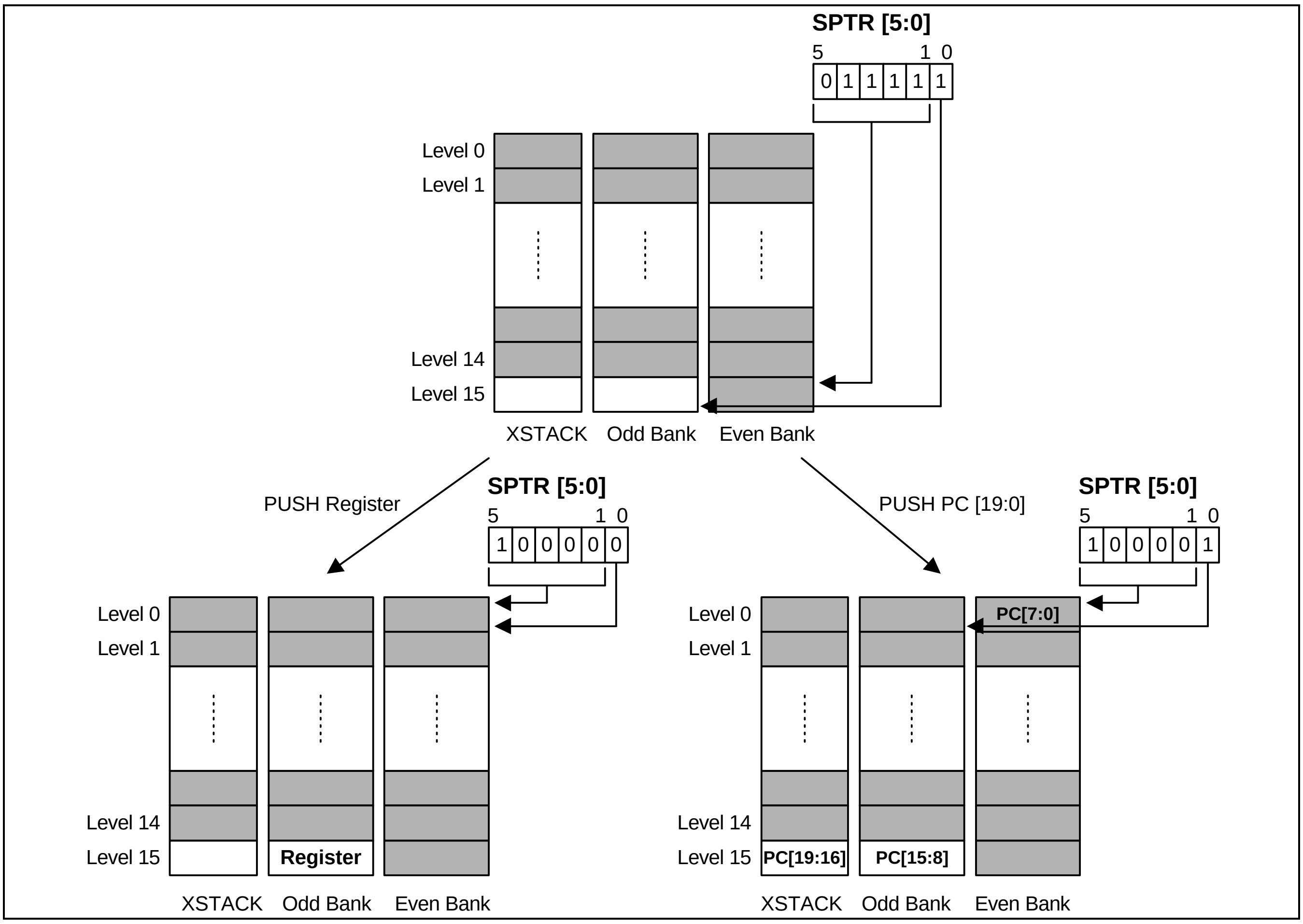


Figure 5-5. Stack Overflow

The first overflow happens due to a register push operation. As explained earlier, a register push operation increments sptr[5:0] (not sptr[5:1]) , which results in sptr[5] = 1, sptr[4:1] = 0 and sptr[0] = 0. As indicated by sptr[5] = 1, an overflow happens. Note that this overflow doesn't overwrite any data in the stack. On the other hand, when PC[19:0] is pushed, sptr[5:1] is incremented by 1 instead of sptr[5:0], and as expected, an overflow results. Unlike the first overflow, PC[7:0] is pushed into level 0 of the even bank and the data that has been there before the push operation is *overwritten*. A similar argument can be made about stack underflows. Note that any stack operation, which causes the stack to overflow or underflow, doesn't necessarily mean that any data in the stack are lost, as is observed in the first example.

In SR1, there is a status flag, SF (Stack Full Flag), which is exactly the same as sptr[5]. In other words, the value of sptr[5] can be checked by reading SF (or SR1[4]). SF is not a sticky flag in the sense that if there was a stack overflow/underflow but any following stack access instructions clear sptr[5] to 0, then SF = 0 and programmers cannot tell whether there was a stack overflow/underflow by reading SF. For example, if a program pushes a register 64 times in a row, sptr[5:0] is exactly the same as sptr[5:0] before the push sequence. Therefore, special attention should be paid.

Another mechanism to detect a stack overflow/underflow is through a stack exception. A stack exception happens only when the execution of any stack access instruction results in SF = 1 (or sptr[5] = 1). Suppose a register push operation makes SF = 1 (the SF value before the push operation doesn't matter). Then the stack exception due to the push operation is immediately generated and served If the stack exception enable flag (exe of SR0) is 1. If the stack exception enable flag is 0, then the generated interrupt is not served but pending. Sometime later when the stack exception enable flag is set to 1, the pending exception request is served even if SF = 0. More details are available in the stack exception section.

NOTES

6 EXCEPTIONS

OVERVIEW

Exceptions in CalmRISC are listed in the table below. Exception handling routines, residing at the given addresses in the table, are invoked when the corresponding exception occurs. The starting address of each exception routine is specified by concatenating 0H (leading 4 bits of 0) and the 16-bit data in the exception vector listed in the table. For example, the interrupt service routine for NMI starts from 0H:PM[00001H]. Note that “:” means concatenation and PM[*] stands for the 16-bit content at the address * of the program memory. Aside from the exception due to reset release, the current PC is pushed in the stack on an exception. When an exception is executed due to NMI/IRQ[1:0]/IEXP, the global interrupt enable flag, ie bit (SR0[1]), is set to 0, whereas ie is set to 1 when IRET or an instruction that explicitly sets ie is executed.

Table 6-1. Exceptions

Name	Address	Priority	Description
Reset	00000H	1 st	Exception due to rest release.
NMI	00001H	2 nd	Exception due to *nNMI* signal. Non-maskable. Not used.
IRQ[0]	00002H	4 th	Exception due to *nIRQ[0]* signal. Maskable by setting ie/ie0.
IRQ[1]	00003H	5 th	Exception due to *nIRQ[1]* signal. Maskable by setting ie/ie1.
IEXP	00004H	3 rd	Exception due to stack full. Maskable by setting exe.
–	00005H	–	Reserved.
–	00006H	–	Reserved.
–	00007H	–	Reserved.

NOTE: Break mode due to BKREQ has a higher priority than all the exceptions above. That is, when BKREQ is active, even the exception due to reset release is not executed.

HARDWARE RESET

When Hardware Reset is active (the reset input signal pin *nRES* = 0), the control pins in the CalmRISC core are initialized to be disabled, and SR0 and sptr (the hardware stack pointer) are initialized to be 0. Additionally, the interrupt sensing block is cleared. When Hardware Reset is released (nRES = 1), the reset exception is executed by loading the JP instruction in IR (Instruction Register) and 0h:0000h in PC. Therefore, when Hardware Reset is released, the “JP {0h:PM[00000h]}” instruction is executed. When the reset exception is executed, a core output signal *nEXPACK* is generated to acknowledge the exception.

NMI EXCEPTION (EDGE SENSITIVE)

On the falling edge of a core input signal *nNMI*, the NMI exception is executed by loading the CALL instruction in IR and 0h:0001h in PC. Therefore, when NMI exception is activated, the “CALL {0h:PM[00001h]}” instruction is executed. When the NMI exception is executed, the ie bit (SR0[1]) becomes 0 and a core output signal *nEXPACK* is generated to acknowledge the exception.

IRQ[0] EXCEPTION (LEVEL-SENSITIVE)

When a core input signal *nIRQ[0]* is low, SR0[6] (ie0) is high, and SR0[1] (ie) is high, IRQ[0] exception is generated, and this will load the CALL instruction in IR (Instruction Register) and 0h:0002h in PC. Therefore, on an IRQ[0] exception, the “CALL {0h:PM[00002h]}” instruction is executed. When the IRQ[0] exception is executed, SR0[1] (ie) is set to 0 and a core output signal *nEXPACK* is generated to acknowledge the exception.

IRQ[1] EXCEPTION (LEVEL-SENSITIVE)

When a core input signal *nIRQ[1]* is low, SR0[7] (ie1) is high, and SR0[1] (ie) is high, IRQ[1] exception is generated, and this will load the CALL instruction in IR (Instruction Register) and 0h:0003h in PC. Therefore, on an IRQ[1] exception, the “CALL {0h:PM[00003h]}” instruction is executed. When the IRQ[1] exception is executed, SR0[1] (ie) is set to 0 and a core output signal *nEXPACK* is generated to acknowledge the exception.

HARDWARE STACK FULL EXCEPTION

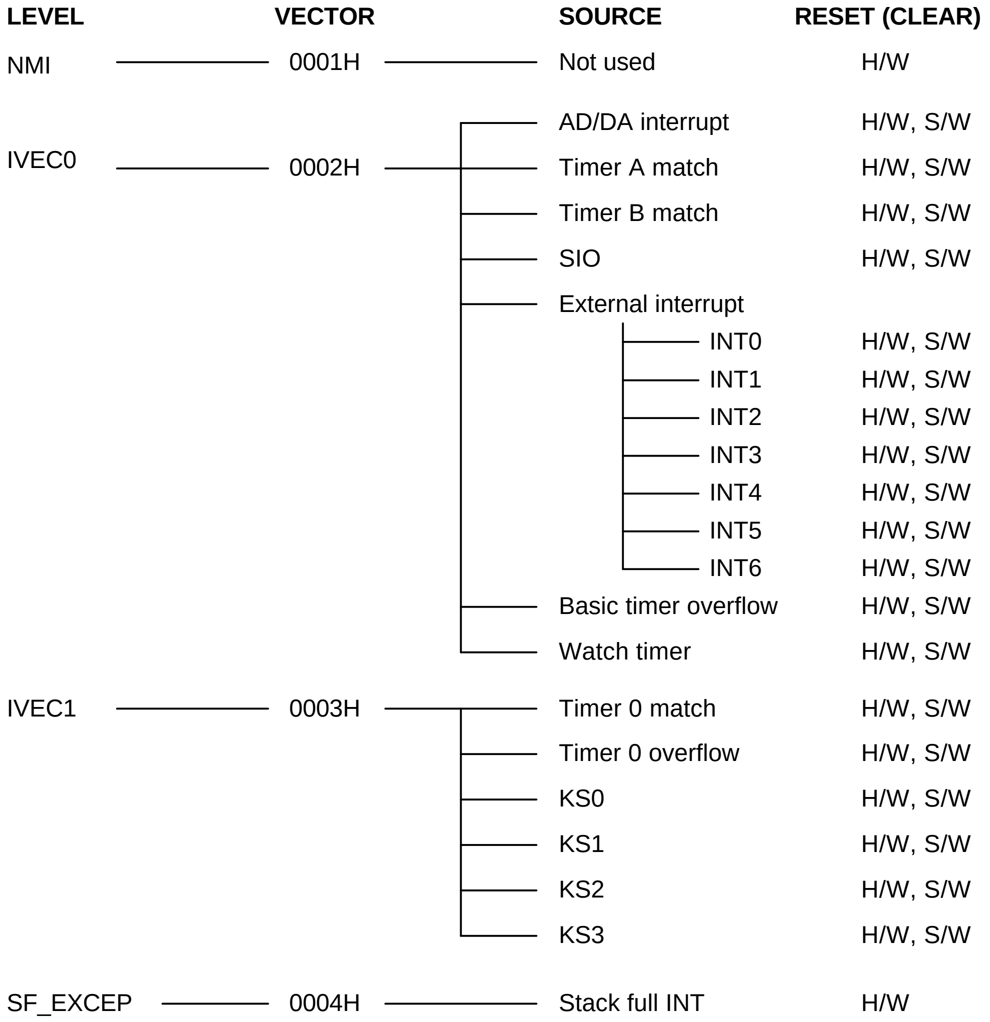
A Stack Full exception occurs when a stack operation is performed and as a result of the stack operation sptr[5] (SF) is set to 1. If the stack exception enable bit, exe (SR0[5]), is 1, the Stack Full exception is served. One exception to this rule is when nNMI causes a stack operation that sets sptr[5] (SF), since it has higher priority.

Handling a Stack Full exception may cause another Stack Full exception. In this case, the new exception is ignored. On a Stack Full exception, the CALL instruction is loaded in IR (Instruction Register) and 0h:0004h in PC. Therefore, when the Stack Full exception is activated, the “CALL {0h:PM[00004h]}” instruction is executed. When the exception is executed, SR0[1] (ie) is set to 0, and a core output signal *nEXPACK* is generated to acknowledge the exception.

BREAK EXCEPTION

Break exception is reserved only for an in-circuit debugger. When a core input signal, *BKREQ*, is high, the CalmRISC core is halted or in the break mode, until *BKREQ* is deactivated. Another way to drive the CalmRISC core into the break mode is by executing a break instruction, BREAK. When BREAK is fetched, it is decoded in the fetch cycle (IF stage) and the CalmRISC core output signal *nBKACK* is generated in the second cycle (ID/MEM stage). An in-circuit debugger generates *BKREQ* active by monitoring *nBKACK* to be active. BREAK instruction is exactly the same as the NOP (no operation) instruction except that it does not increase the program counter and activates nBKACK in the second cycle (or ID/MEM stage of the pipeline). There, once BREAK is encountered in the program execution, it falls into a deadlock. BREAK instruction is reserved for in-circuit debuggers only, so it should not be used in user programs.

EXCEPTIONS (or INTERRUPTS)

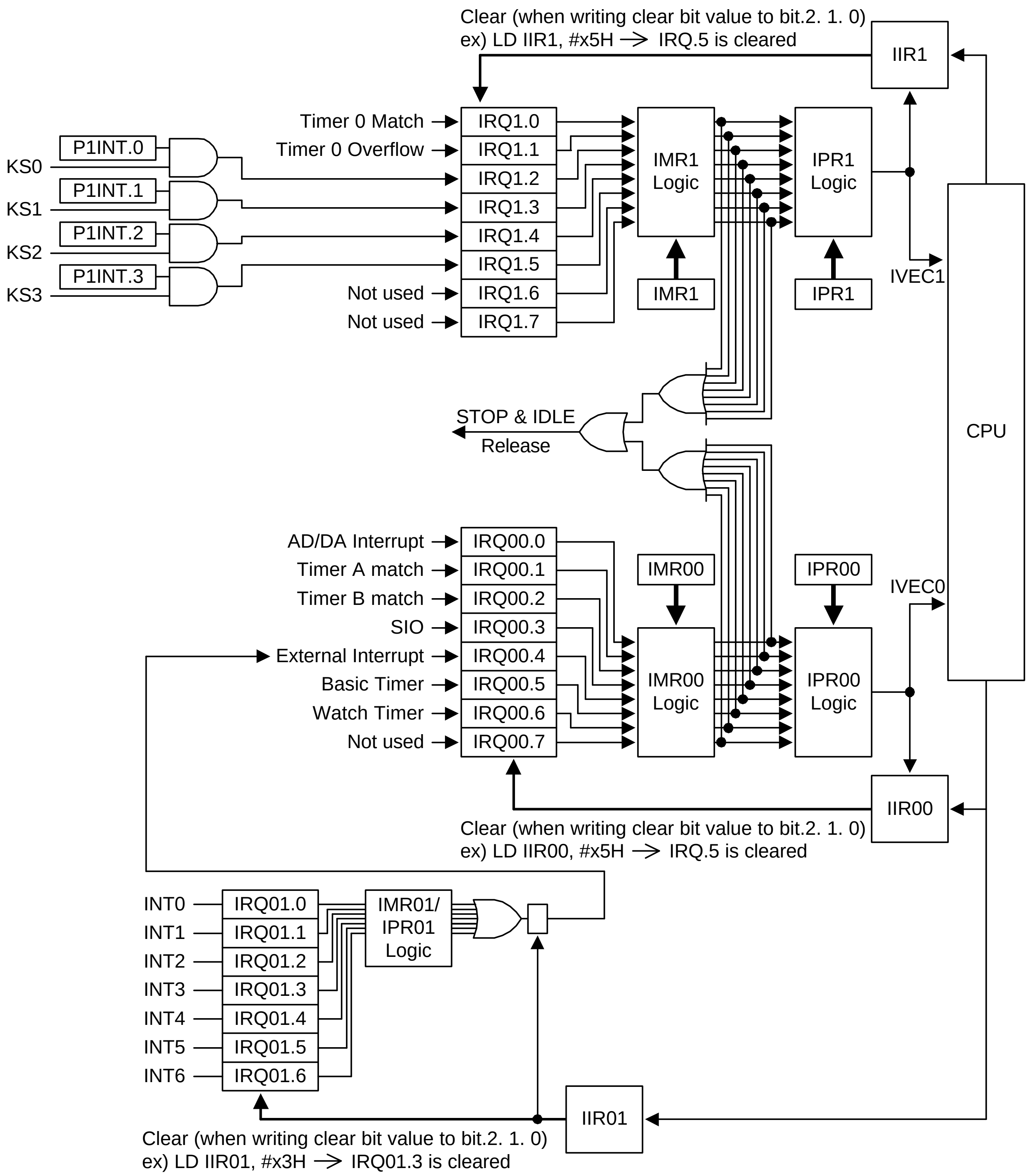


LEVEL	VECTOR	SOURCE		RESET (CLEAR)
NMI	0001H	Not used		H/W
		AD/DA interrupt		H/W, S/W
IVEC0	0002H	Timer A match		H/W, S/W
		Timer B match		H/W, S/W
		SIO		H/W, S/W
		External interrupt		
			INT0	H/W, S/W
			INT1	H/W, S/W
			INT2	H/W, S/W
			INT3	H/W, S/W
			INT4	H/W, S/W
			INT5	H/W, S/W
			INT6	H/W, S/W
		Basic timer overflow		H/W, S/W
		Watch timer		H/W, S/W
IVEC1	0003H	Timer 0 match		H/W, S/W
		Timer 0 overflow		H/W, S/W
		KS0		H/W, S/W
		KS1		H/W, S/W
		KS2		H/W, S/W
		KS3		H/W, S/W
SF_EXCEP	0004H	Stack full INT		H/W

NOTES:

1. There are two interrupt vectors, and the one interrupt vector have several interrupt sources. The priority of the sources is controlled by setting the IPR register.
2. IMR00, IPR00, IRQ00, is responsible for AD/DA interrupt, Timer A, Timer B, SIO, External, Basic timer, Watch timer interrupt. IMR01, IPR01, IRQ01 is responsible for INT0-INT6 interrupt. IMR1, IPR1, IRQ1 is responsible for Timer 0 Match, Timer 0 Overflow, Key scan interrupts.
3. External interrupts are triggered by a rising or falling edge, depending on the corresponding control register setting.
4. The NMI has the most higher priority in the interrupt levels. And the priority of SF_EXCEP is next but higher then IVEC0, IVEC1's priority is the last.
5. When system reset occurs, IPR register value is undefined. After reset, the interrupt priority can be changed by setting of IPR register.
6. The pending bit is cleared by Hardware when CPU reads the IIR register value.
7. If you write "LD IIRx, #8H", all bits of IRQx are cleared. (Where x is 1, 00, 01)

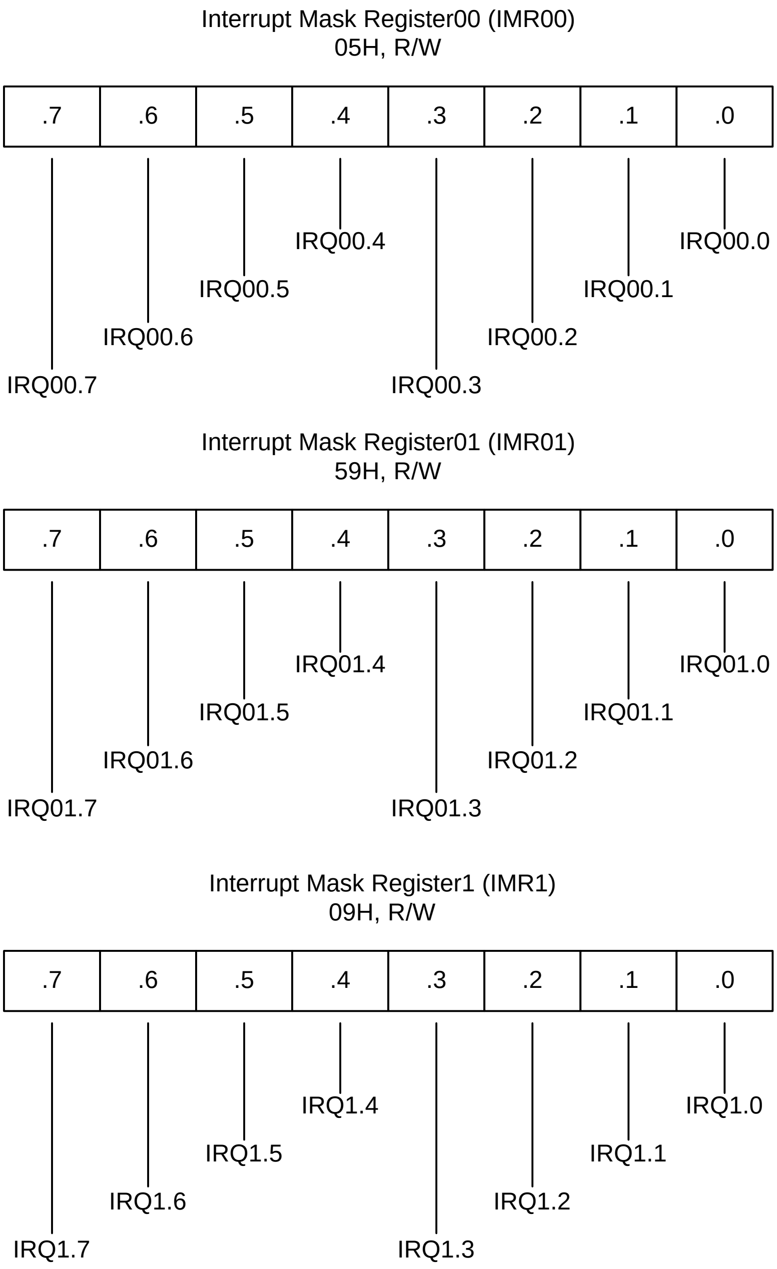
Figure 6-1. Interrupt Structure



NOTE: If you write "LD" IIRx, #8H", all bits of IRQx are cleared. (Where x is 1, 00, 01)

Figure 6-2. Interrupt Structure

INTERRUPT MASK REGISTERS

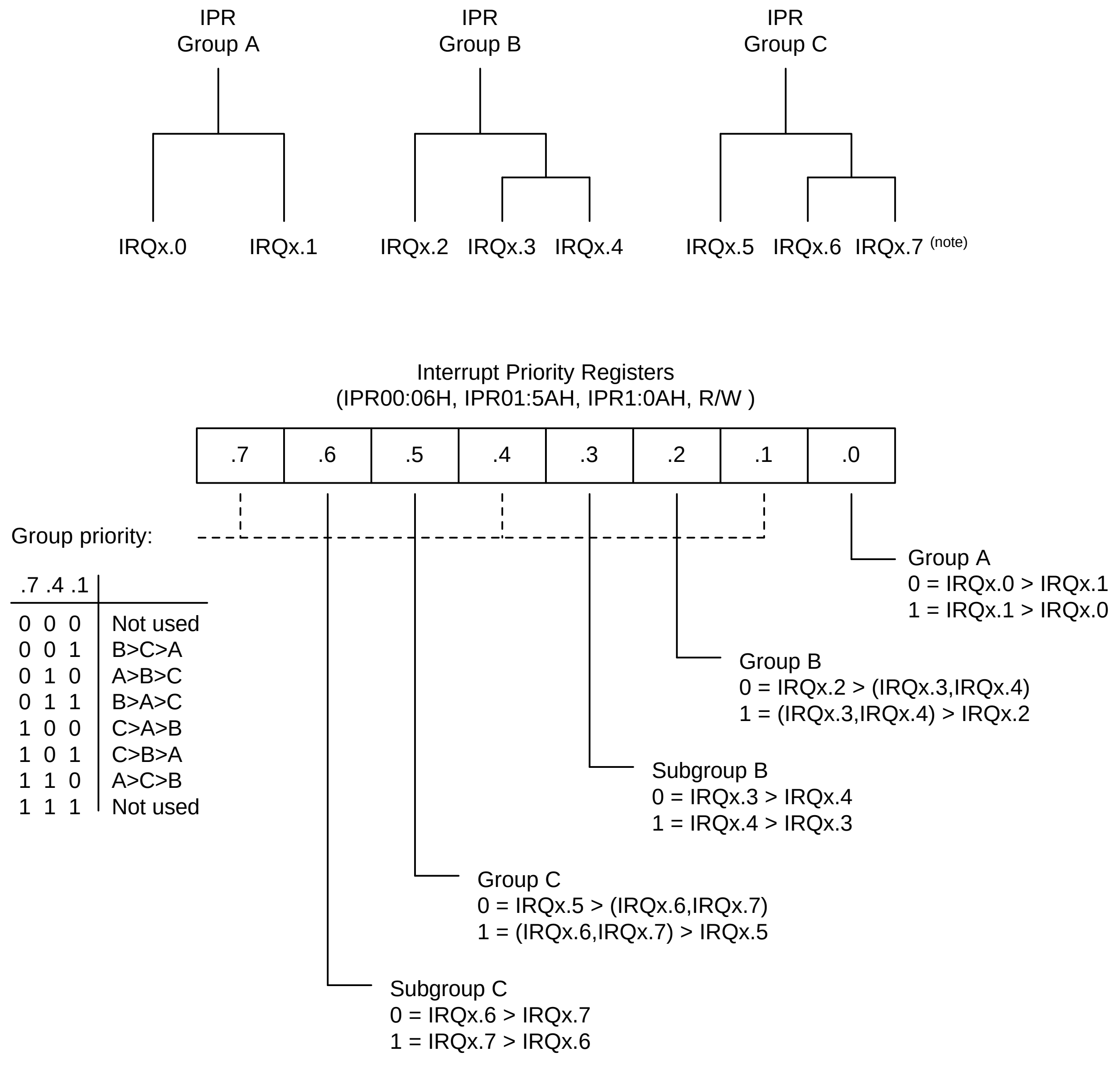


Interrupt request enable bits:
0 = Disable interrupt request
1 = Enable interrupt request

NOTE: If you want to change the value of the IMR register, then you first make disable global INT by DI instruction, and change the value of the IMR register.

Figure 6-3. Interrupt Mask Register

INTERRUPT PRIORITY REGISTER



.7 .4 .1	
0 0 0	Not used
0 0 1	B>C>A
0 1 0	A>B>C
0 1 1	B>A>C
1 0 0	C>A>B
1 0 1	C>B>A
1 1 0	A>C>B
1 1 1	Not used

Group A
0 = IRQx.0 > IRQx.1
1 = IRQx.1 > IRQx.0

Group B
0 = IRQx.2 > (IRQx.3,IRQx.4)
1 = (IRQx.3,IRQx.4) > IRQx.2

Subgroup B
0 = IRQx.3 > IRQx.4
1 = IRQx.4 > IRQx.3

Group C
0 = IRQx.5 > (IRQx.6,IRQx.7)
1 = (IRQx.6,IRQx.7) > IRQx.5

Subgroup C
0 = IRQx.6 > IRQx.7
1 = IRQx.7 > IRQx.6

NOTE: If you want to change the value of the IPR register, then you first make disable global INT by DI instruction, and change the value of the IPR register.

Figure 6-4. Interrupt Priority Register

☞ PROGRAMMING TIP — Interrupt Programming Tip 1

Jumped from vector 2

```
            PUSH        SR1
            PUSH        R0
            LD          R0, IIR00
            CP          R0, #03h
            JR          ULE, LTE03
            CP          R0, #05h
            JR          ULE, LTE05
            CP          R0, #06h
            JP          EQ, IRQ6_srv
            JP          T, IRQ7_srv
LTE05       CP          R0, #04
            JP          EQ, IRQ4_srv
            JP          T, IRQ5_srv
LTE03       CP          R0, #01
            JR          ULE, LTE01
            CP          R0, #02
            JP          EQ, IRQ2_srv
            JP          T, IRQ3_srv
LTE01       CP          R0, #00h
            JP          EQ, IRQ0_srv
            JP          T, IRQ1_srv
IRQ0_srv                                        ; →   service for IRQ0
            •
            POP         R0
            POP         SR1
            IRET
IRQ1_srv                                        ; →   service for IRQ1
            •
            •
            POP         R0
            POP         SR1
            IRET
            •
            •
IRQ7_srv                                        ; →   service for IRQ7
            •
            •
            POP         R0
            POP         SR1
            IRET
```

NOTE: If the SR0 register is changed in the interrupt service routine, then the SR0 register must be pushed and popped in the interrupt service routine.

☞ PROGRAMMING TIP — Interrupt Programming Tip 2

Jumped from vector 2

```
          PUSH      SR1
          PUSH      R0
          PUSH      R1
          LD        R0, IIR00
          SL        R0
          LD        R1, < TBL_INTx
          ADD       R0, > TBL_INTx
          PUSH      R1
          PUSH      R0
          RET
TBL_INTx  LJP       IRQ0_svr
          LJP       IRQ1_svr
          LJP       IRQ2_svr
          LJP       IRQ3_svr
          LJP       IRQ4_svr
          LJP       IRQ5_svr
          LJP       IRQ6_svr
          LJP       IRQ7_svr
IRQ0_srv                                ; →   service for IRQ0
          •
          •
          POP       R1
          POP       R0
          POP       SR1
          IRET
IRQ1_srv                                ; →   service for IRQ1
          •
          •
          POP       R1
          POP       R0
          POP       SR1
          IRET
          •
          •
IRQ7_srv                                ; →   service for IRQ7
          •
          •
          POP       R1
          POP       R0
          POP       SR1
          IRET
```

NOTES:

1. If the SR0 register is changed in the interrupt service routine, then the SR0 register must be pushed and popped in the interrupt service routine.
2. Above example is assumed that ROM size is less than 64K-word and all the LJP instructions in the jump table (TBL_INTx) is in the same page.

7 COPROCESSOR INTERFACE

OVERVIEW

CalmRISC supports an efficient and seamless interface with coprocessors. By integrating a MAC (multiply and accumulate) DSP coprocessor engine with the CalmRISC core, not only the microcontroller functions but also complex digital signal processing algorithms can be implemented in a single development platform (or MDS). CalmRISC has a set of dedicated signal pins, through which data/command/status are exchanged to and from a coprocessor. Depicted below are the coprocessor signal pins and the interface between two processors.

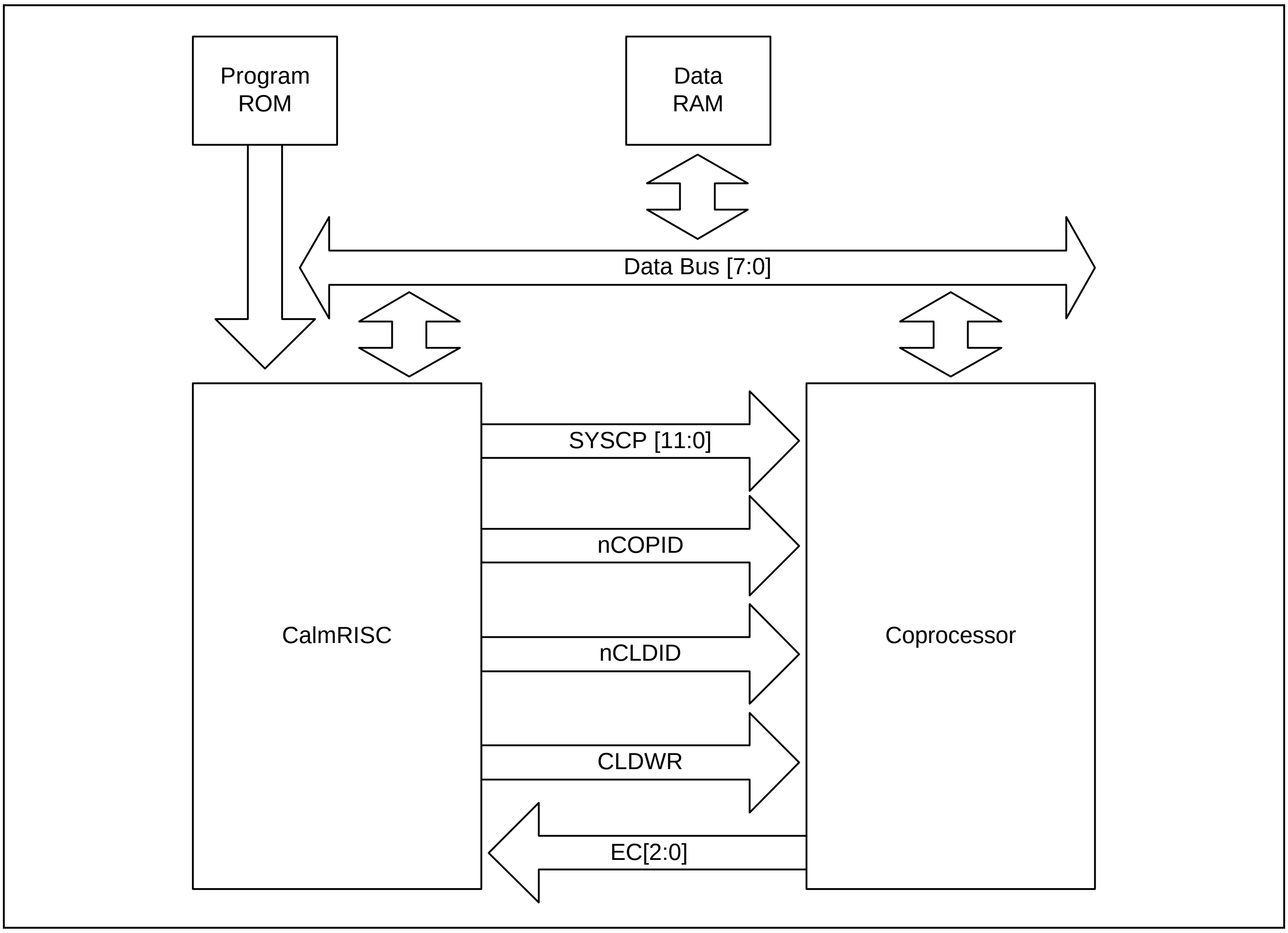


Figure 7-1. Coprocessor Interface Diagram

As shown in the coprocessor interface diagram above, the coprocessor interface signals of CalmRISC are: *SYSCP[11:0]*, *nCOPID*, *nCLDID*, *nCLDWR*, and *EC[2:0]*. The data are exchanged through data buses, *DI[7:0]* and *DO[7:0]*. A command is issued from CalmRISC to a coprocessor through *SYSCP[11:0]* in COP instructions. The status of a coprocessor can be sent back to CalmRISC through EC[2:0] and these flags can be checked in the condition codes of branch instructions. The coprocessor instructions are listed in the following table

Table 7-1. Coprocessor instructions

Mnemonic	Op 1	Op 2	Description
COP	#imm:12	–	Coprocessor operation
CLD	GPR	imm:8	Data transfer from coprocessor into GPR
CLD	imm:8	GPR	Data transfer of GPR to coprocessor
JP(or JR) CALL LNK	EC2–EC0	label	Conditional branch with coprocessor status flags

The coprocessor of CalmRISC does not have its own program memory (i.e., it is a passive coprocessor) as shown in Figure 7 -1. In fact, the coprocessor instructions are fetched and decoded by CalmRISC, and CalmRISC issues the command to the coprocessor through the interface signals. For example, if “COP #imm:12” instruction is fetched, then the 12-bit immediate value (imm:12) is loaded on *SYSCP[11:0]* signal with *nCOPID* active in ID/MEM stage, to request the coprocessor to perform the designated operation. The interpretation of the 12-bit immediate value is totally up to the coprocessor. By arranging the 12-bit immediate field, the instruction set of the coprocessor is determined. In other words, CalmRISC only provides a set of generic coprocessor instructions, and its installation to a specific coprocessor instruction set can differ from one coprocessor to another. CLD Write instructions (“CLD imm:8, GPR”) put the content of a GPR register of CalmRISC on the data bus (*DO[7:0]*) and issue the address(imm:8) of the coprocessor internal register on *SYSCP[7:0]* with *nCLDID* active and *CLDWR* active. CLD Read instructions (“CLD GPR, imm:8” in Table 7-1) work similarly, except that the content of the coprocessor internal register addressed by the 8-bit immediate value is read into a GPR register through *DI[7:0]* with *nCLDID* active and *CLDWR* deactivated.

The timing diagram given below is a coprocessor instruction pipeline and shows when the coprocessor performs the required operations. Suppose I_2 is a coprocessor instruction. First, it is fetched and decoded by CalmRISC (at t = T(i-1)). Once it is identified as a coprocessor instruction, CalmRISC indicates to the coprocessor the appropriate command through the coprocessor interface signals (at t = T(i)). Then the coprocessor performs the designated tasks at t = T(i) and t = T(i+1). Hence IF from CalmRISC and then ID/MEM and EX from the coprocessor constitute the pipeline for I_2. Similarly, if I_3 is a coprocessor instruction, the coprocessor’s ID/MEM and EX stages replace the corresponding stages of CalmRISC.

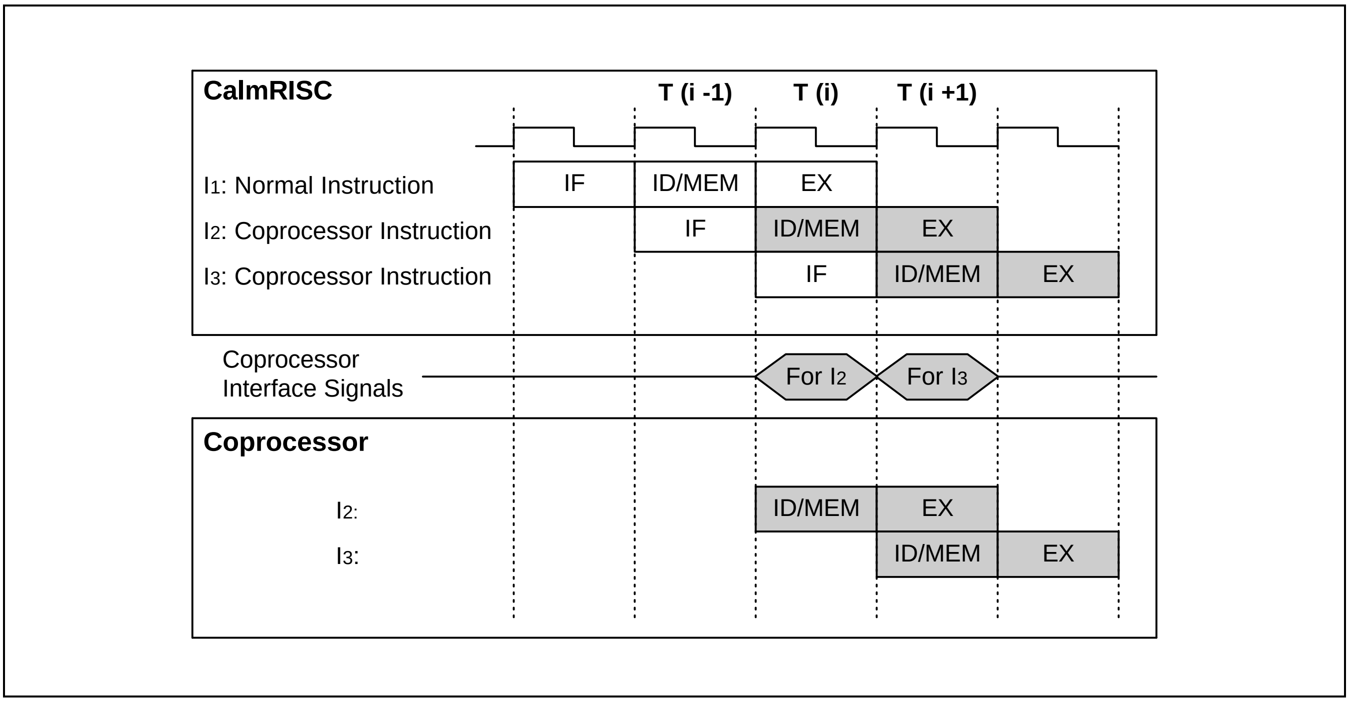


Figure 7-2. Coprocessor Instruction Pipeline

In a multi-processor system, the data transfer between processors is an important factor to determine the efficiency of the overall system. Suppose an input data stream is accepted by a processor, in order for the data to be shared by another processors. There should be some efficient mechanism to transfer the data to the processors. In CalmRISC, data transfers are accomplished through a single shared data memory. The shared data memory in a multi-processor has some inherent problems such as data hazards and deadlocks. However, the coprocessor in CalmRISC accesses the shared data memory only at the designated time by CalmRISC at which time CalmRISC is guaranteed not to access the data memory, and therefore there is no contention over the shared data memory. Another advantage of the scheme is that the coprocessor can access the data memory in its own bandwidth.

NOTES

8 INSTRUCTION SET

OVERVIEW

GLOSSARY

This chapter describes the CalmRISC instruction set and the details of each instruction are listed in alphabetical order. The following notations are used for the description.

Table 8-1. Instruction Notation Conventions

Notation	Interpretation
<opN>	Operand N. N can be omitted if there is only one operand. Typically, <op1> is the destination (and source) operand and <op2> is a source operand.
GPR	General Purpose Register
SPR	Special Purpose Register (IDL0, IDL1, IDH, SR0, ILX, ILH, ILL, SR1)
adr:N	N-bit address specifier
@idm	Content of memory location pointed by ID0 or ID1
(adr:N)	Content of memory location specified by adr:N
cc:4	4-bit condition code. Table 8-6 describes cc:4.
imm:N	N-bit immediate number
&	Bit-wise AND
|	Bit-wise OR
~	Bit-wise NOT
^	Bit-wise XOR
N**M	Mth power of N
$(N)_M$	M-based number N

As additional note, only the affected flags are described in the tables in this section. That is, if a flag is not affected by an operation, it is NOT specified.

INSTRUCTION SET MAP

Table 8-2.Overall Instruction Set Map

IR	[12:10]000	001	010	011	100	101	110	111
[15:13,7:2] 000 xxxxxx	ADD GPR, #imm:8	SUB GPR, #imm:8	CP GPR, #imm8	LD GPR, #imm:8	TM GPR, #imm:8	AND GPR, #imm:8	OR GPR, #imm:8	XOR GPR, #imm:8
001 xxxxxx	ADD GPR, @idm	SUB GPR, @idm	CP GPR, @idm	LD GPR, @idm	LD @idm, GPR	AND GPR, @idm	OR GPR, @idm	XOR GPR, @idm
010 xxxxxx	ADD GPR, adr:8	SUB GPR, adr:8	CP GPR, adr:8	LD GPR, adr:8	BITT adr:8.bs		BITS adr:8.bs	
011 xxxxxx	ADC GPR, adr:8	SBC GPR, adr:8	CPC GPR, adr:8	LD adr:8, GPR	BITR adr:8.bs		BITC adr:8.bs	
100 000000	ADD GPR, GPR	SUB GPR, GPR	CP GPR, GPR	BMS/BMC	LD SPR0, #imm:8	AND GPR, adr:8	OR GPR, adr:8	XOR GPR, adr:8
100 000001	ADC GPR, GPR	SBC GPR, GPR	CPC GPR, GPR	*invalid*				
100 000010	*invalid*	*invalid*	*invalid*	*invalid*				
100 000011	AND GPR, GPR	OR GPR, GPR	XOR GPR, GPR	*invalid*				
100 00010x	SLA/SL/RLC/RL/SRA/SR/RRC/RR/GPR	INC/INCC/DEC/DECC/COM/COM2/COMC GPR	*invalid*	*invalid*				
100 00011x	LD SPR, GPR	LD GPR, SPR	SWAP GPR, SPR	LD TBH/TBL, GPR				
100 00100x	PUSH SPR	POP SPR	*invalid*	*invalid*				
100 001010	PUSH GPR	POP GPR	LD GPR, GPR	LD GPR, TBH/TBL				

Table 8-2. Overall Instruction Set Map (Continued)

IR	[12:10]000	001	010	011	100	101	110	111
100 001011	POP	*invalid*	LDC	*invalid*	LD SPR0, #imm:8	AND GPR, adr:8	OR GPR, adr:8	XOR GPR, adr:8
100 00110x	RET/LRET/ IRET/NOP/ BREAK	*invalid*	*invalid*	*invalid*				
100 00111x	*invalid*	*invalid*	*invalid*	*invalid*				
100 01xxxx	LD GPR:bank, GPR:bank	AND SR0, #imm:8	OR SR0, #imm:8	BANK #imm:2				
100 100000 100 110011	*invalid*	*invalid*	*invalid*	*invalid*				
100 1101xx	LCALL cc:4, imm:20 (2-word instruction)							
100 1110xx	LLNK cc:4, imm:20 (2-word instruction)							
100 1111xx	LJP cc:4, imm:20 (2-word instruction)							
[15:10] 101 xxx	JR cc:4, imm:9							
110 0xx	CALLS imm:12							
110 1xx	LNKS imm:12							
111 xxx	CLD GPR, imm:8 / CLD imm:8, GPR / JNZD GPR, imm:8 / SYS #imm:8 / COP #imm:12							

NOTE: “*invalid*” - invalid instruction.

Table 8-3. Instruction Encoding

Instruction	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD GPR, #imm:8	000			000			GPR		imm[7:0]							
SUB GPR, #imm:8				001												
CP GPR, #imm:8				010												
LD GPR, #imm:8				011												
TM GPR, #imm:8				100												
AND GPR, #imm:8				101												
OR GPR, #imm:8				110												
XOR GPR, #imm:8				111												
ADD GPR, @idm	001			000			GPR		idx	mod		offset[4:0]				
SUB GPR, @idm				001												
CP GPR, @idm				010												
LD GPR, @idm				011												
LD @idm, GPR				100												
AND GPR, @idm				101												
OR GPR, @idm				110												
XOR GPR, @idm				111												
ADD GPR, adr:8	010			000			GPR		adr[7:0]							
SUB GPR, adr:8				001												
CP GPR, adr:8				010												
LD GPR, adr:8				011												
BITT adr:8.bs				10		bs										
BITS adr:8.bs				11												
ADC GPR, adr:8	011			000			GPR		adr[7:0]							
SBC GPR, adr:8				001												
CPC GPR, adr:8				010												
LD adr:8, GPR				011												
BITR adr:8.bs				10		bs										
BITC adr:8.bs				11												

Table 8-3. Instruction Encoding (Continued)

Instruction	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD GPRd, GPRs	100			000			GPRd		000000						GPRs	
SUB GPRd, GPRs				001												
CP GPRd, GPRs				010												
BMS/BMC				011												
ADC GPRd, GPRs				000					000001							
SBC GPRd, GPRs				001												
CPC GPRd, GPRs				010												
invalid				011												
invalid				ddd					000010							
AND GPRd, GPRs				000					000011							
OR GPRd, GPRs				001												
XOR GPRd, GPRs				010												
invalid				011												
ALUop1				000			GPR		00010					ALUop1		
ALUop2				001			GPR							ALUop2		
invalid				010–011			xx							xxx		
LD SPR, GPR				000			GPR		00011					SPR		
LD GPR, SPR				001			GPR							SPR		
SWAP GPR, SPR				010			GPR							SPR		
LD TBL, GPR				011			GPR							x	0	x
LD TBH, GPR														x	1	x
PUSH SPR				000			xx		00100					SPR		
POP SPR				001			xx							SPR		
invalid				010–011			xx							xxx		
PUSH GPR				000			GPR		001010						GPR	
POP GPR				001			GPR								GPR	
LD GPRd, GPRs				010			GPRd								GPRs	
LD GPR, TBL				011			GPR								0	x
LD GPR, TBH															1	x
POP				000			xx		001011						xx	
LDC @IL				010											0	x
LDC @IL+															1	x
Invalid				001, 011											xx	

NOTE: "x" means not applicable.

Table 8-3. Instruction Encoding (Concluded)

Instruction	15-13	12	11	10	9	8	7	6	5	4	3	2	1	0	2nd word
MODop1	100	000			xx		00110					MODop1			–
Invalid		001–011			xx							xxx			
Invalid		000			xx		01		xxxxxx						
AND SR0, #imm:8		001			imm[7:6]				imm[5:0]						
OR SR0, #imm:8		010			imm[7:6]										
BANK #imm:2		011			xx				x	imm [1:0]		xxx			
Invalid		0	xxxx				10000000-11001111								
LCALL cc, imm:20			cc				1101				imm[19:16]				imm[15:0]
LLNK cc, imm:20															
LJP cc, imm:20															
LD SPR0, #imm:8		1	00		SPR0		IMM[7:0]								–
AND GPR, adr:8			01		GPR		ADR[7:0]								
OR GPR, adr:8			10												
XOR GPR, adr:8			11												
JR cc, imm:9	101	imm [8]	cc				imm[7:0]								
CALLS imm:12	110	0	imm[11:0]												
LNKS imm:12		1													
CLD GPR, imm:8	111	0	00		GPR		imm[7:0]								
CLD imm:8, GPR			01		GPR										
JNZD GPR, imm:8			10		GPR										
SYS #imm:8			11		xx										
COP #imm:12		1	imm[11:0]												

NOTES:

1. "x" means not applicable.
2. There are several MODop1 codes that can be used, as described in table 8-9.
3. The operand 1(GPR) of the instruction JNZD is Bank 3's register.

Table 8-4. Index Code Information (“idx”)

Symbol	Code	Description
ID0	0	Index 0 IDH:IDL0
ID1	1	Index 1 IDH:IDL1

Table 8-5. Index Modification Code Information (“mod”)

Symbol	Code	Function
@IDx + offset:5	00	DM[IDx], IDx ← IDx + offset
@[IDx - offset:5]	01	DM[IDx + (2’s complement of offset:5)], IDx ← IDx + (2’s complement of offset:5)
@[IDx + offset:5]!	10	DM[IDx + offset], IDx ← IDx
@[IDx - offset:5]!	11	DM[IDx + (2’s complement of offset:5)], IDx ← IDx

NOTE: Carry from IDL is propagated to IDH. In case of @[IDx - offset:5] or @[IDx - offset:5]!, the assembler should convert offset:5 to the 2’s complement format to fill the operand field (offset[4:0]).
Furthermore, @[IDx - 0] and @[IDx - 0]! are converted to @[IDx + 0] and @[IDx + 0]!, respectively.

Table 8-6. Condition Code Information (“cc”)

Symbol (cc:4)	Code	Function
Blank	0000	always
NC or ULT	0001	C = 0, unsigned less than
C or UGE	0010	C = 1, unsigned greater than or equal to
Z or EQ	0011	Z = 1, equal to
NZ or NE	0100	Z = 0, not equal to
OV	0101	V = 1, overflow - signed value
ULE	0110	~C | Z, unsigned less than or equal to
UGT	0111	C & ~Z, unsigned greater than
ZP	1000	N = 0, signed zero or positive
MI	1001	N = 1, signed negative
PL	1010	~N & ~Z, signed positive
ZN	1011	Z | N, signed zero or negative
SF	1100	Stack Full
EC0-EC2	1101-1111	EC[0] = 1/EC[1] = 1/EC[2] = 1

NOTE: EC[2:0] is an external input (CalmRISC core’s point of view) and used as a condition.

Table 8-7. “ALUop1” Code Information

Symbol	Code	Function
SLA	000	arithmetic shift left
SL	001	shift left
RLC	010	rotate left with carry
RL	011	rotate left
SRA	100	arithmetic shift right
SR	101	shift right
RRC	110	rotate right with carry
RR	111	rotate right

Table 8-8. “ALUop2” Code Information

Symbol	Code	Function
INC	000	increment
INCC	001	increment with carry
DEC	010	decrement
DECC	011	decrement with carry
COM	100	1’s complement
COM2	101	2’s complement
COMC	110	1’s complement with carry
–	111	reserved

Table 8-9. “MODop1” Code Information

Symbol	Code	Function
LRET	000	return by IL
RET	001	return by HS
IRET	010	return from interrupt (by HS)
NOP	011	no operation
BREAK	100	reserved for debugger use only
–	101	reserved
–	110	reserved
–	111	reserved

QUICK REFERENCE

Operation	op1	op2	Function	Flag	# of word / cycle
AND	GPR	adr:8	op1 ← op1 & op2	z,n	1W1C
OR		#imm:8	op1 ← op1 | op2	z,n	
XOR		GPR	op1 ← op1 ^ op2	z,n	
ADD		@idm	op1 ← op1 + op2	c,z,v,n	
SUB			op1 ← op1 + ~op2 + 1	c,z,v,n	
CP			op1 + ~op2 + 1	c,z,v,n	
ADC	GPR	GPR	op1 ← op1 + op2 + c	c,z,v,n	
SBC		adr:8	op1 ← op1 + ~op2 + c	c,z,v,n	
CPC			op1 + ~op2 + c	c,z,v,n	
TM	GPR	#imm:8	op1 & op2	z,n	
BITS	R3	adr:8.bs	op1 ← (op2[bit] ← 1)	z	
BITR			op1 ← (op2[bit] ← 0)	z	
BITC			op1 ← ~(op2[bit])	z	
BITT			z ← ~(op2[bit])	z	
BMS/BMC	–	–	TF ← 1 / 0	–	
PUSH	GPR	–	HS[sptr] ← GPR, (sptr ← sptr + 1)	–	
POP			GPR ← HS[sptr - 1], (sptr ← sptr - 1)	z,n	
PUSH	SPR	–	HS[sptr] ← SPR, (sptr ← sptr + 1)	–	
POP			SPR ← HS[sptr - 1], (sptr ← sptr - 1)		
POP	–	–	sptr ← sptr – 2	–	
SLA	GPR	–	c ← op1[7], op1 ← {op1[6:0], 0}	c,z,v,n	
SL			c ← op1[7], op1 ← {op1[6:0], 0}	c,z,n	
RLC			c ← op1[7], op1 ← {op1[6:0], c}	c,z,n	
RL			c ← op[7], op1 ← {op1[6:0], op1[7]}	c,z,n	
SRA			c ← op[0], op1 ← {op1[7],op1[7:1]}	c,z,n	
SR			c ← op1[0], op1 ← {0, op1[7:1]}	c,z,n	
RRC			c ← op1[0], op1 ← {c, op1[7:1]}	c,z,n	
RR			c ← op1[0], op1 ← {op1[0], op1[7:1]}	c,z,n	
INC			op1 ← op1 + 1	c,z,v,n	
INCC			op1 ← op1 + c	c,z,v,n	
DEC			op1 ← op1 + 0FFh	c,z,v,n	
DECC			op1 ← op1 + 0FFh + c	c,z,v,n	
COM			op1 ← ~op1	z,n	
COM2			op1 ← ~op1 + 1	c,z,v,n	
COMC			op1 ← ~op1 + c	c,z,v,n	

QUICK REFERENCE (Continued)

Operation	op1	op2	Function	Flag	# of word / cycle
LD	GPR :bank	GPR :bank	op1 ← op2	z,n	1W1C
LD	SPR0	#imm:8	op1 ← op2	–	
LD	GPR	GPR SPR adr:8 @idm #imm:8 TBH/TBL	op1 ← op2	z,n	
LD	SPR TBH/TBL	GPR	op1 ← op2	–	
LD	adr:8	GPR	op1 ← op2	–	
LD	@idm	GPR	op1 ← op2	–	
LDC	@IL @IL+	–	(TBH:TBL) ← PM[(ILX:ILH:ILL)], ILL++ if @IL+	–	1W2C
AND OR	SR0	#imm:8	SR0 ← SR0 & op2 SR0 ← SR0 | op2	–	1W1C
BANK	#imm:2	–	SR0[4:3] ← op2	–	
SWAP	GPR	SPR	op1 ← op2, op2 ← op1 (excluding SR0/SR1)	–	
LCALL cc	imm:20	–	If branch taken, push XSTACK, HS[15:0] ← {PC[15:12],PC[11:0] + 2} and PC ← op1 else PC[11:0] ← PC[11:0] + 2	–	2W2C
LLNK cc	imm:20	–	If branch taken, IL[19:0] ← {PC[19:12], PC[11:0] + 2} and PC ← op1 else PC[11:0] ← PC[11:0] + 2	–	
CALLS	imm:12	–	push XSTACK, HS[15:0] ← {PC[15:12], PC[11:0] + 1} and PC[11:0] ← op1	–	1W2C
LNKS	imm:12	–	IL[19:0] ← {PC[19:12], PC[11:0] + 1} and PC[11:0] ← op1	–	
JNZD	Rn	imm:8	if (Rn == 0) PC ← PC[delay slot] - 2's complement of imm:8, Rn-- else PC ← PC[delay slot]++, Rn--	–	
LJP cc	imm:20	–	If branch taken, PC ← op1 else PC[11:0] < PC[11:0] + 2	–	2W2C
JR cc	imm:9	–	If branch taken, PC[11:0] ← PC[11:0] + op1 else PC[11:0] ← PC[11:0] + 1	–	1W2C

NOTE: op1 - operand1, op2 - operand2, 1W1C - 1-Word 1-Cycle instruction, 1W2C - 1-Word 2-Cycle instruction, 2W2C - 2-Word 2-Cycle instruction. The Rn of instruction JNZD is Bank 3's GPR.

QUICK REFERENCE (Concluded)

Operation	op1	op2	Function	Flag	# of word / cycle
LRET	–	–	PC ← IL[19:0]	–	1W2C
RET			PC ← HS[sptr - 2], (sptr ← sptr - 2)		1W2C
IRET			PC ← HS[sptr - 2], (sptr ← sptr - 2)		1W2C
NOP			no operation		1W1C
BREAK			no operation and hold PC		1W1C
SYS	#imm:8	–	no operation but generates SYSCP[7:0] and nSYSID	–	1W1C
CLD	imm:8	GPR	op1 ← op2, generates SYSCP[7:0], nCLDID, and CLDWR	–	
CLD	GPR	imm:8	op1 ← op2, generates SYSCP[7:0], nCLDID, and CLDWR	z,n	
COP	#imm:12	–	generates SYSCP[11:0] and nCOPID	–	

NOTES:

1. op1 - operand1, op2 - operand2, sptr - stack pointer register, 1W1C - 1-Word 1-Cycle instruction, 1W2C - 1-Word 2-Cycle instruction
2. Pseudo instructions
   - SCF/RCF
     Carry flag set or reset instruction
   - STOP/IDLE
     MCU power saving instructions
   - EI/DI
     Exception enable and disable instructions
   - JP/LNK/CALL
     If JR/LNKS/CALLS commands (1 word instructions) can access the target address, there is no conditional code in the case of CALL/LNK, and the JP/LNK/CALL commands are assembled to JR/LNKS/CALLS in linking time, or else the JP/LNK/CALL commands are assembled to LJP/LLNK/LCALL (2 word instructions) instructions.

INSTRUCTION GROUP SUMMARY

ALU INSTRUCTIONS

“ALU instructions” refer to the operations that use ALU to generate results. ALU instructions update the values in Status Register 1 (SR1), namely carry (C), zero (Z), overflow (V), and negative (N), depending on the operation type and the result.

ALUop GPR, adr:8

Performs an ALU operation on the value in GPR and the value in DM[adr:8] and stores the result into GPR.
ALUop = ADD, SUB, CP, AND, OR, XOR
For SUB and CP, GPR+(not DM[adr:8])+1 is performed.
adr:8 is the offset in a specific data memory page.

The data memory page is 0 or the value of IDH (Index of Data Memory Higher Byte Register), depending on the value of eid in Status Register 0 (SR0).

Operation

GPR ← GPR ALUop DM[00h:adr:8] if eid = 0
GPR ← GPR ALUop DM[IDH:adr8] if eid = 1
Note that this is an 8-bit operation.

Example

```
ADD R0, 80h          // Assume eid = 1 and IDH = 01H
                     // R0 ← R0 + DM[0180h]
```

ALUop GPR, #imm:8

Stores the result of an ALU operation on GPR and an 8-bit immediate value into GPR.
ALUop = ADD, SUB, CP, AND, OR, XOR
For SUB and CP, GPR+(not #imm:8)+1 is performed.
#imm:8 is an 8-bit immediate value.

Operation

GPR ← GPR ALUop #imm:8

Example

```
ADD R0, #7Ah         // R0 ← R0 + 7Ah
```

ALUop GPRd, GPRs

Store the result of ALUop on GPRs and GPRd into GPRd.
ALUop = ADD, SUB, CP, AND, OR, XOR
For SUB and CP, GPRd + (not GPRs) + 1 is performed.
GPRs and GPRd need not be distinct.

Operation

```
GPRd ← GPRd ALUop GPRs
GPRd - GPRs when ALUop = CP (comparison only)
```

Example

```
ADD R0, R1              // R0 ← R0 + R1
```

ALUop GPR, @idm

Performs ALUop on the value in GPR and DM[ID] and stores the result into GPR. Index register ID is IDH:IDL (IDH:IDL0 or IDH:IDL1).
ALUop = ADD, SUB, CP, AND, OR, XOR
For SUB and CP, GPR+(not DM[idm])+1 is performed.
idm = IDx+off:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]!
(IDx = ID0 or ID1)

Operation

```
GPR - DM[idm] when ALUop = CP (comparison only)
GPR ← GPR ALUop DM[IDx], IDx ← IDx + offset:5 when idm = IDx + offset:5
GPR ← GPR ALUop DM[IDx - offset:5], IDx ← IDx - offset:5 when idm = [IDx - offset:5]
GPR ← GPR ALUop DM[IDx + offset:5] when idm = [IDx + offset:5]!
GPR ← GPR ALUop DM[IDx - offset:5] when idm = [IDx - offset:5]!
```

When carry is generated from IDL (on a post-increment or pre-decrement), it is propagated to IDH.

Example

```
ADD R0, @ID0+2          // assume ID0 = 02FFh
                        // R0 ← R0 + DM[02FFh], IDH ← 03h and IDL0 ← 01h
ADD R0, @[ID0-2]        // assume ID0 = 0201h
                        // R0 ← R0 + DM[01FFh], IDH ← 01h and IDL0 ← FFh
ADD R0, @[ID1+2]!       // assume ID1 = 02FFh
                        // R0 ← R0 + DM[0301], IDH ← 02h and IDL1 ← FFh
ADD R0, @[ID1-2]!       // assume ID1 = 0200h
                        // R0 ← R0 + DM[01FEh], IDH ← 02h and IDL1 ← 00h
```

ALUopc GPRd, GPRs

Performs ALUop with carry on GPRd and GPRs and stores the result into GPRd.
ALUopc = ADC, SBC, CPC
GPRd and GPRs need not be distinct.

Operation

GPRd ← GPRd + GPRs + C when ALUopc = ADC
GPRd ← GPRd + (not GPRs) + C when ALUopc = SBC
GPRd + (not GPRs) + C when ALUopc = CPC (comparison only)

Example

```
ADD R0, R2        // assume R1:R0 and R3:R2 are 16-bit signed or unsigned numbers.
ADC R1, R3        // to add two 16-bit numbers, use ADD and ADC.

SUB R0, R2        // assume R1:R0 and R3:R2 are 16-bit signed or unsigned numbers.
SBC R1, R3        // to subtract two 16-bit numbers, use SUB and SBC.

CP R0, R2         // assume both R1:R0 and R3:R2 are 16-bit unsigned numbers.
CPC R1, R3        // to compare two 16-bit unsigned numbers, use CP and CPC.
```

ALUopc GPR, adr:8

Performs ALUop with carry on GPR and DM[adr:8].

Operation

GPR ← GPR + DM[adr:8] + C when ALUopc = ADC
GPR ← GPR + (not DM[adr:8]) + C when ALUopc = SBC
GPR + (not DM[adr:8]) + C when ALUopc = CPC (comparison only)

CPLop GPR (Complement Operations)

CPLop = COM, COM2, COMC

Operation

COM GPR	not GPR (logical complement)
COM2 GPR	not GPR + 1 (2’s complement of GPR)
COMC GPR	not GPR + C (logical complement of GPR with carry)

Example

```
COM2 R0           // assume R1:R0 is a 16-bit signed number.
COMC R1           // COM2 and COMC can be used to get the 2’s complement of it.
```

IncDec GPR (Increment/Decrement Operations)

IncDec = INC, INCC, DEC, DECC

Operation

INC GPR	Increase GPR, i.e., GPR ← GPR + 1
INCC GPR	Increase GPR if carry = 1, i.e., GPR ← GPR + C
DEC GPR	Decrease GPR, i.e., GPR ← GPR + FFh
DECC GPR	Decrease GPR if carry = 0, i.e., GPR ← GPR + FFh + C

Example

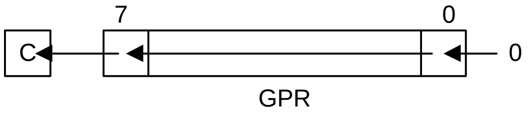
```
INC R0          // assume R1:R0 is a 16-bit number
INCC R1         // to increase R1:R0, use INC and INCC.

DEC R0          // assume R1:R0 is a 16-bit number
DECC R1         // to decrease R1:R0, use DEC and DECC.
```

SHIFT/ROTATE INSTRUCTIONS

Shift (Rotate) instructions shift (rotate) the given operand by 1 bit. Depending on the operation performed, a number of Status Register 1 (SR1) bits, namely Carry (C), Zero (Z), Overflow (V), and Negative (N), are set.

SL GPR

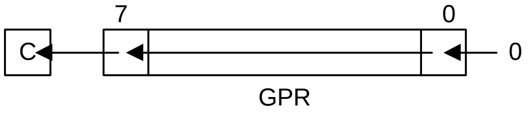
Operation



Carry (C) is the MSB of GPR before shifting, Negative (N) is the MSB of GPR after shifting. Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

SLA GPR

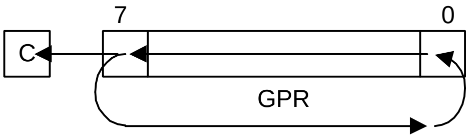
Operation



Carry (C) is the MSB of GPR before shifting, Negative (N) is the MSB of GPR after shifting. Overflow (V) will be 1 if the MSB of the result is different from C. Z will be 1 if the result is 0.

RL GPR

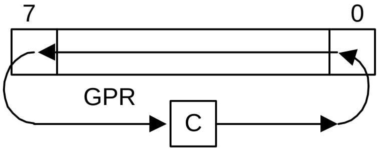
Operation



Carry (C) is the MSB of GPR before rotating. Negative (N) is the MSB of GPR after rotatin/g. Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

RLC GPR

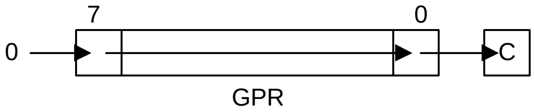
Operation



Carry (C) is the MSB of GPR before rotating, Negative (N) is the MSB of GPR after rotating. Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

SR GPR

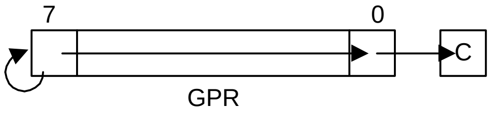
Operation



Carry (C) is the LSB of GPR before shifting, Negative (N) is the MSB of GPR after shifting. Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

SRA GPR

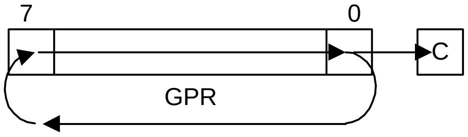
Operation



Carry (C) is the LSB of GPR before shifting, Negative (N) is the MSB of GPR after shifting. Overflow (V) is not affected. Z will be 1 if the result is 0.

RR GPR

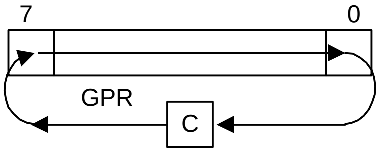
Operation



Carry (C) is the LSB of GPR before rotating. Negative (N) is the MSB of GPR after rotating. Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

RRC GPR

Operation



Carry (C) is the LSB of GPR before rotating, Negative (N) is the MSB of GPR after rotating. Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

LOAD INSTRUCTIONS

Load instructions transfer data from data memory to a register or from a register to data memory, or assigns an immediate value into a register. As a side effect, a load instruction placing a value into a register sets the Zero (Z) and Negative (N) bits in Status Register 1 (SR1), if the placed data is 00h and the MSB of the data is 1, respectively.

LD GPR, adr:8

Loads the value of DM[adr:8] into GPR. Adr:8 is offset in the page specified by the value of eid in Status Register 0 (SR0).

Operation

```
GPR ← DM[00h:adr:8]  if eid = 0
GPR ← DM[IDH:adr:8]  if eid = 1
```

Note that this is an 8-bit operation.

Example

```
LD R0, 80h            // assume eid = 1 and IDH= 01H
                      // R0 ← DM[0180h]
```

LD GPR, @idm

Loads a value from the data memory location specified by @idm into GPR.
idm = IDx+off:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]!
(IDx = ID0 or ID1)

Operation

```
GPR ← DM[IDx], IDx ← IDx + offset:5 when idm = IDx + offset:5
GPR ← DM[IDx - offset:5], IDx ← IDx - offset:5 when idm = [IDx - offset:5]
GPR ← DM[IDx + offset:5] when idm = [IDx + offset:5]!
GPR ← DM[IDx - offset:5] when idm = [IDx - offset:5]!
```

When carry is generated from IDL (on a post-increment or pre-decrement), it is propagated to IDH.

Example

```
LD R0, @[ID0 + 03h]!  // assume IDH:IDL0 = 0270h
                      // R0 ← DM[0273h], IDH:IDL0 ← 0270h
```

LD REG, #imm:8

Loads an 8-bit immediate value into REG. REG can be either GPR or an SPR0 group register - IDH (Index of Data Memory Higher Byte Register), IDL0 (Index of Data Memory Lower Byte Register)/ IDL1, and Status Register 0 (SR0). #imm:8 is an 8-bit immediate value.

Operation

REG ← #imm:8

Example

```
LD R0 #7Ah                 // R0 ← 7Ah
LD IDH, #03h               // IDH ← 03h
```

LD GPR:bs:2, GPR:bs:2

Loads a value of a register from a specified bank into another register in a specified bank.

Example

```
LD R0:1, R2:3              // R0 in bank 1, R2 in bank 3
```

LD GPR, TBH/TBL

Loads the value of TBH or TBL into GPR. TBH and TBL are 8-bit long registers used exclusively for LDC instructions that access program memory. Therefore, after an LDC instruction, LD GPR, TBH/TBL instruction will usually move the data into GPRs, to be used for other operations.

Operation

GPR ← TBH (or TBL)

Example

```
LDC @IL                    // gets a program memory item residing @ ILX:ILH:ILL
LD R0, TBH
LD R1, TBL
```

LD TBH/TBL, GPR

Loads the value of GPR into TBH or TBL. These instructions are used in pair in interrupt service routines to save and restore the values in TBH/TBL as needed.

Operation

TBH (or TBL) ← GPR

LD GPR, SPR

Loads the value of SPR into GPR.

Operation

GPR ← SPR

Example

```
LD R0, IDH                 // R0 ← IDH
```

LD SPR, GPR

Loads the value of GPR into SPR.

Operation

SPR ← GPR

Example

```
LD IDH, R0                    // IDH ← R0
```

LD adr:8, GPR

Stores the value of GPR into data memory (DM). adr:8 is offset in the page specified by the value of eid in Status Register 0 (SR0).

Operation

DM[00h:adr:8] ← GPR if eid = 0
DM[IDH:adr:8] ← GPR if eid = 1

Note that this is an 8-bit operation.

Example

```
LD 7Ah, R0                    // assume eid = 1 and IDH = 02h.
                              // DM[027Ah] ← R0
```

LD @idm, GPR

Loads a value into the data memory location specified by @idm from GPR.
idm = IDx+off:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]!
(IDx = ID0 or ID1)

Operation

DM[IDx] ← GPR, IDx ← IDx + offset:5 when idm = IDx + offset:5
DM[IDx - offset:5] ← GPR, IDx ← IDx - offset:5 when idm = [IDx - offset:5]
DM[IDx + offset:5] ← GPR when idm = [IDx + offset:5]!
DM[IDx - offset:5] ← GPR when idm = [IDx - offset:5]!

When carry is generated from IDL (on a post-increment or pre-decrement), it is propagated to IDH.

Example

```
LD @[ID0 + 03h]!, R0          // assume IDH:IDL0 = 0170h
                              // DM[0173h] ← R0, IDH:IDL0 ← 0170h
```

BRANCH INSTRUCTIONS

Branch instructions can be categorized into jump instruction, link instruction, and call instruction. A jump instruction does not save the current PC, whereas a call instruction saves (“pushes”) the current PC onto the stack and a link instruction saves the PC in the link register IL. Status registers are not affected. Each instruction type has a 2-word format that supports a 20-bit long jump.

JR cc:4, imm:9

imm:9 is a signed number (2’s complement), an offset to be added to the current PC to compute the target (PC[19:12]:(PC[11:0] + imm:9)).

Operation

```
PC[11:0] ← PC[11:0] + imm:9         if branch taken (i.e., cc:4 resolves to be true)
PC[11:0] ← PC[11:0] + 1             otherwise
```

Example

```
L18411:                             // assume current PC = 18411h.
        JR Z, 107h                  // next PC is 18518 (18411h + 107h) if Zero (Z) bit is set.
```

LJP cc:4, imm:20

Jumps to the program address specified by imm:20. If program size is less than 64K word, PC[19:16] is not affected.

Operation

```
PC[15:0] ← imm[15:0]   if branch taken and program size is less than 64K word
PC[19:0] ← imm[19:0]   if branch taken and program size is equal to 64K word or more
PC [11:0] ← PC[11:0] + 1 otherwise
```

Example

```
L18411:                             // assume current PC = 18411h.
        LJP Z, 10107h               // next instruction’s PC is 10107h If Zero (Z) bit is set
```

JNZD Rn, imm:8

Jumps to the program address specified by imm:8 if the value of the bank 3 register Rn is not zero. JNZD performs only backward jumps, with the value of Rn automatically decreased. There is one delay slot following the JNZD instruction that is always executed, regardless of whether JNZD is taken or not.

Operation

```
If (Rn == 0) PC ← PC[delay slot] (-) 2’s complement of imm:8, Rn ← Rn - 1
else PC ← PC[delay slot] + 1, Rn ← Rn - 1.
```

Example

```
LOOP_A:                         // start of loop body
          •
          •
          •
          JNZD R0, LOOP_A       // jump back to LOOP_A if R0 is not zero
          ADD R1, #2            // delay slot, always executed (you must use one cycle instruction only)
```

CALLS imm:12

Saves the current PC on the stack (“pushes” PC) and jumps to the program address specified by imm:12. The current page number PC[19:12] is not changed. Since this is a 1-word instruction, the return address pushed onto the stack is (PC + 1). If nP64KW is low when PC is saved, PC[19:16] is not saved in the stack.

Operation

```
HS[sptr][15:0] ← current PC + 1 and sptr ← sptr + 2 (push stack)      if nP64KW = 0
HS[sptr][19:0] ← current PC + 1 and sptr ← sptr + 2 (push stack)      if nP64KW = 1
PC[11:0] ← imm:12
```

Example

```
L18411:                         // assume current PC = 18411h.
          CALLS 107h            // call the subroutine at 18107h, with the current PC pushed
                                // onto the stack (HS ← 18412h) if nP64KW = 1.
```

LCALL cc:4, imm:20

Saves the current PC onto the stack (pushes PC) and jumps to the program address specified by imm:20. Since this is a 2-word instruction, the return address saved in the stack is (PC + 2). If nP64KW, a core input signal is low when PC is saved, 0000111111PC[19:16] is not saved in the stack and PC[19:16] is not set to imm[19:16].

Operation

```
HS[sptr][15:0] ← current PC + 2 and sptr + 2 (push stack)   if branch taken and nP64KW = 0
HS[sptr][19:0] ← current PC + 2 and sptr + 2 (push stack)   if branch taken and nP64KW = 1
PC[15:0] ← imm[15:0]   if branch taken and nP64KW = 0
PC[19:0] ← imm[19:0]   if branch taken and nP64KW = 1
PC[11:0] ← PC[11:0] + 2   otherwise
```

Example

```
L18411:                         // assume current PC = 18411h.
          LCALL NZ, 10107h      // call the subroutine at 10107h with the current PC pushed
                                // onto the stack (HS ← 18413h)
```

LNKS imm:12

Saves the current PC in IL and jumps to the program address specified by imm:12. The current page number PC[19:12] is not changed. Since this is a 1-word instruction, the return address saved in IL is (PC + 1). If the program size is less than 64K word when PC is saved, PC[19:16] is not saved in ILX.

Operation

```
IL[15:0] ← current PC + 1        if program size is less than 64K word
IL[19:0] ← current PC + 1        if program size is equal to 64K word or more
PC[11:0] ← imm:12
```

Example

```
L18411:                            // assume current PC = 18411h.
          LNKS 107h                // call the subroutine at 18107h, with the current PC saved
                                   // in IL (IL[19:0] ← 18412h) if program size is 64K word or more.
```

LLNK cc:4, imm:20

Saves the current PC in IL and jumps to the program address specified by imm:20. Since this is a 2-word instruction, the return address saved in IL is (PC + 2). If the program size is less than 64K word when PC is saved, PC[19:16] is not saved in ILX.

Operation

```
IL[15:0] ← current PC + 2   if branch taken and program size is less than 64K word
IL[19:0] ← current PC + 2   if branch taken and program size is 64K word or more
PC[15:0] ← imm[15:0]   if branch taken and program size is less than 64K word
PC[19:0] ← imm[19:0]   if branch taken and program size is 64K word or more
PC[11:0] ← PC[11:0] + 2   otherwise
```

Example

```
L18411:                            // assume current PC = 18411h.
          LLNK NZ, 10107h          // call the subroutine at 10107h with the current PC saved
                                   // in IL (IL[19:0] ← 18413h) if program size is 64K word or more
```

RET, IRET

Returns from the current subroutine. IRET sets ie (SR0[1]) in addition. If the program size is less than 64K word, PC[19:16] is not loaded from HS[19:16].

Operation

```
PC[15:0] ← HS[sptr - 2] and sptr ← sptr - 2 (pop stack) if program size is less than 64K word
PC[19:0] ← HS[sptr - 2] and sptr ← sptr - 2 (pop stack) if program size is 64K word or more
```

Example

```
          RET                      // assume sptr = 3h and HS[1] = 18407h.
                                   // the next PC will be 18407h and sptr is set to 1h
```

LRET

Returns from the current subroutine, using the link register IL. If the program size is less than 64K word, PC[19:16] is not loaded from ILX.

Operation

```
PC[15:0] ← IL[15:0]      if program size is less than 64K word
PC[19:0] ← IL[19:0]      if program size is 64K word or more
```

Example

```
LRET                        // assume IL = 18407h.
                            // the next instruction to execute is at PC = 18407h
                            // if program size is 64K word or more
```

JP/LNK/CALL

JP/LNK/CALL instructions are pseudo instructions. If JR/LNKS/CALLS commands (1 word instructions) can access the target address, there is no conditional code in the case of CALL/LNK and the JP/LNK/CALL commands are assembled to JR/LNKS/CALLS in linking time or else the JP/LNK/CALL commands are assembled to LJP/LLNK/LCALL (2 word instructions) instructions.

BIT MANIPULATION INSTRUCTIONS

BITop adr:8.bs

Performs a bit operation specified by op on the value in the data memory pointed by adr:8 and stores the result into R3 of current GPR bank or back into memory depending on the value of TF bit.

BITop = BITS, BITR, BITC, BITT
BITS: bit set
BITR: bit reset
BITC: bit complement
BITT: bit test (R3 is not touched in this case)
bs: bit location specifier, 0 - 7.

Operation

R3 ← DM[00h:adr:8] BITop bs if eid = 0
R3 ← DM[IDH:adr:8] BITop bs if eid = 1 (no register transfer for BITT)
Set the Zero (Z) bit if the result is 0.

Example

```
BITS 25h.3              // assume eid = 0. set bit 3 of DM[00h:25h] and store the result in R3.
BITT 25h.3              // check bit 3 of DM[00h:25h] if eid = 0.
```

BMC/BMS

Clears or sets the TF bit, which is used to determine the destination of BITop instructions. When TF bit is clear, the result of BITop instructions will be stored into R3 (fixed); if the TF bit is set, the result will be written back to memory.

Operation

TF ← 0 (BMC)
TF ← 1 (BMS)

TM GPR, #imm:8

Performs AND operation on GPR and imm:8 and sets the Zero (Z) and Negative (N) bits. No change in GPR.

Operation

Z, N flag ← GPR & #imm:8

BITop GPR.bs

Performs a bit operation on GPR and stores the result in GPR.
Since the equivalent functionality can be achieved using OR GPR, #imm:8, AND GPR, #imm:8, and XOR GPR, #imm:8, this instruction type doesn't have separate op codes.

AND SR0, #imm:8/OR SR0, #imm:8

Sets/resets bits in SR0 and stores the result back into SR0.

Operation

```
SR0 ← SR0 & #imm:8
SR0 ← SR0 | #imm:8
```

BANK #imm:2

Loads SR0[4:3] with #imm[1:0].

Operation

```
SR0[4:3] ← #imm[1:0]
```

MISCELLANEOUS INSTRUCTION

SWAP GPR, SPR

Swaps the values in GPR and SPR. SR0 and SR1 can NOT be used for this instruction.
No flag is updated, even though the destination is GPR.

Operation

```
temp ← SPR
SPR ← GPR
GPR ← temp
```

Example

```
SWAP R0, IDH          // assume IDH = 00h and R0 = 08h.
                      // after this, IDH = 08h and R0 = 00h.
```

PUSH REG

Saves REG in the stack (Pushes REG into stack).
REG = GPR, SPR

Operation

```
HS[sptr][7:0] ← REG and sptr ← sptr + 1
```

Example

```
PUSH R0               // assume R0 = 08h and sptr = 2h
                      // then HS[2][7:0] ← 08h and sptr ← 3h
```

POP REG

Pops stack into REG.
REG = GPR, SPR

Operation

REG ← HS[sptr-1][7:0] and sptr ← sptr – 1

Example

```
        POP R0              // assume sptr = 3h and HS[2] = 18407h
                            // R0 ← 07h and sptr ← 2h
```

POP

Pops 2 bytes from the stack and discards the popped data.

NOP

Does no work but increase PC by 1.

BREAK

Does nothing and does NOT increment PC. This instruction is for the debugger only. When this instruction is executed, the processor is locked since PC is not incremented. Therefore, this instruction should not be used under any mode other than the debug mode.

SYS #imm:8

Does nothing but increase PC by 1 and generates SYSCP[7:0] and nSYSID signals.

CLD GPR, imm:8

GPR ← (imm:8) and generates SYSCP[7:0], nCLDID, and nCLDWR signals.

CLD imm:8, GPR

(imm:8) ← GPR and generates SYSCP[7:0], nCLDID, and nCLDWR signals.

COP #imm:12

Generates SYSCP[11:0] and nCOPID signals.

LDC

Loads program memory item into register.

Operation

```
[TBH:TBL] ← PM[ILX:ILH:ILL]           (LDC @IL)
[TBH:TBL] ← PM[ILX:ILH:ILL], ILL++    (LDC @IL+)
```

TBH and TBL are temporary registers to hold the transferred program memory items. These can be accessed only by LD GPR and TBL/TBH instruction.

Example

```
LD ILX, R1              // assume R1:R2:R3 has the program address to access
LD ILH, R2
LD ILL, R3
LDC @IL                 // get the program data @(ILX:ILH:ILL) into TBH:TBL
```

PSEUDO INSTRUCTIONS

EI/DI

Exceptions enable and disable instruction.

Operation

```
SR0 ← OR   SR0,#00000010b  (EI)
SR0 ← AND SR0,#11111101b  (DI)
```

Exceptions are enabled or disabled through this instruction. If there is an EI instruction, the SR0.1 is set and reset, when DI instruction.

Example

```
DI
•
•
•
EI
```

SCF/RCF

Carry flag set and reset instruction.

Operation

```
CP R0,R0       (SCF)
AND R0,R0      (RCF)
```

Carry flag is set or reset through this instruction. If there is an SCF instruction, the SR1.0 is set and reset, when RCF instruction.

Example

```
SCF
RCF
```

STOP/IDLE

MCU power saving instruction.

Operation

```
SYS #0Ah       (STOP)
SYS #05h       (IDLE)
```

The STOP instruction stops the both CPU clock and system clock and causes the microcontroller to enter STOP mode. The IDLE instruction stops the CPU clock while allowing system clock oscillation to continue.

Example

```
STOP(or IDLE)
NOP
NOP
NOP
•
•
```

•

ADC — Add with Carry

Format:

ADC <op1>, <op2>
<op1>: GPR
<op2>: adr:8, GPR

Operation:

<op1> ← <op1> + <op2> + C
ADC adds the values of <op1> and <op2> and carry (C) and stores the result back into <op1>

Flags:

C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
. **N:** exclusive OR of V and MSB of result.

Example:

```
ADC     R0, 80h             // If eid = 0, R0 ← R0 + DM[0080h] + C
                            // If eid = 1, R0 ← R0 + DM[IDH:80h] + C

ADC     R0, R1              // R0 ← R0 + R1 + C

ADD     R0, R2
ADC     R1, R3
```

In the last two instructions, assuming that register pair R1:R0 and R3:R2 are 16-bit signed or unsigned numbers. Even if the result of “ADD R0, R2” is not zero, Z flag can be set to ‘1’ if the result of “ADC R1,R3” is zero. Note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit addition, take care of the change of Z flag.

ADD — Add

Format: ADD <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> ← <op1> + <op2>

ADD adds the values of <op1> and <op2> and stores the result back into <op1>.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
. **N:** exclusive OR of V and MSB of result.

Example: Given: IDH:IDL0 = 80FFh, eid = 1

```
ADD     R0, 80h              // R0 ← R0 + DM[8080h]

ADD     R0, #12h             // R0 ← R0 + 12h

ADD     R1, R2               // R1 ← R1 + R2

ADD     R0, @ID0 + 2         // R0 ← R0 + DM[80FFh], IDH ← 81h, IDL0 ← 01h
ADD     R0, @[ID0 – 3]       // R0 ← R0 + DM[80FCh], IDH ← 80h, IDL0 ← FCh
ADD     R0, @[ID0 + 2]!      // R0 ← R0 + DM[8101h], IDH ← 80h, IDL0 ← FFh
ADD     R0, @[ID0 – 2]!      // R0 ← R0 + DM[80FDh], IDH ← 80h, IDL0 ← FFh
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 8-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

AND — Bit-wise AND

Format: AND <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> ← <op1> & <op2>

AND performs bit-wise AND on the values in <op1> and <op2> and stores the result in <op1>.

Flags:

Z: set if result is zero. Reset if not.
N: set if the MSB of result is 1. Reset if not.

Example: Given: IDH:IDL0 = 01FFh, eid = 1

```
AND       R0, 7Ah               // R0 ← R0 & DM[017Ah]

AND       R1, #40h              // R1 ← R1 & 40h

AND       R0, R1                // R0 ← R0 & R1

AND       R1, @ID0 + 3          // R1 ← R1 & DM[01FFh], IDH:IDL0 ← 0202h
AND       R1, @[ID0 – 5]        // R1 ← R1 & DM[01FAh], IDH:IDL0 ← 01FAh
AND       R1, @[ID0 + 7]!       // R1 ← R1 & DM[0206h], IDH:IDL0 ← 01FFh
AND       R1, @[ID0 – 2]!       // R1 ← R1 & DM[01FDh], IDH:IDL0 ← 01FFh
```

In the first instruction, if eid bit in SR0 is zero, register R0 has garbage value because data memory DM[0051h-007Fh] are not mapped in S3CB519/S3FB519. In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 8-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

AND SR0 — Bit-wise AND with SR0

Format: AND SR0, #imm:8

Operation: SR0 ← SR0 & imm:8

AND SR0 performs the bit-wise AND operation on the value of SR0 and imm:8 and stores the result in SR0.

Flags: –

Example: Given: SR0 = 11000010b

```
nIE     EQU     ~02h
nIE0    EQU     ~40h
nIE1    EQU     ~80h

        AND     SR0, #nIE | nIE0 | nIE1

        AND     SR0, #11111101b
```

In the first example, the statement “AND SR0, #nIE|nIE0|nIE1” clear all of bits of the global interrupt, interrupt 0 and interrupt 1. On the contrary, cleared bits can be set to ‘1’ by instruction “OR SR0, #imm:8”. Refer to instruction OR SR0 for more detailed explanation about enabling bit.

In the second example, the statement “AND SR0, #11111101b” is equal to instruction DI, which is disabling interrupt globally.

BANK — GPR Bank selection

Format: BANK #imm:2

Operation: SR0[4:3] ← imm:2

Flags: –

NOTE: For explanation of the CalmRISC banked register file and its usage, please refer to chapter 3.

Example:

```
BANK     #1                  // Select register bank 1
LD       R0, #11h            // Bank1's R0 ← 11h

BANK     #2                  // Select register bank 2
LD       R1, #22h            // Bank2's R1 ← 22h
```

BITC — Bit Complement

Format: BITC adr:8.bs

bs: 3-digit bit specifier

Operation: R3 ← ((adr:8) ^ (2**bs)) if (TF == 0)

(adr:8) ← ((adr:8) ^ (2**bs)) if (TF == 1)

BITC complements the specified bit of a value read from memory and stores the result in R3 or back into memory, depending on the value of TF. TF is set or clear by BMS/BMC instruction.

Flags: **Z:** set if result is zero. Reset if not.

NOTE: Since the destination register R3 is fixed, it is not specified explicitly.

Example: Given: IDH = 01, DM[0180h] = FFh, eid = 1

```
BMC                     // TF ← 0
BITC    80h.0           // R3 ← FEh, DM[0180h] = FFh

BMS                     // TF ← 1
BITC    80h.1           // DM[0180h] ← FDh
```

BITR — Bit Reset

Format: BITR adr:8.bs

bs: 3-digit bit specifier

Operation: R3 ← ((adr:8) & $((11111111)_2$ - (2**bs))) if (TF == 0)

(adr:8) ← ((adr:8) & $((11111111)_2$ - (2**bs))) if (TF == 1)

BITR resets the specified bit of a value read from memory and stores the result in R3 or back into memory, depending on the value of TF. TF is set or clear by BMS/BMC instruction.

Flags: **Z:** set if result is zero. Reset if not.

NOTE: Since the destination register R3 is fixed, it is not specified explicitly.

Example: Given: IDH = 01, DM[0180h] = FFh, eid = 1

```
BMC                              // TF ← 0
BITR      80h.1                  // R3 ← FDh, DM[0180h] = FFh

BMS                              // TF ← 1
BITR      80h.2                  // DM[0180h] ← FBh
```

BITS — Bit Set

Format: BITS adr:8.bs

bs: 3-digit bit specifier.

Operation: R3 ← ((adr:8) | (2**bs)) if (TF == 0)

(adr:8) ← ((adr:8) | (2**bs)) if (TF == 1)

BITS sets the specified bit of a value read from memory and stores the result in R3 or back into memory, depending on the value of TF. TF is set or clear by BMS/BMC instruction.

Flags: **Z:** set if result is zero. Reset if not.

NOTE: Since the destination register R3 is fixed, it is not specified explicitly.

Example: Given: IDH = 01, DM[0180h] = F0h, eid = 1

```
BMC                             // TF ← 0
BITS      80h.1                 // R3 ← 0F2h, DM[0180h] = F0h

BMS                             // TF ← 1
BITS      80h.2                 // DM[0180h] ← F4h
```

BITT — Bit Test

Format: BITT adr:8.bs

bs: 3-digit bit specifier.

Operation: Z ← ~((adr:8) & (2**bs))

BITT tests the specified bit of a value read from memory.

Flags: **Z:** set if result is zero. Reset if not.

Example: Given: DM[0080h] = F7h, eid = 0

```
        BITT    80h.3               // Z flag is set to ‘1’
        JR      Z, %1               // Jump to label %1 because condition is true.
        •
        •
        •
%1      BITS    80h.3
        NOP
        •
        •
        •
```

BMC/BMS – TF bit clear/set

Format: BMS/BMC

Operation: BMC/BMS clears (sets) the TF bit.

TF ← 0 if BMC

TF ← 1 if BMS

TF is a single bit flag which determines the destination of bit operations, such as BITC, BITR, and BITS.

Flags: –

NOTE: BMC/BMS are the only instructions that modify the content of the TF bit.

Example:

```
BMS                             // TF ← 1
BITS      81h.1

BMC                             // TF ← 0
BITR      81h.2
LD        R0, R3
```

CALL — Conditional Subroutine Call (Pseudo Instruction)

Format: CALL cc:4, imm:20
CALL imm:12

Operation: If CALLS can access the target address and there is no conditional code (cc:4), CALL command is assembled to CALLS (1-word instruction) in linking time, else the CALL is assembled to LCALL (2-word instruction).

Example:

```
        CALL    C, Wait         // HS[sptr][15:0] ← current PC + 2, sptr ← sptr + 2
        •                       // 2-word instruction
        •
        •
        CALL    0088h           // HS[sptr][15:0] ← current PC + 1, sptr ← sptr + 2
        •                       // 1-word instruction
        •
        •
Wait:   NOP                     // Address at 0088h
        NOP
        NOP
        NOP
        NOP
        RET
```

CALLS — Call Subroutine

Format: CALLS imm:12

Operation: HS[sptr][15:0] ← current PC + 1, sptr ← sptr + 2 if the program size is less than 64K word.

HS[sptr][19:0] ← current PC + 1, sptr ← sptr + 2 if the program size is equal to or over 64K word.

PC[11:0] ← imm:12
CALLS unconditionally calls a subroutine residing at the address specified by imm:12.

Flags: –

Example:

```
        CALLS     Wait
        •
        •
        •
Wait:   NOP
        NOP
        NOP
        RET
```

Because this is a 1-word instruction, the saved returning address on stack is (PC + 1).

CLD — Load into Coprocessor

Format: CLD imm:8, <op>

<op>: GPR

Operation: (imm:8) ← <op>

CLD loads the value of <op> into (imm:8), where imm:8 is used to access the external coprocessor's address space.

Flags: –

Example:

```
AH      EQU    00h
AL      EQU    01h
BH      EQU    02h
BL      EQU    03h
        •
        •
        •
        CLD    AH, R0           // A[15:8] ← R0
        CLD    AL, R1           // A[7:0] ← R1

        CLD    BH, R2           // B[15:8] ← R2
        CLD    BL, R3           // B[7:0] ← R3
```

The registers A[15:0] and B[15:0] are Arithmetic Unit (AU) registers of MAC816.
Above instructions generate SYSCP[7:0], nCLDID and CLDWR signals to access MAC816.

CLD — Load from Coprocessor

Format: CLD <op>, imm:8

<op>: GPR

Operation: <op> ← (imm:8)

CLD loads a value from the coprocessor, whose address is specified by imm:8.

Flags: **Z:** set if the loaded value in <op1> is zero. Reset if not.
N: set if the MSB of the loaded value in <op1> is 1. Reset if not.

Example:

```
AH      EQU   00h
AL      EQU   01h
BH      EQU   02h
BL      EQU   03h
        •
        •
        •
        CLD   R0, AH         // R0 ← A[15:8]
        CLD   R1, AL         // R1 ← A[7:0]

        CLD   R2, BH         // R2 ← B[15:8]
        CLD   R3, BL         // R3 ← B[7:0]
```

The registers A[15:0] and B[15:0] are Arithmetic Unit (AU) registers of MAC816.
Above instructions generate SYSCP[7:0], nCLDID and CLDWR signals to access MAC816.

COM — 1's or Bit-wise Complement

Format: COM <op>

<op>: GPR

Operation: <op> ← ~<op>

COM takes the bit-wise complement operation on <op> and stores the result in <op>.

Flags:
Z: set if result is zero. Reset if not.
N: set if the MSB of result is 1. Reset if not.

Example: Given: R1 = 5Ah

```
COM     R1                              // R1 ← A5h, N flag is set to ‘1’
```

COM2 — 2's Complement

Format: COM2 <op>

<op>: GPR

Operation: <op> ← ~<op> + 1

COM2 computes the 2's complement of <op> and stores the result in <op>.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative.

Example: Given: R0 = 00h, R1 = 5Ah

```
COM2    R0                    // R0 ← 00h, Z and C flags are set to ‘1’.

COM2    R1                    // R1 ← A6h, N flag is set to ‘1’.
```

COMC — Bit-wise Complement with Carry

Format: COMC <op>

<op>: GPR

Operation: <op> ← ~<op> + C

COMC takes the bit-wise complement of <op>, adds carry and stores the result in <op>.

Flags:

C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: If register pair R1:R0 is a 16-bit number, then the 2’s complement of R1:R0 can be obtained by COM2 and COMC as following.

```
COM2          R0
COMC          R1
```

Note that Z flag do not exactly reflect result of 16-bit operation. For example, if 16-bit register pair R1: R0 has value of FF01h, then 2’s complement of R1: R0 is made of 00FFh by COM2 and COMC. At this time, by instruction COMC, zero (Z) flag is set to ‘1’ as if the result of 2’s complement for 16-bit number is zero. Therefore when programming 16-bit comparison, take care of the change of Z flag.

COP — Coprocessor

Format: COP #imm:12

Operation: COP passes imm:12 to the coprocessor by generating SYSCP[11:0] and nCOPID signals.

Flags: –

Example:

```
COP     #0D01h          // generate 1 word instruction code(FD01h)
COP     #0234h          // generate 1 word instruction code(F234h)
```

The above two instructions are equal to statement “ELD A, #1234h” for MAC816 operation. The microcode of MAC instruction “ELD A, #1234h” is “FD01F234”, 2-word instruction. In this, code ‘F’ indicates ‘COP’ instruction.

CP — Compare

Format: CP <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> + ~<op2> + 1

CP compares the values of <op1> and <op2> by subtracting <op2> from <op1>. Contents of <op1> and <op2> are not changed.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero (i.e., <op1> and <op2> are same). Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: R0 = 73h, R1 = A5h, IDH:IDL0 = 0123h, DM[0123h] = A5, eid = 1

```
CP      R0, 80h              // C flag is set to ‘1’

CP      R0, #73h             // Z and C flags are set to ‘1’

CP      R0, R1               // V flag is set to ‘1’

CP      R1, @ID0             // Z and C flags are set to ‘1’
CP      R1, @[ID0 – 5]
CP      R2, @[ID0 + 7]!
CP      R2, @[ID0 – 2]!
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 8-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

CPC — Compare with Carry

Format: CPC <op1>, <op2>

<op1>: GPR
<op2>: adr:8, GPR

Operation: <op1> ← <op1> + ~<op2> + C

CPC compares <op1> and <op2> by subtracting <op2> from <op1>. Unlike CP, however, CPC adds (C - 1) to the result. Contents of <op1> and <op2> are not changed.

Flags:

C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: If register pair R1:R0 and R3:R2 are 16-bit signed or unsigned numbers, then use CP and CPC to compare two 16-bit numbers as follows.

```
CP      R0, R1
CPC     R2, R3
```

Because CPC considers C when comparing <op1> and <op2>, CP and CPC can be used in pair to compare 16-bit operands. But note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit comparison, take care of the change of Z flag.

DEC — Decrement

Format: DEC <op>

<op>: GPR

Operation: <op> ← <op> + 0FFh

DEC decrease the value in <op> by adding 0FFh to <op>.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: R0 = 80h, R1 = 00h

```
DEC     R0              // R0 ← 7Fh, C, V and N flags are set to ‘1’

DEC     R1              // R1 ← FFh, N flags is set to ‘1’
```

DECC — Decrement with Carry

Format: DECC <op>

<op>: GPR

Operation: <op> ← <op> + 0FFh + C

DECC decrease the value in <op> when carry is not set. When there is a carry, there is no change in the value of <op>.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: If register pair R1:R0 is 16-bit signed or unsigned number, then use DEC and DECC to decrement 16-bit number as follows.

```
DEC     R0
DECC    R1
```

Note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit decrement, take care of the change of Z flag.

DI — Disable Interrupt (Pseudo Instruction)

Format: DI

Operation: Disables interrupt globally. It is same as “AND SR0, #0FDh” .
DI instruction sets bit1 (ie: global interrupt enable) of SR0 register to “0”

Flags: –

Example: Given: SR0 = 03h

```
DI                                      // SR0 ← SR0 & 11111101b
```

DI instruction clears SR0[1] to ‘0’, disabling interrupt processing.

EI — Enable Interrupt (Pseudo Instruction)

Format: EI

Operation: Enables interrupt globally. It is same as “OR SR0, #02h” .
EI instruction sets the bit1 (ie: global interrupt enable) of SR0 register to “1”

Flags: –

Example: Given: SR0 = 01h

```
EI                                        // SR0 ← SR0 | 00000010b
```

The statement “EI” sets the SR0[1] to ‘1’, enabling all interrupts.

IDLE — Idle Operation (Pseudo Instruction)

Format: IDLE

Operation: The IDLE instruction stops the CPU clock while allowing system clock oscillation to continue. Idle mode can be released by an interrupt or reset operation.
The IDLE instruction is a pseudo instruction. It is assembled as "SYS #05H", and this generates the SYSCP[7-0] signals. Then these signals are decoded and the decoded signals execute the idle operation.

Flags: –

NOTE: The next instruction of IDLE instruction is executed, so please use the NOP instruction after the IDLE instruction.

Example:

```
IDLE
NOP
NOP
NOP
•
•
•
```

The IDLE instruction stops the CPU clock but not the system clock.

INC — Increment

Format: INC <op>

<op>: GPR

Operation: <op> ← <op> + 1

INC increase the value in <op>.

Flags:

C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: R0 = 7Fh, R1 = FFh

```
INC     R0              // R0 ← 80h, V flag is set to ‘1’

INC     R1              // R1 ← 00h, Z and C flags are set to ‘1’
```

INCC — Increment with Carry

Format: INCC <op>

<op>: GPR

Operation: <op> ← <op> + C

INCC increase the value of <op> only if there is carry. When there is no carry, the value of <op> is not changed.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: exclusive OR of V and MSB of result.

Example: If register pair R1:R0 is 16-bit signed or unsigned number, then use INC and INCC to increment 16-bit number as following.

```
INC       R0
INCC      R1
```

Assume R1:R0 is 0010h, statement “INC R0” increase R0 by one without carry and statement “INCC R1” set zero (Z) flag to ‘1’ as if the result of 16-bit increment is zero. Note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit increment, take care of the change of Z flag.

IRET — Return from Interrupt Handling

Format: IRET

Operation: PC ← HS[sptr - 2], sptr ← sptr - 2

IRET pops the return address (after interrupt handling) from the hardware stack and assigns it to PC. The ie (i.e., SR0[1]) bit is set to allow further interrupt generation.

Flags: –

NOTE: The program size (indicated by the nP64KW signal) determines which portion of PC is updated. When the program size is less than 64K word, only the lower 16 bits of PC are updated (i.e., PC[15:0] ← HS[sptr – 2]).
When the program size is 64K word or more, the action taken is PC[19:0] ← HS[sptr - 2].

Example:

```
SF_EXCEP:    NOP                      // Stack full exception service routine
             •
             •
             •
             IRET
```

JNZD — Jump Not Zero with Delay slot

Format: JNZD <op>, imm:8

<op>: GPR (bank 3's GPR only)

imm:8 is an signed number

Operation: PC ← PC[delay slot] - 2's complement of imm:8

<op> ← <op> - 1

JNZD performs a backward PC-relative jump if <op> evaluates to be non-zero. Furthermore, JNZD decrease the value of <op>. The instruction immediately following JNZD (i.e., in delay slot) is always executed, and this instruction must be 1 cycle instruction.

Flags: –

NOTE: Typically, the delay slot will be filled with an instruction from the loop body. It is noted, however, that the chosen instruction should be "dead" outside the loop for it executes even when the loop is exited (i.e., JNZD is not taken).

Example: Given: IDH = 03h, eid = 1

```
        BANK    #3
        LD      R0, #0FFh          // R0 is used to loop counter
        LD      R1, #0
%1      LD      IDL0, R0
        JNZD    R0, %B1            // If R0 of bank3 is not zero, jump to %1.
        LD      @ID0, R1           // Clear register pointed by ID0
        •
        •
        •
```

This example can be used for RAM clear routine. The last instruction is executed even if the loop is exited.

JP — Conditional Jump (Pseudo Instruction)

Format: JP cc:4 imm:20
JP cc:4 imm:9

Operation: If JR can access the target address, JP command is assembled to JR (1 word instruction) in linking time, else the JP is assembled to LJP (2 word instruction) instruction.
There are 16 different conditions that can be used, as described in table 8-6.

Example:

```
%1    LD      R0, #10h            // Assume address of label %1 is 020Dh
      •
      •
      •
      JP      Z, %B1              // Address at 0264h

      JP      C, %F2              // Address at 0265h
      •
      •
      •
%2    LD      R1, #20h            // Assume address of label %2 is 089Ch
      •
      •
      •
```

In the above example, the statement “JP Z, %B1” is assembled to JR instruction. Assuming that current PC is 0264h and condition is true, next PC is made by PC[11:0] ← PC[11:0] + offset, offset value is “64h + A9h” without carry. ‘A9’ means 2’s complement of offset value to jump backward. Therefore next PC is 020Dh. On the other hand, statement “JP C, %F2” is assembled to LJP instruction because offset address exceeds the range of imm:9.

JR — Conditional Jump Relative

Format: JR cc:4 imm:9

cc:4: 4-bit condition code

Operation: PC[11:0] ← PC[11:0] + imm:9 if condition is true. imm:9 is a signed number, which is sign-extended to 12 bits when added to PC.
There are 16 different conditions that can be used, as described in table 8-6.

Flags: –

NOTE: Unlike LJP, the target address of JR is PC-relative. In the case of JR, imm:9 is added to PC to compute the actual jump address, while LJP directly jumps to imm:20, the target.

Example:

```
        JR      Z, %1               // Assume current PC = 1000h
        •
        •
        •
%1      LD      R0, R1              // Address at 10A5h
        •
        •
        •
```

After the first instruction is executed, next PC has become 10A5h if Z flag bit is set to ‘1’. The range of the relative address is from +255 to –256 because imm:9 is signed number.

LCALL — Conditional Subroutine Call

Format: LCALL cc:4, imm:20

Operation: HS[sptr][15:0] ← current PC + 2, sptr ← sptr + 2, PC[15:0] ← imm[15:0] if the condition holds and the program size is less than 64K word.

HS[sptr][19:0] ← current PC + 2, sptr ← sptr + 2, PC[19:0] ← imm:20 if the condition holds and the program size is equal to or over 64K word.

PC[11:0] ← PC[11:0] + 2 otherwise.
LCALL instruction is used to call a subroutine whose starting address is specified by imm:20.

Flags: –

Example:

```
LCALL     L1

LCALL     C, L2
```

Label L1 and L2 can be allocated to the same or other section. Because this is a 2-word instruction, the saved returning address on stack is (PC + 2).

LD adr:8 — Load into Memory

Format: LD adr:8, <op>

<op>: GPR

Operation: DM[00h:adr:8] ← <op> if eid = 0
DM[IDH:adr:8] ← <op> if eid = 1

LD adr:8 loads the value of <op> into a memory location. The memory location is determined by the eid bit and adr:8.

Flags: –

Example: Given: IDH = 01h

```
LD          80h, R0
```

If eid bit of SR0 is zero, the statement “LD 80h, R0” load value of R0 into DM[0080h], else eid bit was set to ‘1’, the statement “LD 80h, R0” load value of R0 into DM[0180h]

LD @idm — Load into Memory Indexed

Format: LD @idm, <op>

<op>: GPR

Operation: (@idm) ← <op>

LD @idm loads the value of <op> into the memory location determined by @idm. Details of the @idm format and how the actual address is calculated can be found in chapter 2.

Flags: –

Example: Given R0 = 5Ah, IDH:IDL0 = 8023h, eid = 1

```
LD      @ID0, R0            // DM[8023h] ← 5Ah
LD      @ID0 + 3, R0        // DM[8023h] ← 5Ah, IDL0 ← 26h
LD      @[ID0-5], R0        // DM[801Eh] ← 5Ah, IDL0 ← 1Eh
LD      @[ID0+4]!, R0       // DM[8027h] ← 5Ah, IDL0 ← 23h
LD      @[ID0-2]!, R0       // DM[8021h] ← 5Ah, IDL0 ← 23h
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 8-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

LD — Load Register

Format: LD <op1>, <op2>

<op1>: GPR
<op2>: GPR, SPR, adr:8, @idm, #imm:8

Operation: <op1> ← <op2>

LD loads a value specified by <op2> into the register designated by <op1>.

Flags:
Z: set if result is zero. Reset if not.
N: exclusive OR of V and MSB of result.

Example: Given: R0 = 5Ah, R1 = AAh, IDH:IDL0 = 8023h, eid = 1

```
LD      R0, R1              // R0 ← AAh

LD      R1, IDH             // R1 ← 80h

LD      R2, 80h             // R2 ← DM[8080h]

LD      R0, #11h            // R0 ← 11h

LD      R0, @ID0+1          // R0 ← DM[8023h], IDL0 ← 24h
LD      R1, @[ID0-2]        // R1 ← DM[8021h], IDL0 ← 21h
LD      R2, @[ID0+3]!       // R2 ← DM[8026h], IDL0 ← 23h
LD      R3, @[ID0-5]!       // R3 ← DM[801Eh], IDL0 ← 23h
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 8-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

LD — Load GPR:bankd, GPR:banks

Format: LD <op1>, <op2>

<op1>: GPR: bankd
<op2>: GPR: banks

Operation: <op1> ← <op2>

LD loads a value of a register in a specified bank (banks) into another register in a specified bank (bankd).

Flags:
Z: set if result is zero. Reset if not.
N: exclusive OR of V and MSB of result.

Example:

```
LD        R2:1, R0:3              // Bank1’s R2 ← bank3’s R0

LD        R0:0, R0:2              // Bank0’s R0 ← bank2’s R0
```

LD — Load GPR, TBH/TBL

Format: LD <op1>, <op2>

<op1>: GPR
<op2>: TBH/TBL

Operation: <op1> ← <op2>

LD loads a value specified by <op2> into the register designated by <op1>.

Flags:
Z: set if result is zero. Reset if not.
N: exclusive OR of V and MSB of result.

Example: Given: register pair R1:R0 is 16-bit unsigned data.

```
LDC     @IL                      // TBH:TBL ← PM[ILX:ILH:ILL]
LD      R1, TBH                  // R1 ← TBH
LD      R0, TBL                  // R0 ← TBL
```

LD — Load TBH/TBL, GPR

Format: LD <op1>, <op2>

<op1>: TBH/TBL
<op2>: GPR

Operation: <op1> ← <op2>

LD loads a value specified by <op2> into the register designated by <op1>.

Flags: –

Example: Given: register pair R1:R0 is 16-bit unsigned data.

```
LD          TBH, R1                    // TBH ← R1
LD          TBL, R0                    // TBL ← R0
```

LD SPR — Load SPR

Format: LD <op1>, <op2>

<op1>: SPR
<op2>: GPR

Operation: <op1> ← <op2>

LD SPR loads the value of a GPR into an SPR.
Refer to Table 3-1 for more detailed explanation about kind of SPR.

Flags: –

Example: Given: register pair R1:R0 = 1020h

```
LD        ILH, R1                 // ILH ← 10h
LD        ILL, R0                 // ILL ← 20h
```

LD SPR0 — Load SPR0 Immediate

Format: LD SPR0, #imm:8

Operation: SPR0 ← imm:8

LD SPR0 loads an 8-bit immediate value into SPR0.

Flags: –

Example: Given: eid = 1, idb = 0 (index register bank 0 selection)

```
LD          IDH, #80h              // IDH point to page 80h
LD          IDL1, #44h
LD          IDL0, #55h
LD          SR0, #02h
```

The last instruction set ie (global interrupt enable) bit to ‘1’.
Special register group 1 (SPR1) registers are not supported in this addressing mode.

LDC — Load Code

Format: LDC <op1>

<op1>: @IL, @IL+

Operation: TBH:TBL ← PM[ILX:ILH:ILL]

ILL ← ILL + 1 (@IL+ only)

LDC loads a data item from program memory and stores it in the TBH:TBL register pair.

@IL+ increase the value of ILL, efficiently implementing table lookup operations.

Flags: –

Example:

```
LD          ILX, R1
LD          ILH, R2
LD          ILL, R3
LDC         @IL                      // Loads value of PM[ILX:ILH:ILL] into TBH:TBL

LD          R1, TBH                  // Move data in TBH:TBL to GPRs for further processing
LD          R0, TBL
```

The statement “LDC @IL” do not increase, but if you use statement “LDC @IL+”, ILL register is increased by one after instruction execution.

LJP — Conditional Jump

Format: LJP cc:4, imm:20

cc:4: 4-bit condition code

Operation: PC[15:0] ← imm[15:0] if condition is true and the program size is less than 64K word. If the program is equal to or larger than 64K word, PC[19:0] ← imm[19:0] as long as the condition is true. There are 16 different conditions that can be used, as described in table 8-6.

Flags: –

NOTE: LJP cc:4 imm:20 is a 2-word instruction whose immediate field directly specifies the target address of the jump.

Example:

```
        LJP     C, %1                   // Assume current PC = 0812h
        •
        •
        •
%1      LD      R0, R1                  // Address at 10A5h
        •
        •
        •
```

After the first instruction is executed, LJP directly jumps to address 10A5h if condition is true.

LLNK — Linked Subroutine Call Conditional

Format: LLNK cc:4, imm:20

cc:4: 4-bit condition code

Operation: If condition is true, IL[19:0] ← {PC[19:12], PC[11:0] + 2}.

Further, when the program is equal to or larger than 64K word, PC[19:0] ← imm[19:0] as long as the condition is true. If the program is smaller than 64K word, PC[15:0] ← imm[15:0].
There are 16 different conditions that can be used, as described in table 8-6.

Flags: –

NOTE: LLNK is used to conditionally to call a subroutine with the return address saved in the link register (IL) without stack operation. This is a 2-word instruction.

Example:

```
        LLNK    Z, %1                    // Address at 005Ch, ILX:ILH:ILL ← 00:00:5Eh
        NOP                              // Address at 005Eh
        •
        •
        •
%1      LD      R0, R1
        •
        •
        •
        LRET
```

LNK — Linked Subroutine Call (Pseudo Instruction)

Format: LNK cc:4, imm:20
LNK imm:12

Operation: If LNKS can access the target address and there is no conditional code (cc:4), LNK command is assembled to LNKS (1 word instruction) in linking time, else the LNK is assembled to LLNK (2 word instruction).

Example:

```
        LNK     Z, Link1            // Equal to “LLNK Z, Link1”
        LNK     Link2               // Equal to “LNKS Link2”
        NOP
        •
        •
        •
Link2:  NOP
        •
        •
        •
        LRET

Subroutines     section CODE, ABS 0A00h
        Subroutines
Link1:  NOP
        •
        •
        •
        LRET
```

LNKS — Linked Subroutine Call

Format: LNKS imm:12

Operation: IL[19:0] ← {PC[19:12], PC[11:0] + 1} and PC[11:0] ← imm:12
LNKS saves the current PC in the link register and jumps to the address specified by imm:12.

Flags: –

NOTE: LNKS is used to call a subroutine with the return address saved in the link register (IL) without stack operation.

Example:

```
        LNKS    Link1              // Address at 005Ch, ILX:ILH:ILL ← 00:00:5Dh
        NOP                        // Address at 005Dh
        •
        •
        •

Link1:  NOP
        •
        •
        •
        LRET
```

LRET — Return from Linked Subroutine Call

Format: LRET

Operation: PC ← IL[19:0]
LRET returns from a subroutine by assigning the saved return address in IL to PC.

Flags: –

Example:

```
        LNK       Link1
Link1:  NOP
        •
        •
        •
        LRET                                  ;   PC[19:0] ← ILX:ILH:ILL
```

NOP — No Operation

Format: NOP

Operation: No operation.

When the instruction NOP is executed in a program, no operation occurs. Instead, the instruction time is delayed by approximately one machine cycle per each NOP instruction encountered.

Flags: –

Example:

NOP

OR — Bit-wise OR

Format: OR <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> ← <op1> | <op2>
OR performs the bit-wise OR operation on <op1> and <op2> and stores the result in <op1>.

Flags: **Z:** set if result is zero. Reset if not.
N: exclusive OR of V and MSB of result.

Example: Given: IDH:IDL0 = 031Eh, eid = 1

```
OR        R0, 80h                  // R0 ← R0 | DM[0380h]

OR        R1, #40h                 // Mask bit6 of R1

OR        R1, R0                   // R1 ← R1 | R0

OR        R0, @ID0                 // R0 ← R0 | DM[031Eh], IDL0 ← 1Eh
OR        R1, @[ID0-1]             // R1 ← R1 | DM[031Dh], IDL0 ← 1Dh
OR        R2, @[ID0+1]!            // R2 ← R2 | DM[031Fh], IDL0 ← 1Eh
OR        R3, @[ID0-1]!            // R3 ← R3 | DM[031Dh], IDL0 ← 1Eh
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 8-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

OR SR0 — Bit-wise OR with SR0

Format: OR SR0, #imm:8

Operation: SR0 ← SR0 | imm:8

OR SR0 performs the bit-wise OR operation on SR0 and imm:8 and stores the result in SR0.

Flags: –

Example: Given: SR0 = 00000000b

```
EID     EQU     01h
IE      EQU     02h
IDB1    EQU     04h
IE0     EQU     40h
IE1     EQU     80h

        OR      SR0, #IE | IE0 | IE1

        OR      SR0, #00000010b
```

In the first example, the statement “OR SR0, #EID|IE|IE0” set global interrupt(ie), interrupt 0(ie0) and interrupt 1(ie1) to ‘1’ in SR0. On the contrary, enabled bits can be cleared with instruction “AND SR0, #imm:8”. Refer to instruction AND SR0 for more detailed explanation about disabling bit.

In the second example, the statement “OR SR0, #00000010b” is equal to instruction EI, which is enabling interrupt globally.

POP — POP

Format: POP

Operation: sptr ← sptr – 2

POP decrease sptr by 2. The top two bytes of the hardware stack are therefore invalidated.

Flags: –

Example: Given: sptr[5:0] = 001010b

POP

This POP instruction decrease sptr[5:0] by 2. Therefore sptr[5:0] is 001000b.

POP — POP to Register

Format: POP <op>

<op>: GPR, SPR

Operation: <op> ← HS[sptr - 1], sptr ← sptr - 1

POP copies the value on top of the stack to <op> and decrease sptr by 1.

Flags:
Z: set if the value copied to <op> is zero. Reset if not.
N: set if the value copied to <op> is negative. Reset if not.
When <op> is SPR, no flags are affected, including Z and N.

Example:

```
POP     R0          // R0 ← HS[sptr-1], sptr  ← sptr-1

POP     IDH         // IDH ← HS[sptr-1], sptr  ← sptr-1
```

In the first instruction, value of HS[sptr-1] is loaded to R0 and the second instruction “POP IDH” load value of HS[sptr-1] to register IDH. Refer to chapter 5 for more detailed explanation about POP operations for hardware stack.

PUSH — Push Register

Format: PUSH <op>

<op>: GPR, SPR

Operation: HS[sptr] ← <op>, sptr ← sptr + 1

PUSH stores the value of <op> on top of the stack and increase sptr by 1.

Flags: –

Example:

```
PUSH      R0                       // HS[sptr] ← R0, sptr ← sptr + 1

PUSH      IDH                      // HS[sptr] ← IDH, sptr ← sptr + 1
```

In the first instruction, value of register R0 is loaded to HS[sptr-1] and the second instruction "PUSH IDH" load value of register IDH to HS[sptr-1]. Current HS pointed by stack point sptr[5:0] be emptied. Refer to chapter 5 for more detailed explanation about PUSH operations for hardware stack.

RET — Return from Subroutine

Format: RET

Operation: PC ← HS[sptr - 2], sptr ← sptr – 2

RET pops an address on the hardware stack into PC so that control returns to the subroutine call site.

Flags: –

Example: Given: sptr[5:0] = 001010b

```
        CALLS    Wait                 // Address at 00120h
        •
        •
        •
Wait:   NOP                           // Address at 01000h
        NOP
        NOP
        NOP
        NOP
        RET
```

After the first instruction CALLS execution, “PC+1”, 0121h is loaded to HS[5] and hardware stack pointer sptr[5:0] have 001100b and next PC became 01000h. The instruction RET pops value 0121h on the hardware stack HS[sptr-2] and load to PC then stack pointer sptr[[5:0] became 001010b.

RL — Rotate Left

Format: RL <op>

<op>: GPR

Operation: C ← <op>[7], <op> ← {<op>[6:0], <op>[7]}

RL rotates the value of <op> to the left and stores the result back into <op>.
The original MSB of <op> is copied into carry (C).

Flags:
C: set if the MSB of <op> (before rotating) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after rotating) is 1. Reset if not.

Example: Given: R0 = 01001010b, R1 = 10100101b

```
RL      R0                  // N flag is set to ‘1’, R0 ← 10010100b

RL      R1                  // C flag is set to ‘1’, R1 ← 01001011b
```

RLC — Rotate Left with Carry

Format: RLC <op>

<op>: GPR

Operation: C ← <op>[7], <op> ← {<op>[6:0], C}

RLC rotates the value of <op> to the left and stores the result back into <op>.
The original MSB of <op> is copied into carry (C), and the original C bit is copied into <op>[0].

Flags:
C: set if the MSB of <op> (before rotating) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after rotating) is 1. Reset if not.

Example: Given: R2 = A5h, if C = 0

```
RLC     R2                  // R2 ← 4Ah, C flag is set to ‘1’

RL      R0
RLC     R1
```

In the second example, assuming that register pair R1:R0 is 16-bit number, then RL and RLC are used for 16-bit rotate left operation. But note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit decrement, take care of the change of Z flag.

RR — Rotate Right

Format: RR <op>

<op>: GPR

Operation: C ← <op>[0], <op> ← {<op>[0], <op>[7:1]}

RR rotates the value of <op> to the right and stores the result back into <op>. The original LSB of <op> is copied into carry (C).

Flags:
C: set if the LSB of <op> (before rotating) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after rotating) is 1. Reset if not.

Example: Given: R0 = 01011010b, R1 = 10100101b

```
RR      R0                  // No change of flag, R0 ← 00101101b

RR      R1                  // C and N flags are set to '1', R1 ← 11010010b
```

RRC — Rotate Right with Carry

Format: RRC <op>

<op>: GPR

Operation: C ← <op>[0], <op> ← {C, <op>[7:1]}

RRC rotates the value of <op> to the right and stores the result back into <op>. The original LSB of <op> is copied into carry (C), and C is copied to the MSB.

Flags:
C: set if the LSB of <op> (before rotating) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after rotating) is 1. Reset if not.

Example: Given: R2 = A5h, if C = 0

```
RRC     R2                        // R2 ← 52h, C flag is set to '1'

RR      R0
RRC     R1
```

In the second example, assuming that register pair R1:R0 is 16-bit number, then RR and RRC are used for 16-bit rotate right operation. But note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit decrement, take care of the change of Z flag.

SBC — Subtract with Carry

Format: SBC <op1>, <op2>

<op1>: GPR
<op2>: adr:8, GPR

Operation: <op1> ← <op1> + ~<op2> + C

SBC computes (<op1> - <op2>) when there is carry and (<op1> - <op2> - 1) when there is no carry.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated.
N: set if result is negative. Reset if not.

Example:

```
SBC     R0, 80h             // If eid = 0, R0 ← R0 + ~DM[0080h] + C
                            // If eid = 1, R0 ← R0 + ~DM[IDH:80h] + C

SBC     R0, R1              // R0 ← R0 + ~R1 + C

SUB     R0, R2
SBC     R1, R3
```

In the last two instructions, assuming that register pair R1:R0 and R3:R2 are 16-bit signed or unsigned numbers. Even if the result of “ADD R0, R2” is not zero, zero (Z) flag can be set to ‘1’ if the result of “SBC R1,R3” is zero. Note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit addition, take care of the change of Z flag.

SL — Shift Left

Format: SL <op>

<op>: GPR

Operation: C ← <op>[7], <op> ← {<op>[6:0], 0}

SL shifts <op> to the left by 1 bit. The MSB of the original <op> is copied into carry (C).

Flags:
C: set if the MSB of <op> (before shifting) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after shifting) is 1. Reset if not.

Example: Given: R0 = 01001010b, R1 = 10100101b

```
SL      R0                      // N flag is set to ‘1’, R0 ← 10010100b

SL      R1                      // C flag is set to ‘1’, R1 ← 01001010b
```

SLA — Shift Left Arithmetic

Format: SLA <op>

<op>: GPR

Operation: C ← <op>[7], <op> ← {<op>[6:0], 0}

SLA shifts <op> to the left by 1 bit. The MSB of the original <op> is copied into carry (C).

Flags:

C: set if the MSB of <op> (before shifting) is 1. Reset if not.
Z: set if result is zero. Reset if not.
V: set if the MSB of the result is different from C. Reset if not.
N: set if the MSB of <op> (after shifting) is 1. Reset if not.

Example: Given: R0 = AAh

```
SLA     R0                          // C, V, N flags are set to ‘1’, R0 ← 54h
```

SR — Shift Right

Format: SR <op>

<op>: GPR

Operation: C ← <op>[0], <op> ← {0, <op>[7:1]}

SR shifts <op> to the right by 1 bit. The LSB of the original <op> (i.e., <op>[0]) is copied into carry (C).

Flags:
C: set if the LSB of <op> (before shifting) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after shifting) is 1. Reset if not.

Example: Given: R0 = 01011010b, R1 = 10100101b

```
SR      R0                  // No change of flags, R0 ← 00101101b

SR      R1                  // C flag is set to ‘1’, R1 ← 01010010b
```

SRA — Shift Right Arithmetic

Format: SRA <op>

<op>: GPR

Operation: C ← <op>[0], <op> ← {<op>[7], <op>[7:1]}

SRA shifts <op> to the right by 1 bit while keeping the sign of <op>. The LSB of the original <op> (i.e., <op>[0]) is copied into carry (C).

Flags: **C:** set if the LSB of <op> (before shifting) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after shifting) is 1. Reset if not.

NOTE: SRA keeps the sign bit or the MSB (<op>[7]) in its original position. If SRA is executed 'N' times, N significant bits will be set, followed by the shifted bits.

Example: Given: R0 = 10100101b

```
SRA     R0                      // C, N flags are set to '1', R0 ← 11010010b
SRA     R0                      // N flag is set to '1', R0 ← 11101001b
SRA     R0                      // C, N flags are set to '1', R0 ← 11110100b
SRA     R0                      // N flags are set to '1', R0 ← 11111010b
```

STOP — Stop Operation (pseudo instruction)

Format: STOP

Operation: The STOP instruction stops the both the CPU clock and system clock and causes the microcontroller to enter the STOP mode. In the STOP mode, the contents of the on-chip CPU registers, peripheral registers, and I/O port control and data register are retained. A reset operation or external or internal interrupts can release stop mode. The STOP instruction is a pseudo instruction. It is assembled as “SYS #0Ah”, which generates the SYSCP[7-0] signals. These signals are decoded and stop the operation.

NOTE: The next instruction of STOP instruction is executed, so please use the NOP instruction after the STOP instruction.

Example:

```
STOP
NOP
NOP
NOP
•
•
•
```

In this example, the NOP instructions provide the necessary timing delay for oscillation stabilization before the next instruction in the program sequence is executed. Refer to the timing diagrams of oscillation stabilization, as described in Figure 18-3, 18-4

SUB — Subtract

Format: SUB <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> ← <op1> + ~<op2> + 1

SUB adds the value of <op1> with the 2's complement of <op2> to perform subtraction on <op1> and <op2>

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: IDH:IDL0 = 0150h, DM[0143h] = 26h, R0 = 52h, R1 = 14h, eid = 1

```
SUB     R0, 43h              // R0 ← R0 + ~DM[0143h] + 1 = 2Ch

SUB     R1, #16h             // R1 ← FEh, N flag is set to ‘1’

SUB     R0, R1               // R0 ← R0 + ~R1 + 1 = 3Eh

SUB     R0, @ID0+1           // R0 ← R0 + ~DM[0150h] + 1, IDL0 ← 51h
SUB     R0, @[ID0-2]         // R0 ← R0 + ~DM[014Eh] + 1, IDL0 ← 4Eh
SUB     R0, @[ID0+3]!        // R0 ← R0 + ~DM[0153h] + 1, IDL0 ← 50h
SUB     R0, @[ID0-2]!        // R0 ← R0 + ~DM[014Eh] + 1, IDL0 ← 50h
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 8-5 for more detailed explanation about this addressing mode. The example in the SBC description shows how SUB and SBC can be used in pair to subtract a 16-bit number from another.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

SWAP — Swap

Format: SWAP <op1>, <op2>

<op1>: GPR
<op2>: SPR

Operation: <op1> ← <op2>, <op2> ← <op1>

SWAP swaps the values of the two operands.

Flags: –

NOTE: Among the SPRs, SR0 and SR1 can not be used as <op2>.

Example: Given: IDH:IDL0 = 8023h, R0 = 56h, R1 = 01h

```
SWAP    R1, IDH             // R1 ← 80h, IDH ← 01h
SWAP    R0, IDL0            // R0 ← 23h, IDL0 ← 56h
```

After execution of instructions, index registers IDH:IDL0 (ID0) have address 0156h.

SYS — System

Format: SYS #imm:8

Operation: SYS generates SYSCP[7:0] and nSYSID signals.

Flags: –

NOTE: Mainly used for system peripheral interfacing.

Example:

```
SYS       #0Ah

SYS       #05h
```

In the first example, statement “SYS #0Ah” is equal to STOP instruction and second example “SYS #05h” is equal to IDLE instruction. This instruction does nothing but increase PC by one and generates SYSCP[7:0] and nSYSID signals.

TM — Test Multiple Bits

Format: TM <op>, #imm:8

<op>: GPR

Operation: TM performs the bit-wise AND operation on <op> and imm:8 and sets the flags. The content of <op> is not changed.

Flags: **Z:** set if result is zero. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: R0 = 01001101b

```
TM          R0, #00100010b           // Z flag is set to '1'
```

XOR — Exclusive OR

Format: XOR <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> ← <op1> ^ <op2>

XOR performs the bit-wise exclusive-OR operation on <op1> and <op2> and stores the result in <op1>.

Flags: **Z:** set if result is zero. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: IDH:IDL0 = 8080h, DM[8043h] = 26h, R0 = 52h, R1 = 14h, eid = 1

```
XOR     R0, 43h              // R0 ← 74h

XOR     R1, #00101100b       // R1 ← 38h

XOR     R0, R1               // R0 ← 46h

XOR     R0, @ID0             // R0 ← R0 ^ DM[8080h], IDL0 ← 81h
XOR     R0, @[ID0-2]         // R0 ← R0 ^ DM[807Eh], IDL0 ← 7Eh
XOR     R0, @[ID0+3]!        // R0 ← R0 ^ DM[8083h], IDL0 ← 80h
XOR     R0, @[ID0-5]!        // R0 ← R0 ^ DM[807Bh], IDL0 ← 80h
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 8-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

NOTES

9 CLOCK CIRCUIT

OVERVIEW

The S3CB519/FB519 microcontroller has two oscillator circuits: a main system clock circuit and a subsystem clock circuit. The CPU and peripheral hardware operate at the system clock frequency supplied by these circuits. The maximum CPU clock frequency is determined by PCON register setting.

SYSTEM CLOCK CIRCUIT

The system clock circuit has the following components:

- External crystal or ceramic resonator oscillation source (or an external clock source)
- Oscillator stop and wake-up functions
- Programmable frequency divider for the CPU clock (f_{OSC} divided by 1, 2, 4, 8, 16, 32, 64, 128)
- System clock control register, PCON

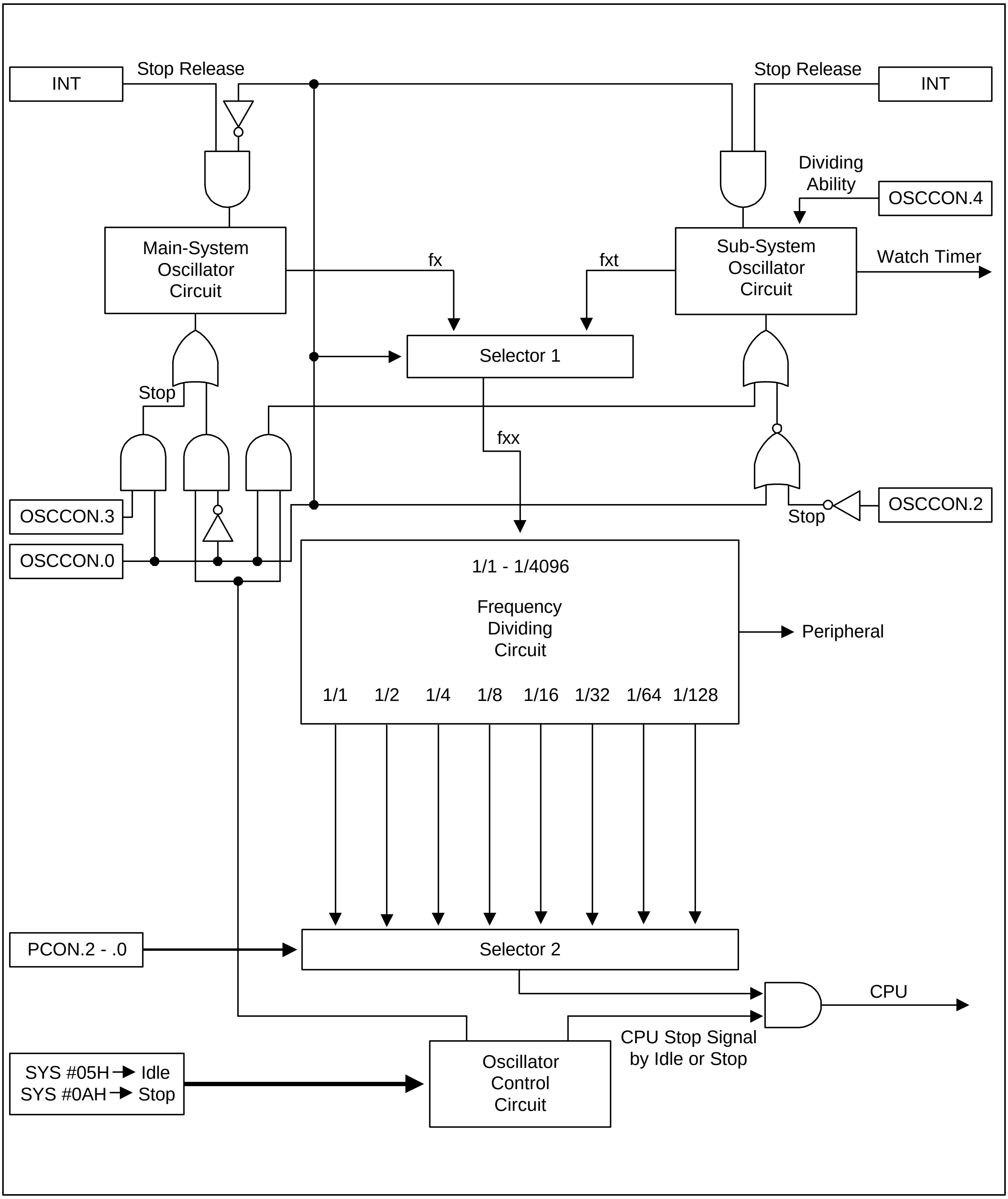


Figure 9-1. System Clock Circuit Diagram

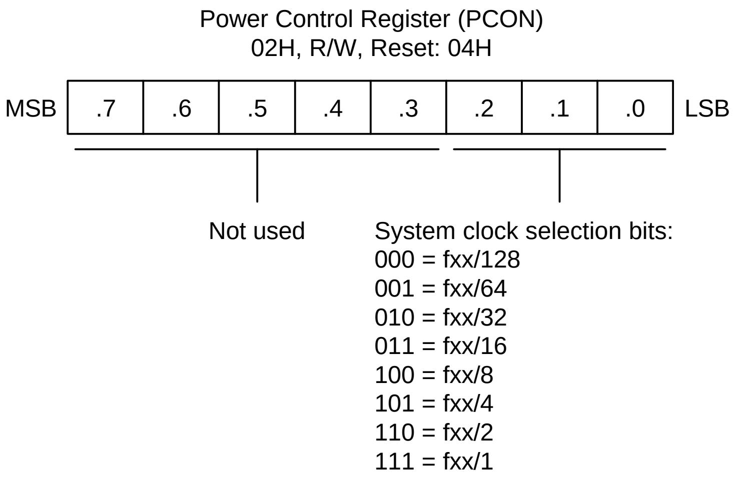
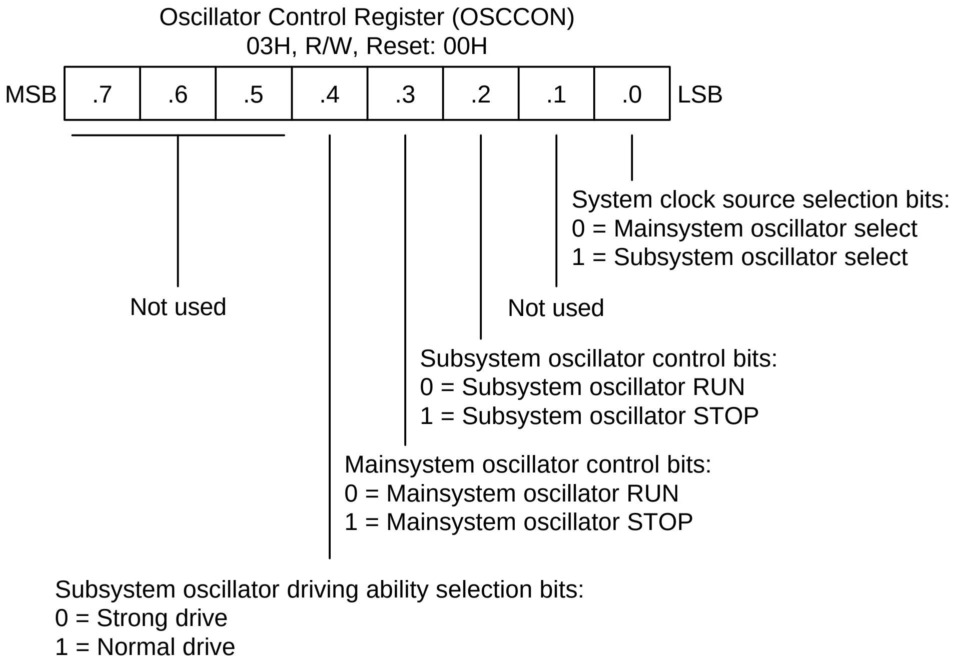


Figure 9-2. Power Control Register (PCON)


Oscillator Control Register (OSCCON)
03H, R/W, Reset: 00H
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
System clock source selection bits:
0 = Mainsystem oscillator select
1 = Subsystem oscillator select
Not used
Not used
Subsystem oscillator control bits:
0 = Subsystem oscillator RUN
1 = Subsystem oscillator STOP
Mainsystem oscillator control bits:
0 = Mainsystem oscillator RUN
1 = Mainsystem oscillator STOP
Subsystem oscillator driving ability selection bits:
0 = Strong drive
1 = Normal drive

NOTE: The oscillator selected by the OSCCON.0 can be stopped only by the "stop" instruction. It cannot be stopped by the OSCCON settings.

Figure 9-3. Oscillator Control Register (OSCCON)

NOTES

10 RESET AND POWER-DOWN

OVERVIEW

During a power-on reset, the voltage at V_{DD} goes to High level and the RESET pin is forced to Low level. The reset signal is input through a schmitt trigger circuit where it is then synchronized with the CPU clock. This procedure brings S3CB519/FB519 into a known operating status.

For the time for CPU clock oscillation to stabilize, the RESET pin must be held to low level for a minimum time interval after the power supply comes within tolerance. (For the minimum time interval, see the electrical characteristic).

In summary, the following sequence of events occurs during a reset operation:

— All interrupts are disabled.

— The watchdog function (basic timer) is enabled.

— Ports are set to input mode except port 5 which is set to output mode.

— Peripheral control and data registers are disabled and reset to their default hardware values.

— The program counter (PC) is loaded with the program reset address in the ROM, 00000H.

— When the programmed oscillation stabilization time interval has elapsed, the instruction stored in ROM location 00000H is fetched and executed.

NOTE

To program the duration of the oscillation stabilization interval, make the appropriate settings to the watchdog timer control register, WDTCON, before entering STOP mode.

NOTES

11 I/O PORTS

OVERVIEW

The S3CB519/FB519 has five I/O ports (P0–P4) for general I/O and one output port (P5) dedicated for the key-strobe with LCD segment data.

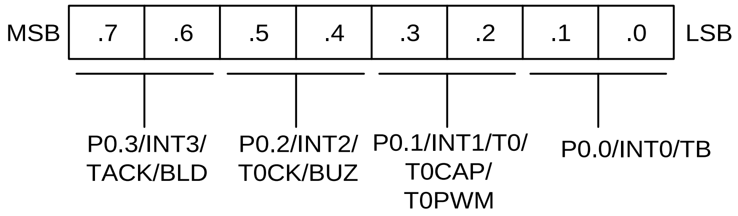
PORT 0

Two 8-bit control registers are used to configure the port 0 pins: P0CONH for pins P0.4–P0.6 and P0CONL for pins P0.0–P0.3. Each byte contains four bit-pairs and each bit-pair configures one pin. The P0CONH and the P0CONL registers also control the alternative functions.

For example, when bits 4 and 5 of P0CONL are “00”, P0.2 is selected for the input mode.
In this mode, you can set P0.2 as a normal input or an interrupt 2 or a timer 0 clock input by controlling P0INT and T0CON.

P0INT and P0EDGE registers control the interrupt functions for INT0–INT6.

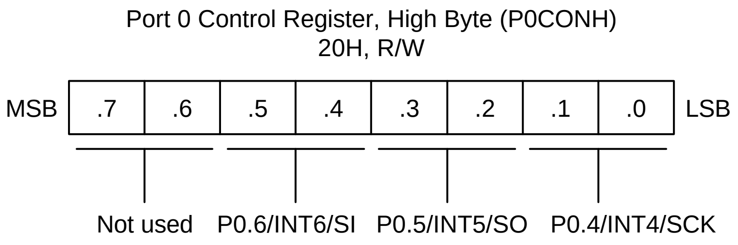
Port 0 Control Register, Low Byte (P0CONL)
21H, R/W



P0CONL bit-pair pin configuration settings:

0 0	Input mode (INT0 for P0.0, INT1/T0CAP for P0.1, INT2/T0CK for P0.2, INT3/TACK for P0.3)
0 1	Input mode, pull-up (INT0 for P0.0, INT1/T0CAP for P0.1, INT2/T0CK for P0.2, INT3/TACK for P0.3)
1 0	Alternative mode (TB for P0.0, T0/T0PWM for P0.1, BUZ for P0.2, BLD for P0.3)
1 1	Output mode, push-pull

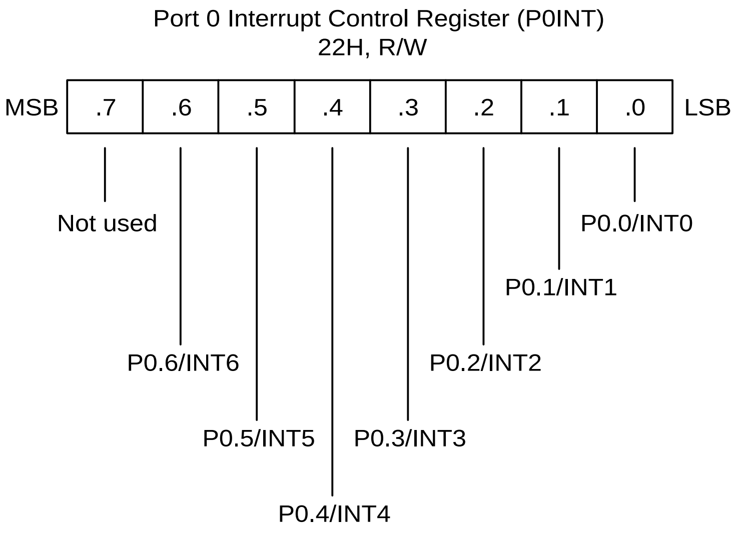
Figure 11-1. Port 0 Low-byte Control Register (P0CONL)



P0CONH bit-pair pin configuration settings:

0 0	Input mode (INT4/SCK input for P0.4, INT5 for P0.5, INT6/SI for P0.6)
0 1	Input mode, pull-up (INT4/SCK input for P0.4, INT5 for P0.5, INT6/SI for P0.6)
1 0	Alternative mode (SCK output for P0.4, SO for P0.5, High-impedance for P0.6)
1 1	Output mode, push-pull

Figure 11-2. Port 0 High-byte Control Register (P0CONH)



P0INT bit settings:

0	Disable interrupt
1	Enable interrupt

Figure 11-3. Port 0 Interrupt Control Register (P0INT)

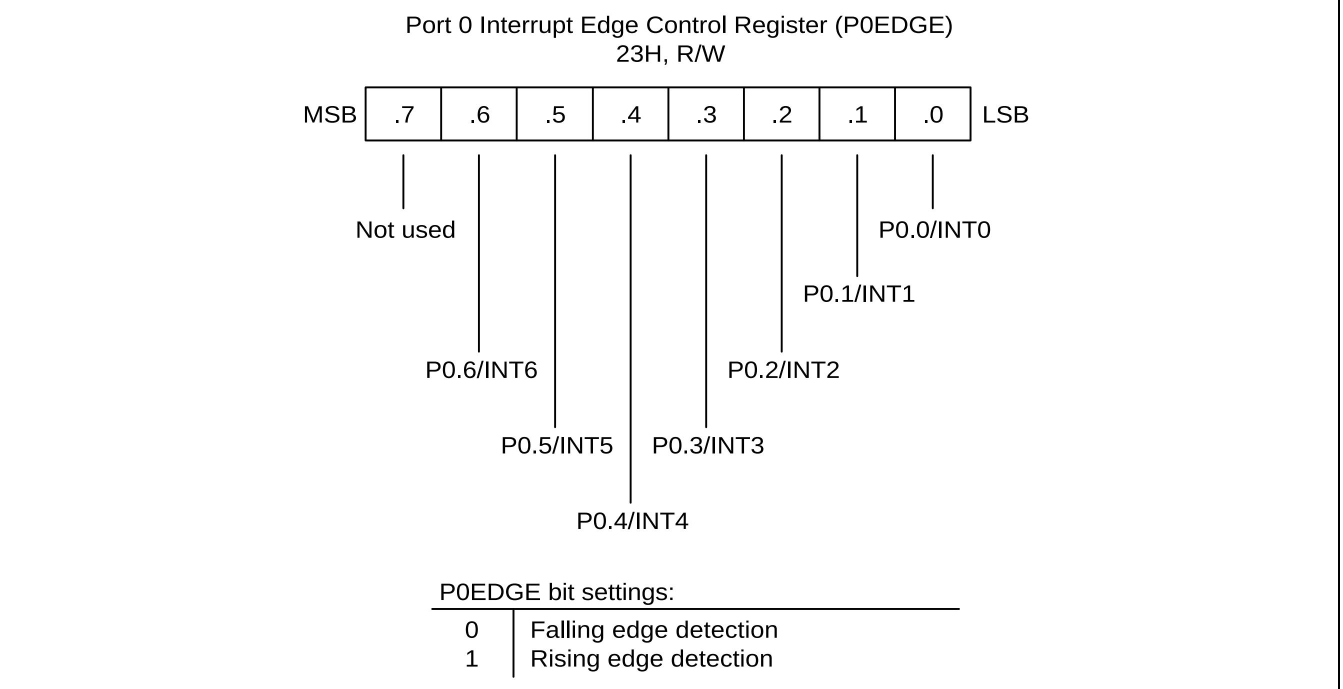
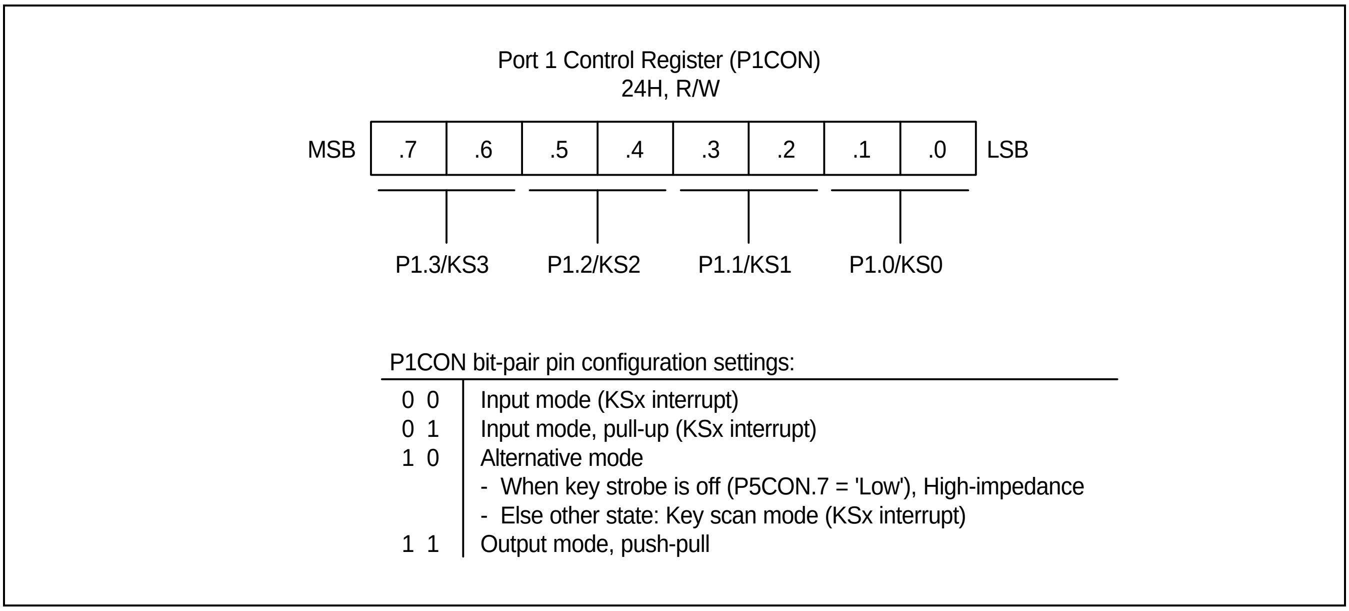


Figure 11-4. Port 0 Interrupt Edge Control Register (P0EDGE)

PORT 1

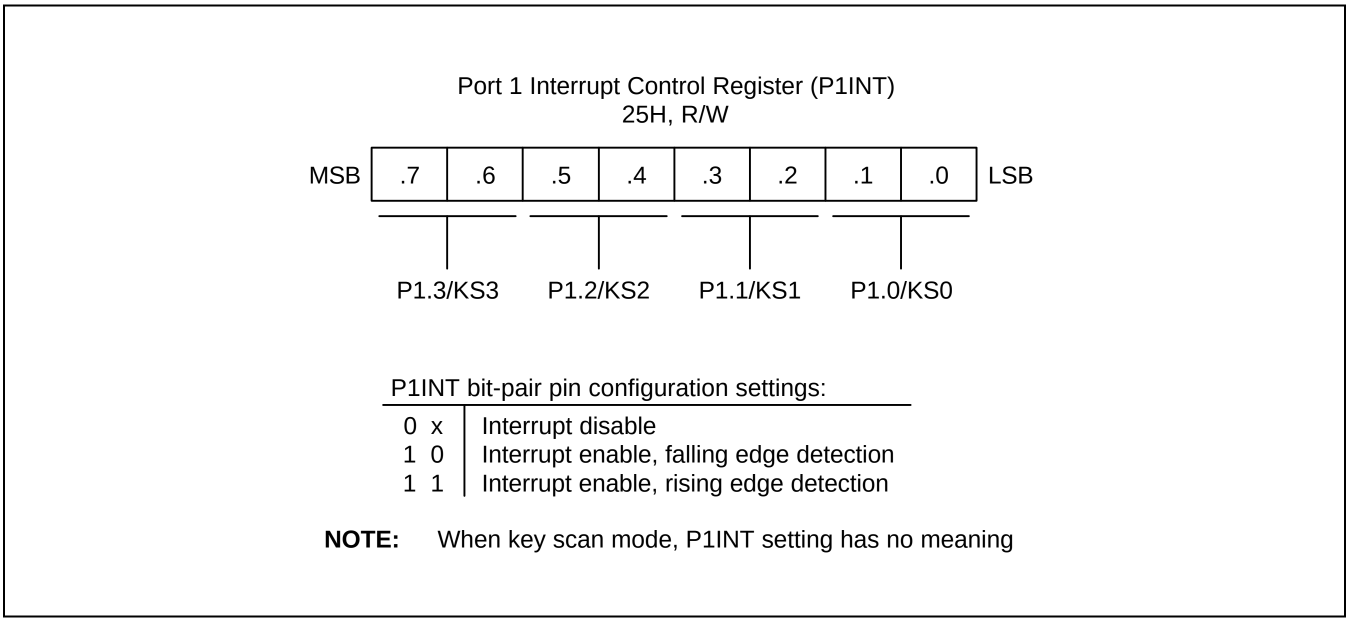
P1CON contains four bit-pairs and bit-pair configures one pin. P1INT controls the interrupt function for KS0–KS3. When the alternative mode is selected, KS0–KS3 can be used as key scan inputs with P5 (shared with LCD SEG) strobe.



P1CON bit-pair pin configuration settings:

0 0	Input mode (KSx interrupt)
0 1	Input mode, pull-up (KSx interrupt)
1 0	Alternative mode - When key strobe is off (P5CON.7 = 'Low'), High-impedance - Else other state: Key scan mode (KSx interrupt)
1 1	Output mode, push-pull

Figure 11-5. Port 1 Control Register (P1CON)



P1INT bit-pair pin configuration settings:

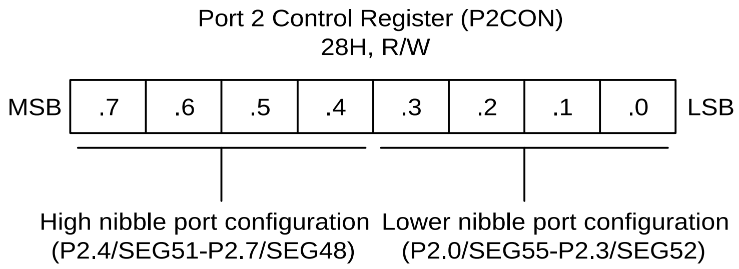
0 x	Interrupt disable
1 0	Interrupt enable, falling edge detection
1 1	Interrupt enable, rising edge detection

NOTE: When key scan mode, P1INT setting has no meaning

Figure 11-6. Port 1 Interrupt Control Register (P1INT)

PORT 2

P2CON and P3CON contain two nibbles each and each nibble configures four pins.
Port 2 is shared by SEG48–SEG55, and Port 3 is shared by SEG40–SEG47.

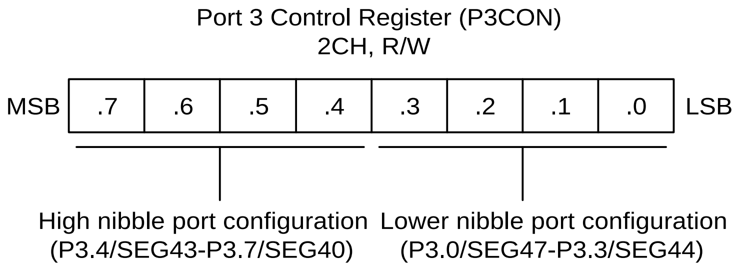

Port 2 Control Register (P2CON)
28H, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
High nibble port configuration
(P2.4/SEG51-P2.7/SEG48)
Lower nibble port configuration
(P2.0/SEG55-P2.3/SEG52)

P2CON pin configuration settings: (bit7, 6, 5, 4 or bit3, 2, 1, 0)

x000	Input mode
x001	Input mode, pull-up
x010	Output mode, push-pull
x011	Output mode, open-drain
x1xx	LCD Segment (SEG55-SEG52 for P2.0-P2.3 or SEG51-SEG48 for P2.4-P2.7)

Figure 11-7. Port 2 Control Register (P2CON)

PORT 3


Port 3 Control Register (P3CON)
2CH, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
High nibble port configuration
(P3.4/SEG43-P3.7/SEG40)
Lower nibble port configuration
(P3.0/SEG47-P3.3/SEG44)

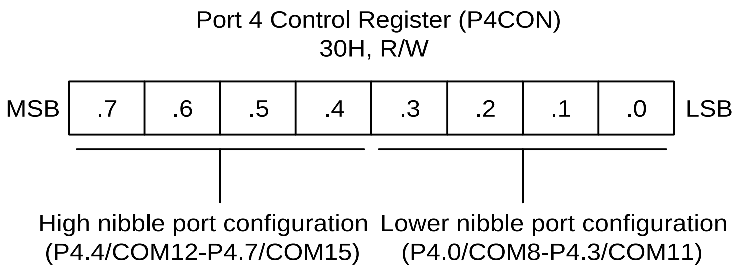
P3CON pin configuration settings: (bit7, 6, 5, 4 or bit3, 2, 1, 0)

x000	Input mode
x001	Input mode, pull-up
x010	Output mode, push-pull
x011	Output mode, open-drain
x1xx	LCD Segment (SEG47-SEG44 for P3.0-P3.3 or SEG43-SEG40 for P3.4-P3.7)

Figure 11-8. Port 3 Control Register (P3CON)

PORT 4

P4CON contains two nibbles and each nibble configures four pins.
Port 4 is shared by COM8–COM15, and I/O and COM switching are up to LMOD register.


Port 4 Control Register (P4CON)
30H, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
High nibble port configuration
(P4.4/COM12-P4.7/COM15)
Lower nibble port configuration
(P4.0/COM8-P4.3/COM11)

P4CON pin configuration settings: (bit7, 6, 5, 4 or bit3, 2, 1, 0)

xx00	Input mode
xx01	Input mode, pull-up
xx10	Output mode, push-pull
xx11	Output mode, open-drain

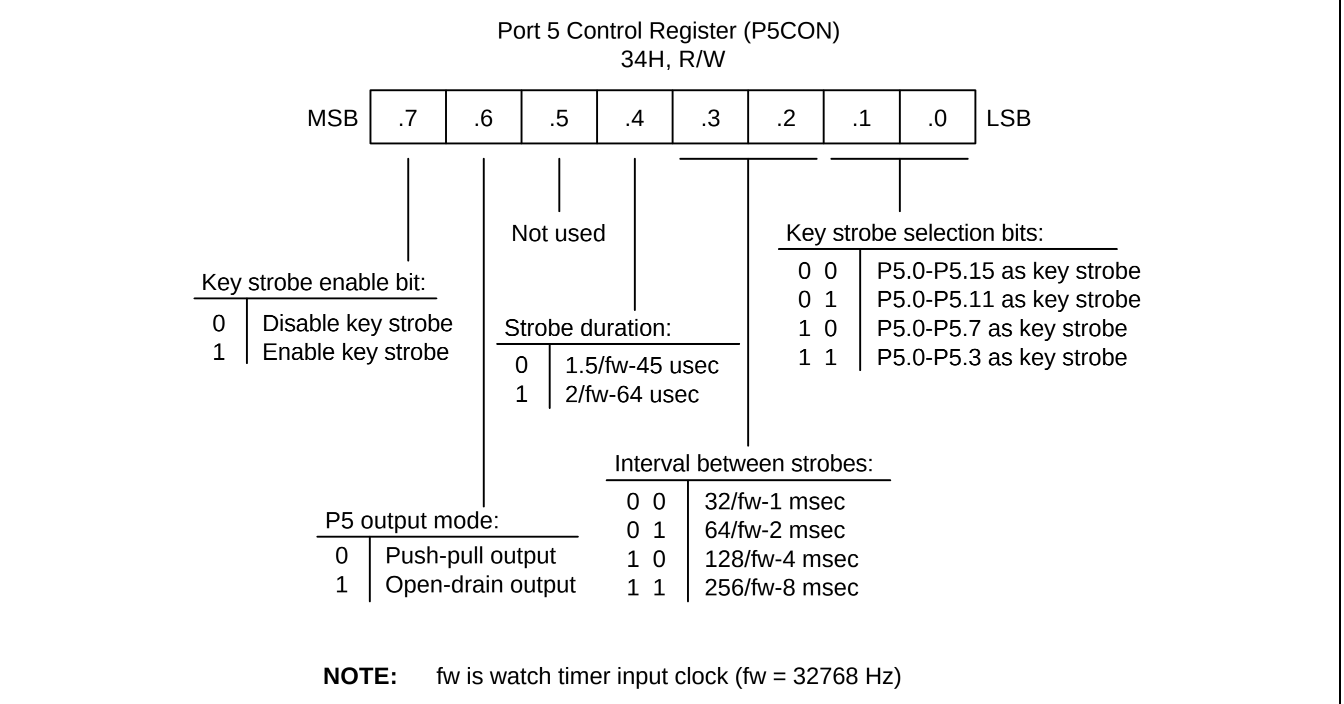
NOTE: P4.0-P4.7 can be converted to COM8-COM15 according to LMOD setting.
If only COM0-COM11 are selected as COM, COM12-COM15 are normal ports
If only COM0-COM7 are selected as COM, COM8-COM15 are normal ports.

Figure 11-9. Port 4 Control Register (P4CON)

PORT 5

Port 5 , which has 15 pins, can be controlled by P5CON but cannot be used as normal I/O.
Port 5 is shared by SEG pins and makes the key-strobe. (for details, see LCD chapter).
When port 1 is selected as the alternative mode (key scan input) and the key strobe function of port 5 is enabled, port 5 data register has the key-strobe value of the time when the key scan interrupt occurs.

For example, when P5.3 outputs strobe and any of port 1 are “Low”-state, forcing the key scan interrupt, port 5 data register has the value "3". For P5.9, port 5 data register has "9".



NOTE: fw is watch timer input clock (fw = 32768 Hz)

Figure 11-10. Port 5 Control Register (P5CON)

NOTES

12 BASIC TIMER/WATCHDOG TIMER

OVERVIEW

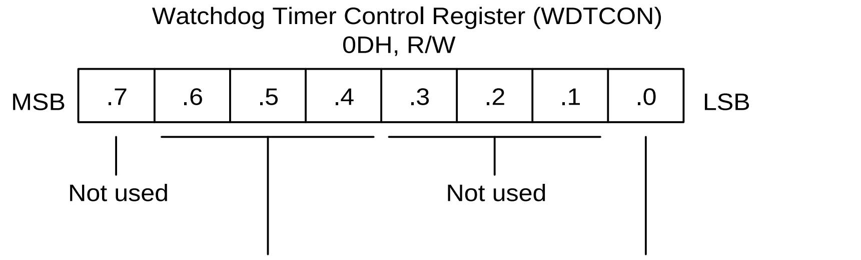
WDTCON controls basic timer clock selection and watchdog timer clear bit.

Basic timer is used in two different ways**:**

- As a clock source to watchdog timer to provide an automatic reset mechanism in the event of a system malfunction (When watchdog function is enabled in ROM code option)
- To signal the end of the required oscillation stabilization interval after a reset or stop mode release.

The reset value of basic timer clock selection bits is decided by the ROM code option. (see the section on ROM code option for details). After reset, programmer can select the basic timer input clock using WDTCON.

Watchdog timer provides an automatic reset mechanism in the event of a system malfunction (When watchdog function is enabled in ROM code option)
When watchdog function is enabled by the ROM code option, programmer must set WDTCON.0 periodically within every 2048 × basic timer input clock time to prevent system reset.

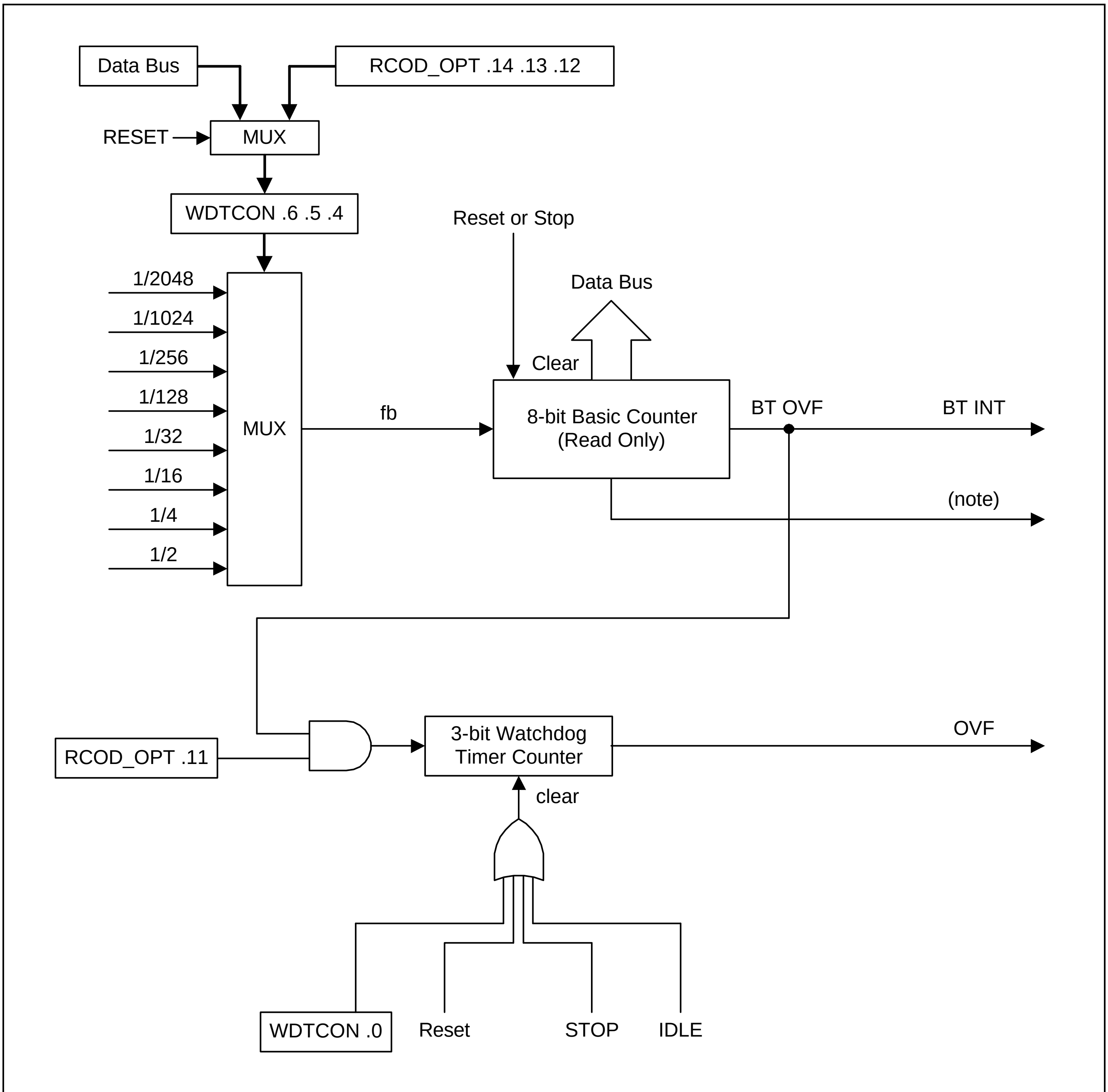


Basic timer counter clock selection bits:
000 = fxx/2
001 = fxx/4
010 = fxx/16
011 = fxx/32
100 = fxx/128
101 = fxx/256
110 = fxx/1024
111 = fxx/2048

Watchdog timer clear bit:
0 = Not applicable
1 = clear watchdog timer counter

Figure 12-1. Watchdog Timer Control Register (WDTCON)

BLOCK DIAGRAM



NOTE: CPU start signal (32/fb)
(Power down release)

Figure 12-2. Basic Timer & Watchdog Timer Functional Block Diagram

13 WATCH TIMER

OVERVIEW

Watch timer functions include real-time and watch-time measurements.
After the watch timer starts and time elapses, the watch timer interrupt is automatically set to "1", and interrupt requests commence in 3.91 ms, 0.25 s, 0.5 s or 1 second.
The watch timer can generate a steady 0.5 kHz, 1 kHz, 2 kHz or 4 kHz signal to the BUZ output when the main system clock frequency is 4.195 MHz. The watch timer supplies the clock frequency for the LCD controller (f_{LCD}) and BLD. Therefore, if the watch timer is disabled, the LCD and BLD controller do not operate.

— Real-time and Watch-time measurements

— Clock source generation for LCD controller

— Buzzer output frequency generator

Table 13-1. Watch Timer Control Register (WTCON): 8-Bit R/W

Bit Name	Values		Function	Address
WTCON.7	–		Not used	70H
WTCON.6	–		Not used	
WTCON .5–.4	0	0	0.5 kHz buzzer (BUZ) signal output	
	0	1	1 kHz buzzer (BUZ) signal output	
	1	0	2 kHz buzzer (BUZ) signal output	
	1	1	4 kHz buzzer (BUZ) signal output	
WTCON .3–.2	0	0	Set watch timer interrupt to 1 S.	
	0	1	Set watch timer interrupt to 0.5 S.	
	1	0	Set watch timer interrupt to 0.25 S.	
	1	1	Set watch timer interrupt to 3.91 mS.	
WTCON.1	0		Selects (fx/128 or fx/64) as the watch timer clock	
	1		Selects the subsystem clock as watch timer clock	
WTCON.0	0		Stops the watch timer counter; clears the frequency dividing circuits	
	1		Runs the watch timer counter	

NOTE: Main system clock frequency (fx) is assumed to be 4.195 MHz.

WATCH TIMER CIRCUIT DIAGRAM

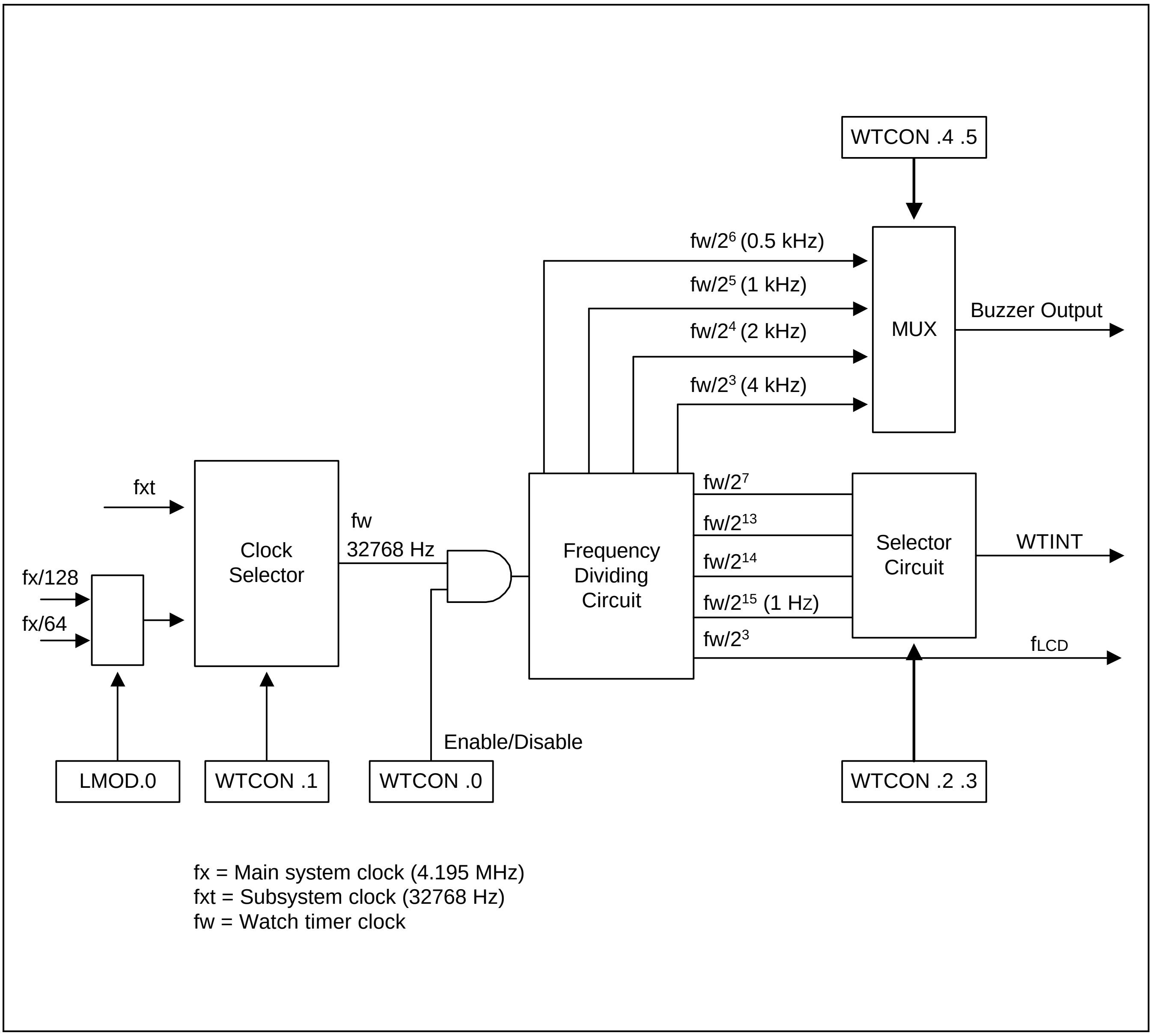
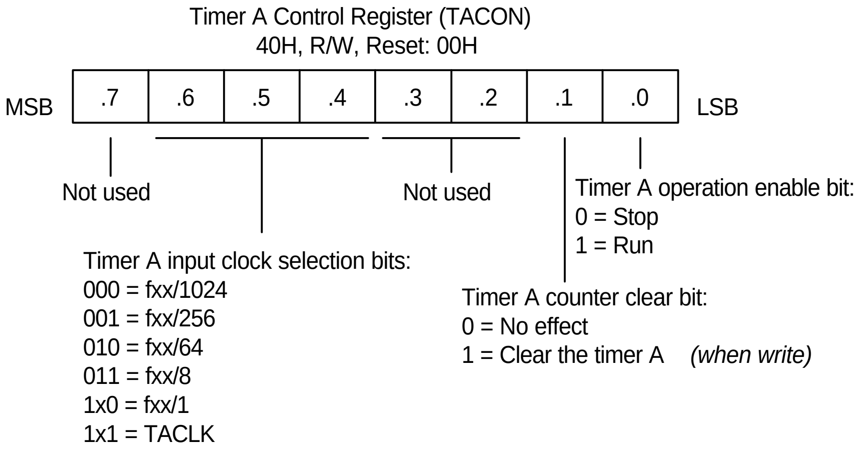


Figure 13-1. Watch Timer Circuit Diagram

14 16-BIT TIMER (8-BIT TIMER A & B)

OVERVIEW

The 16-bit timer is used in one 16-bit timer or two 8-bit timers. When Bit 2 of TBCON is "1" , it operates as one 16-bit timer. When it is “0”, it operates as two 8-bit timers. When it operates as one 16-bit timer, the TBCNT’s clock source can be selected by setting TBCON.3. If TBCON.3 is “0”, the timer A’s overflow would be TBCNT’s clock source. If it is “1”, the timer A’s interval out would be TBCNT’s clock source. The timer clock source can be selected by the S/W.



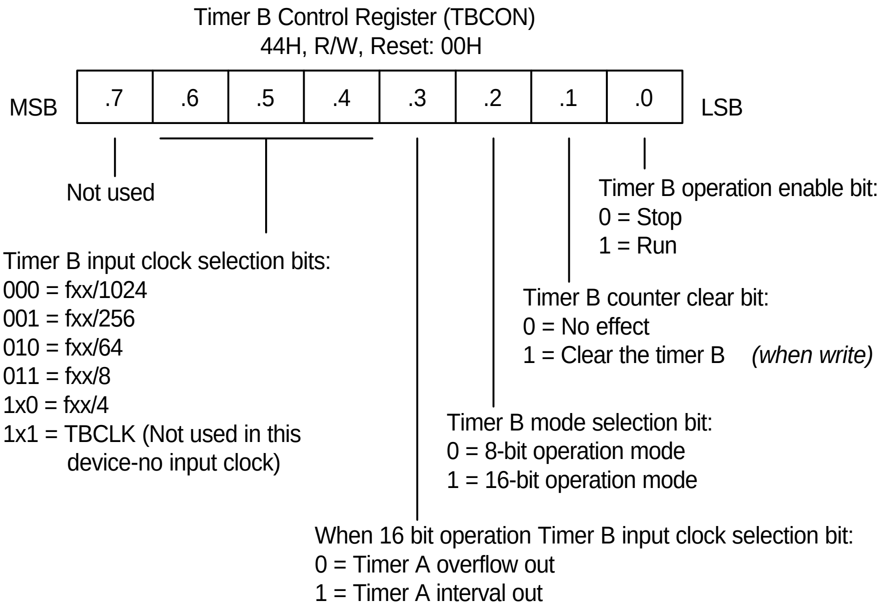
NOTE: 8-Bit Timer A is only available when TBCON.2 is setted "0" for 8-Bit operation mode.

Figure 14-1. Timer A Control Register (TACON)

INTERVAL TIMER FUNCTION

The timer A&B module can generate an interrupt: the Timer A and/or Timer B match interrupt (TAINT, TBINT). In interval timer mode, a match signal is generated when the counter value is identical to the value written to the reference data register, TADATA/TBDATA. The match signal generates Timer A and/or Timer B match interrupt and clears the counter.

TB pin can be toggled whenever the timer B match interrupt occurs if I/O port setting is appropriate.


Timer B Control Register (TBCON)
44H, R/W, Reset: 00H
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Not used
Timer B input clock selection bits:
000 = fxx/1024
001 = fxx/256
010 = fxx/64
011 = fxx/8
1x0 = fxx/4
1x1 = TBCLK (Not used in this device-no input clock)
Timer B operation enable bit:
0 = Stop
1 = Run
Timer B counter clear bit:
0 = No effect
1 = Clear the timer B (when write)
Timer B mode selection bit:
0 = 8-bit operation mode
1 = 16-bit operation mode
When 16 bit operation Timer B input clock selection bit:
0 = Timer A overflow out
1 = Timer A interval out

NOTE: At 16-bit operation mode 16-bit counter clock input is selected by TACON .6, .5, .4

Figure 14-2. Timer B Control Register (TBCON)

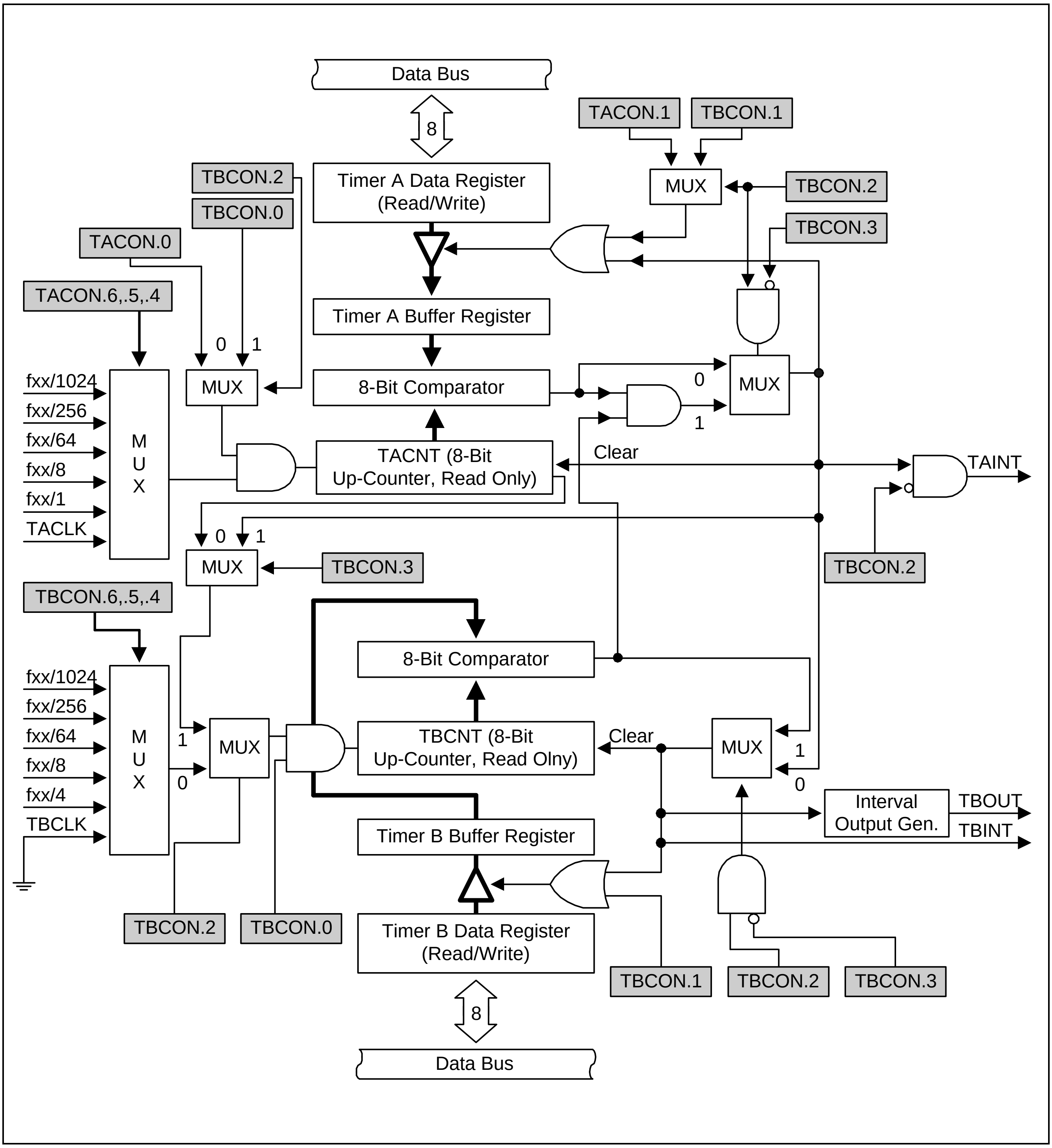


Figure 14-3. Timer A, B Function Block Diagram

NOTES

15 8-BIT TIMER (TIMER 0)

OVERVIEW

The 8-bit timer 0 is an 8-bit general-purpose timer/counter. Timer 0 has three operating modes, one of which you select using the appropriate T0CON setting:

— Interval timer mode (Toggle output at T0 pin)

— Capture input mode with a rising or falling edge trigger at the T0CAP pin

— PWM mode (T0PWM)

FUNCTION DESCRIPTION

Timer 0 Interrupts

The Timer 0 module can generate two interrupts: the Timer 0 overflow interrupt (T0OVF), and the Timer 0 match/ capture interrupt (T0INT).

Interval Timer Function

The Timer 0 module can generate an interrupt: the Timer 0 match interrupt (T0INT).
In interval timer mode, a match signal is generated(,) and T0 is toggled when the counter value is identical to the value written to the T0 reference data register, T0DATA. The match signal generates a Timer 0 match interrupt and clears the counter.
If, for example, you write the value 10H to T0DATA and 0AH to T0CON, the counter will increment until it reaches 10H. At this point, the T0 interrupt request is generated and the counter value is reset and counting resumes.

Pulse Width Modulation Mode

Pulse width modulation (PWM) mode lets you program the width (duration) of the pulse that is output at the T0PWM pin. As in interval timer mode, a match signal is generated when the counter value is identical to the value written to the Timer 0 data register. In PWM mode, however, the match signal does not clear the counter but can generate a match interrupt. The counter runs continuously, overflowing at FFH, and then repeats the incrementing from 00H. Whenever an overflow occurs, an overflow(OVF) interrupt can be generated.
Although you can use the match or the overflow interrupt in PWM mode, interrupts are not typically used in PWM-type applications. Instead, the pulse at the T0PWM pin is held to High level as long as the reference data value is less than or equal to (≤) the counter value, and then the pulse is held to Low level for as long as the data value is greater than (>) the counter value. One pulse width is equal to $t_{CLK} \times 256$.

Capture Mode

In capture mode, a signal edge that is detected at the T0CAP pin opens a gate and loads the current counter value into the T0 data register. You can select the rising or falling edges to trigger this operation.
Timer 0 also gives you capture input source: the signal edge at the T0CAP pin. You select the capture input by setting the value of the Timer 0 capture input selection bit in the port control register.
Both kinds of Timer 0 interrupts can be used in capture mode: the Timer 0 overflow interrupt is generated whenever a counter overflow occurs; the Timer 0 match/capture interrupt is generated whenever the counter value is loaded into the T0 data register.
By reading the captured data value in T0DATA and assuming a specific value for the Timer 0 clock frequency, you can calculate the pulse width (duration) of the signal that is being input at the T0CAP pin.

TIMER 0 CONTROL REGISTER (T0CON)

You use the Timer 0 control register, T0CON, to

- Select the Timer 0 operating mode (interval timer, capture mode, or PWM mode)
- Select the Timer 0 input clock frequency
- Clear the Timer 0 counter, T0CNT
- Enable the Timer 0 overflow interrupt or Timer 0 match/capture interrupt

A reset clears T0CON to '00H'. This sets Timer 0 to normal interval timer mode, selects an input clock frequency of f_{OSC}/1024, and disables all Timer 0 interrupts. You can clear the Timer 0 counter at any time during normal operation by writing a "1" to T0CON.3.

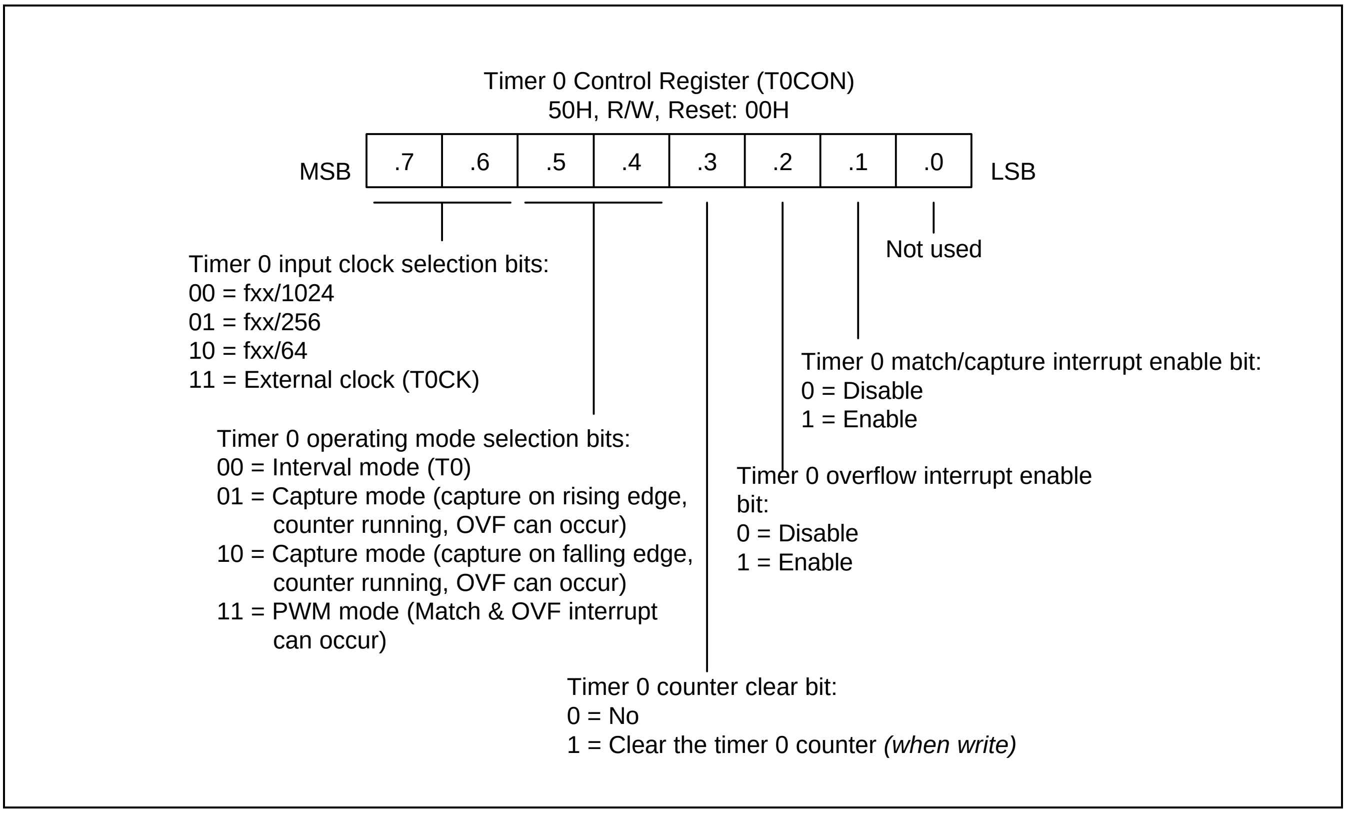

Timer 0 Control Register (T0CON)
50H, R/W, Reset: 00H
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Not used
Timer 0 input clock selection bits:
00 = fxx/1024
01 = fxx/256
10 = fxx/64
11 = External clock (T0CK)
Timer 0 operating mode selection bits:
00 = Interval mode (T0)
01 = Capture mode (capture on rising edge, counter running, OVF can occur)
10 = Capture mode (capture on falling edge, counter running, OVF can occur)
11 = PWM mode (Match & OVF interrupt can occur)
Timer 0 match/capture interrupt enable bit:
0 = Disable
1 = Enable
Timer 0 overflow interrupt enable bit:
0 = Disable
1 = Enable
Timer 0 counter clear bit:
0 = No
1 = Clear the timer 0 counter *(when write)*

Figure 15-1. Timer 0 Control Register (T0CON)

BLOCK DIAGRAM

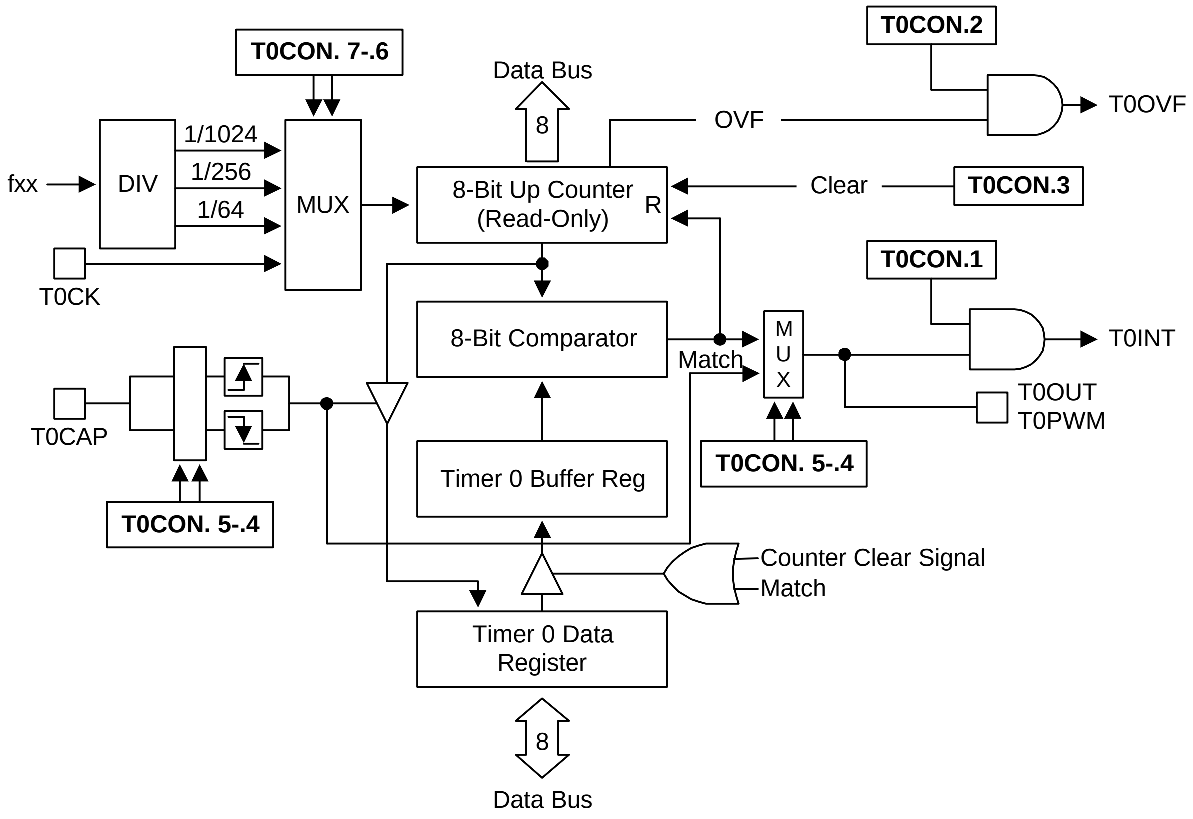


Figure 15-2. Timer 0 Functional Block Diagram

16 SERIAL I/O INTERFACE

OVERVIEW

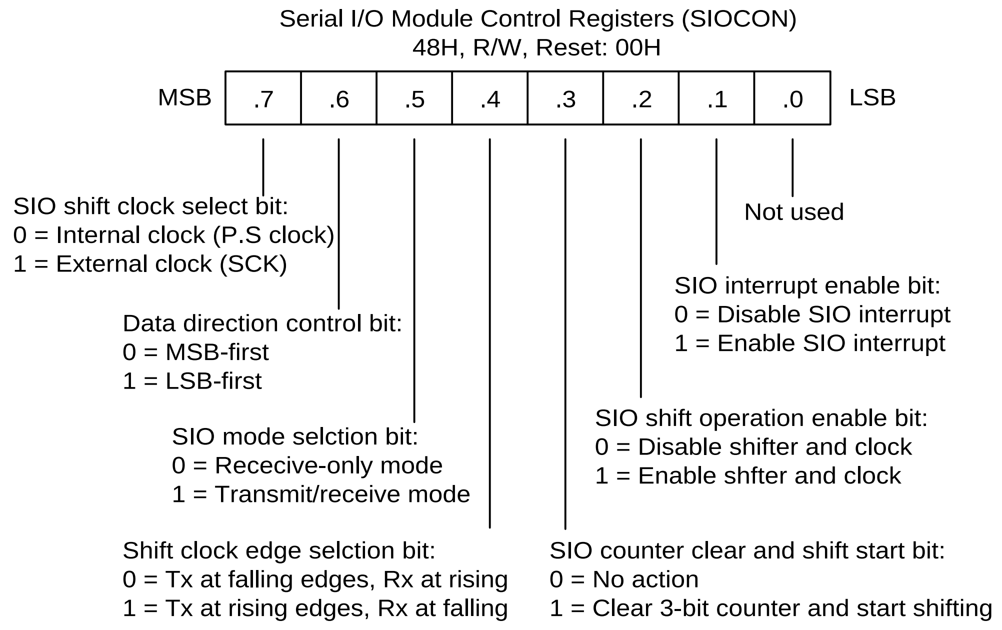
The SIO module can transmit or receive 8-bit serial data at a frequency determined by its corresponding control register settings. To ensure flexible data transmission rates, you can select an internal or external clock source.

PROGRAMMING PROCEDURE

To program the SIO modules, follow these basic steps:

1. Configure the I/O pins at port (SO,SCK, SI) by loading the appropriate value to the P0CONH register, if necessary.
2. Load an 8-bit value to the SIOCON register to properly configure the serial I/O module. In this operation, SIOCON.2 must be set to "1" to enable the data shifter.
3. For interrupt generation, set the serial I/O interrupt enable bit (SIOCON.1) to "1".
4. When you transmit data to the serial buffer, write data to SIODATA and set SIOCON.3 to 1, the shift operation starts.
5. When the shift operation (transmit/receive) is completed, the SIO pending bit is set to "1", and a SIO interrupt request is generated.

SIO CONTROL REGISTER (SIOCON)



SIO shift clock select bit:
0 = Internal clock (P.S clock)
1 = External clock (SCK)

Data direction control bit:
0 = MSB-first
1 = LSB-first

SIO mode selction bit:
0 = Rececive-only mode
1 = Transmit/receive mode

Shift clock edge selction bit:
0 = Tx at falling edges, Rx at rising
1 = Tx at rising edges, Rx at falling

SIO counter clear and shift start bit:
0 = No action
1 = Clear 3-bit counter and start shifting

SIO shift operation enable bit:
0 = Disable shifter and clock
1 = Enable shfter and clock

SIO interrupt enable bit:
0 = Disable SIO interrupt
1 = Enable SIO interrupt

Not used

Figure 16-1. Serial I/O Control Register (SIOCON)

SIO PRE-SCALER REGISTER (SIOPS)

The values stored in the SIO pre-scaler registers, SIOPS, lets you determine the SIO clock rate (baud rate) as follows:

Baud rate = Input clock/(Pre-scaler value + 1), or SCLK input clock

where the input clock is fxx/4

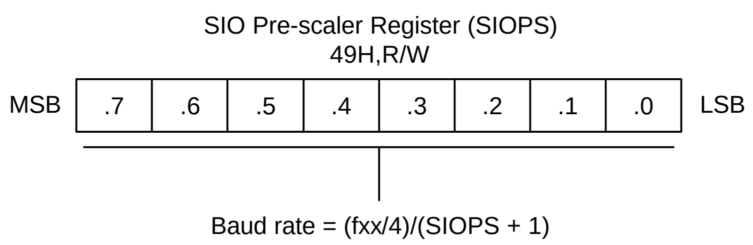

SIO Pre-scaler Register (SIOPS)
49H,R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Baud rate = (fxx/4)/(SIOPS + 1)

Figure 16-2. SIO Pre-scaler Register (SIOPS)

BLOCK DIAGRAM

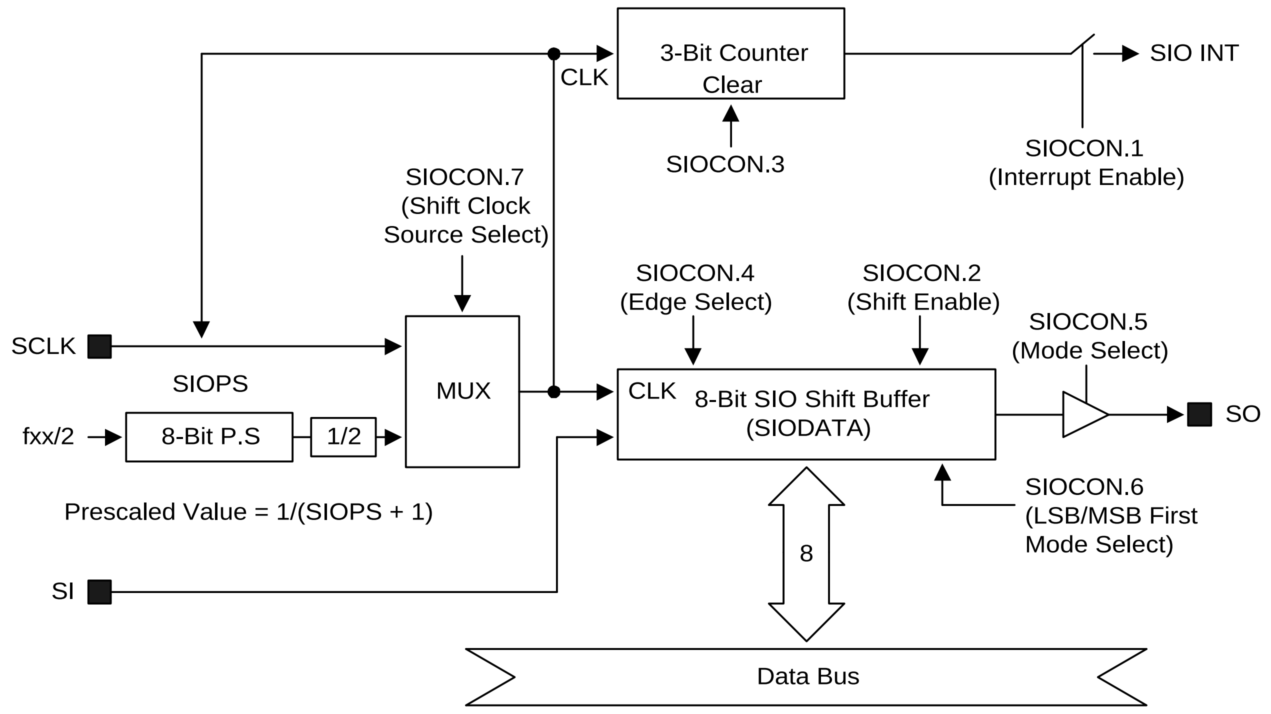


Figure 16-3. SIO Functional Block Diagram

SERIAL I/O TIMING DIAGRAMS

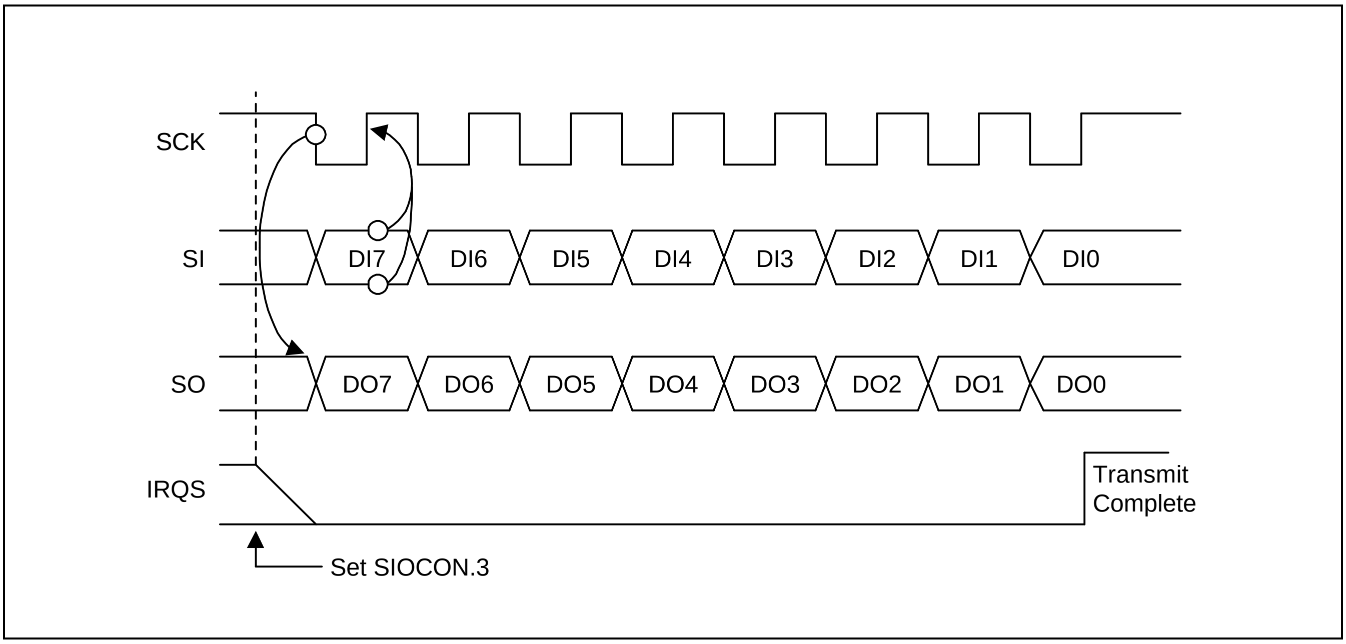


Figure 16-4. Serial I/O Timing in Transmit/Receive Mode (Tx at falling, SIOCON.4 = 0)

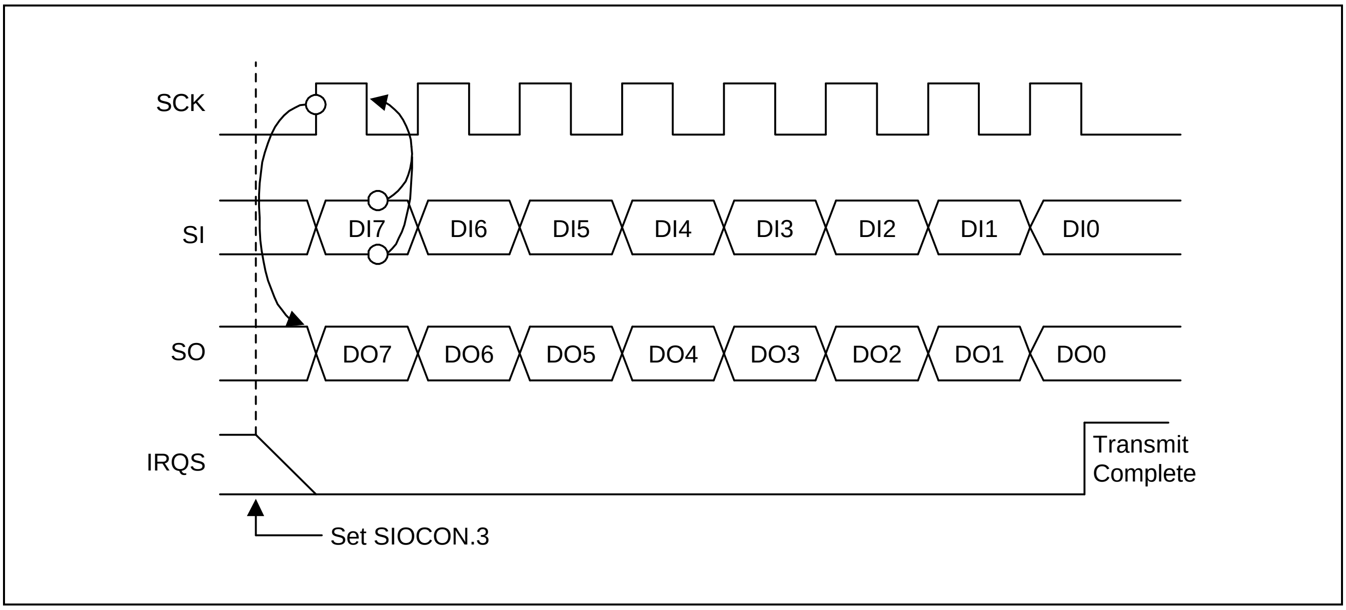


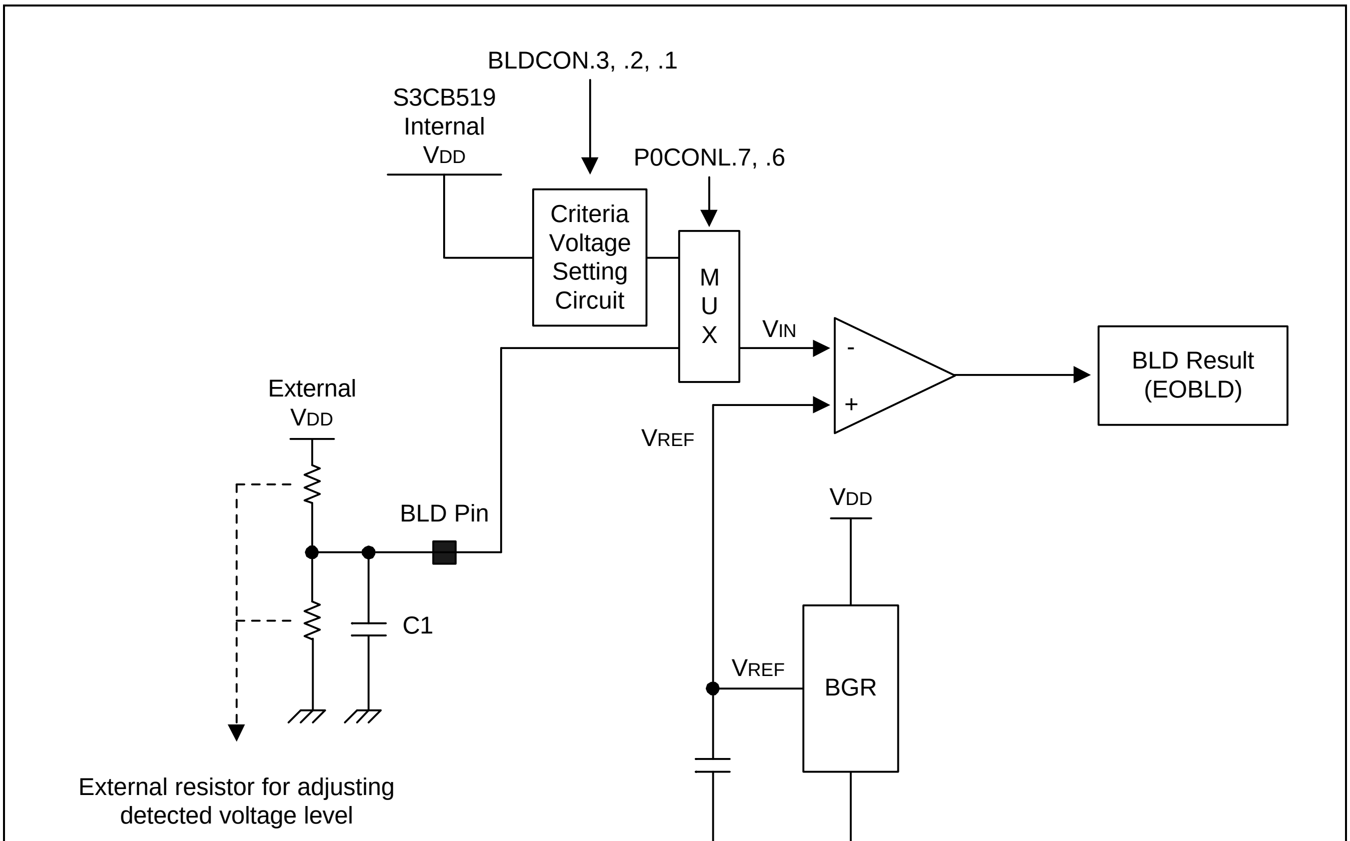
Figure 16-5. Serial I/O Timing in Transmit/Receive Mode (Tx at rising, SIOCON.4 = 1)

17 BATTERY LEVEL DETECTOR

OVERVIEW

The S3CB519/FB519 microcontroller has a built-in BLD (Battery Level Detector) circuit which allows detection of power voltage drop of an external input level or internal $V_{DD.}$
When external input is selected by P0CONL, detection voltage level can be adjusted through the external divided resistors ratio on BLD pin. Internal reference voltage is 1.2 V.

After detection, BLD is automatically disabled, and EOBLD bit is set.
Because the clock for BLD comes from the watch timer, watch timer must be enabled to use BLD.



NOTES:
1. Internal reference voltage (VREF) = 1.2 V
2. C1 = 0.1 µF

Figure 17-1. Voltage Level Detection Circuit

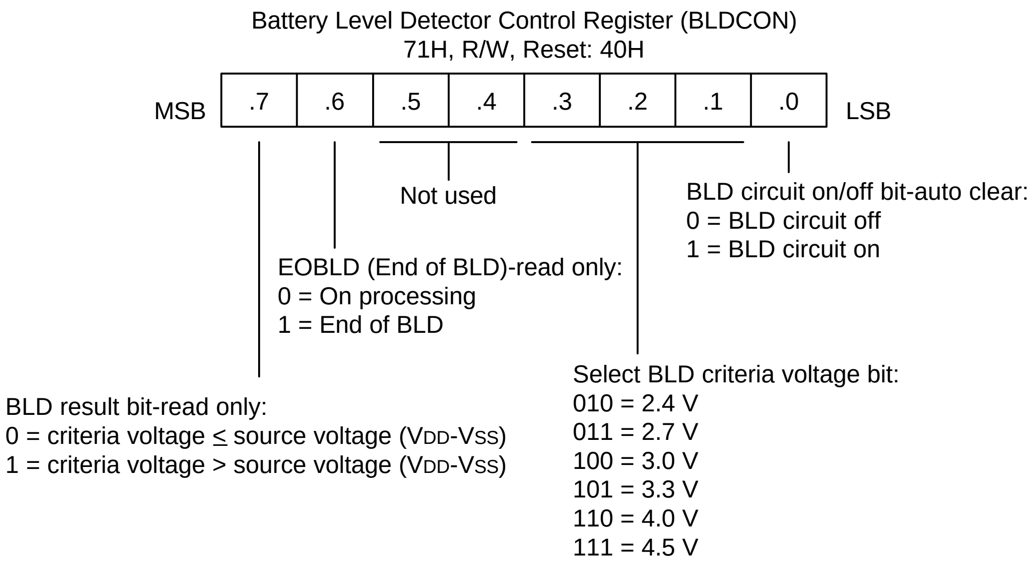

Battery Level Detector Control Register (BLDCON)
71H, R/W, Reset: 40H
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Not used
BLD circuit on/off bit-auto clear:
0 = BLD circuit off
1 = BLD circuit on
EOBLD (End of BLD)-read only:
0 = On processing
1 = End of BLD
Select BLD criteria voltage bit:
010 = 2.4 V
011 = 2.7 V
100 = 3.0 V
101 = 3.3 V
110 = 4.0 V
111 = 4.5 V
BLD result bit-read only:
0 = criteria voltage ≤ source voltage (VDD-VSS)
1 = criteria voltage > source voltage (VDD-VSS)

Figure 17-2. Battery Level Detector Control Register (BLDCON)

18 LCD CONTROLLER/DRIVER

OVERVIEW

This microcontroller can directly drive the (56 segments × 16 commons) LCD panel. Data written to the LCD display RAM can be transferred to the segment signal pins automatically without program control.
When a subsystem clock is selected as the LCD clock source, the LCD display is enabled even during Idle modes.

LCD RAM ADDRESS AREA

LCD RAM can be addressed by 8-bit RAM access instructions. When the bit value of a display segment is "1", the LCD display is turned on; when the bit value is "0", the display is turned off.
Display RAM data are sent out through segment pins SEG0–SEG55 using a direct memory access (DMA) method that is synchronized with the f_{LCD} signal. RAM addresses in this location that are not used for the LCD display can be allocated to general-purpose use.

LCD RAM (RAM BANK 12)

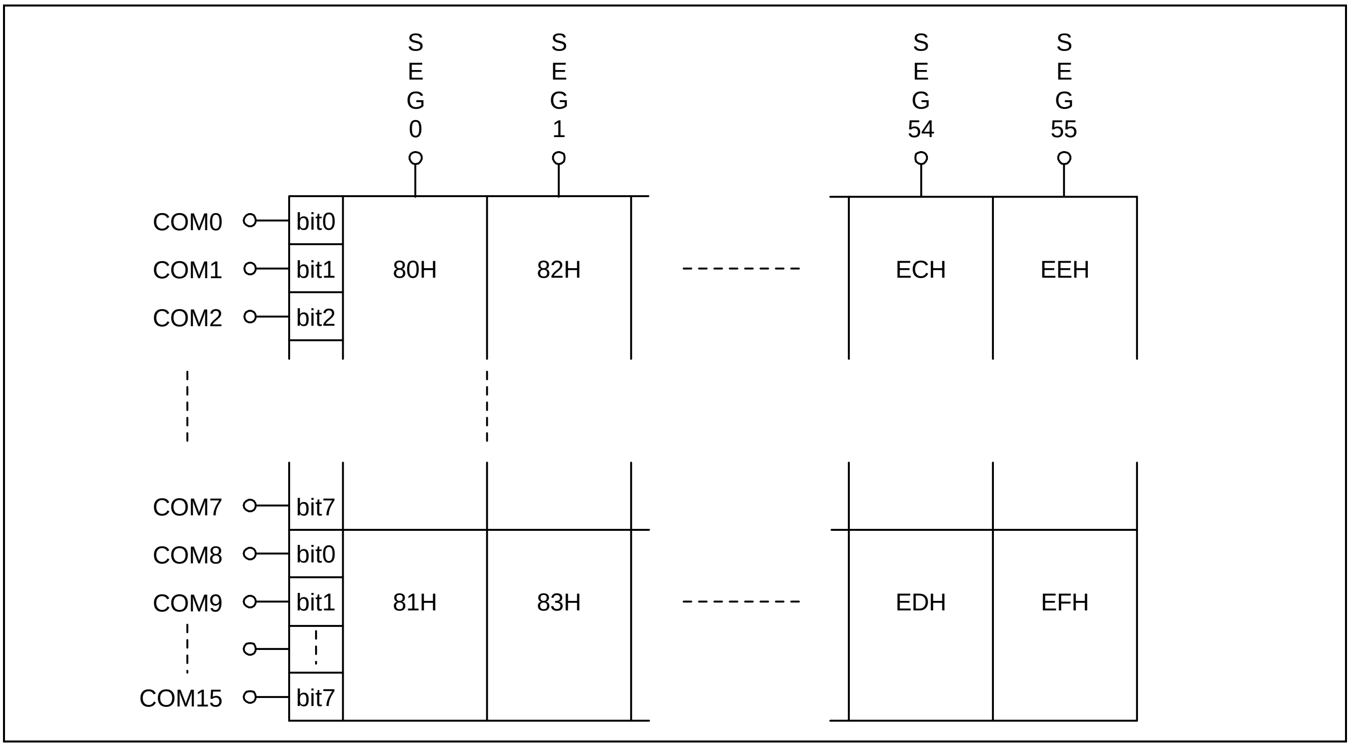
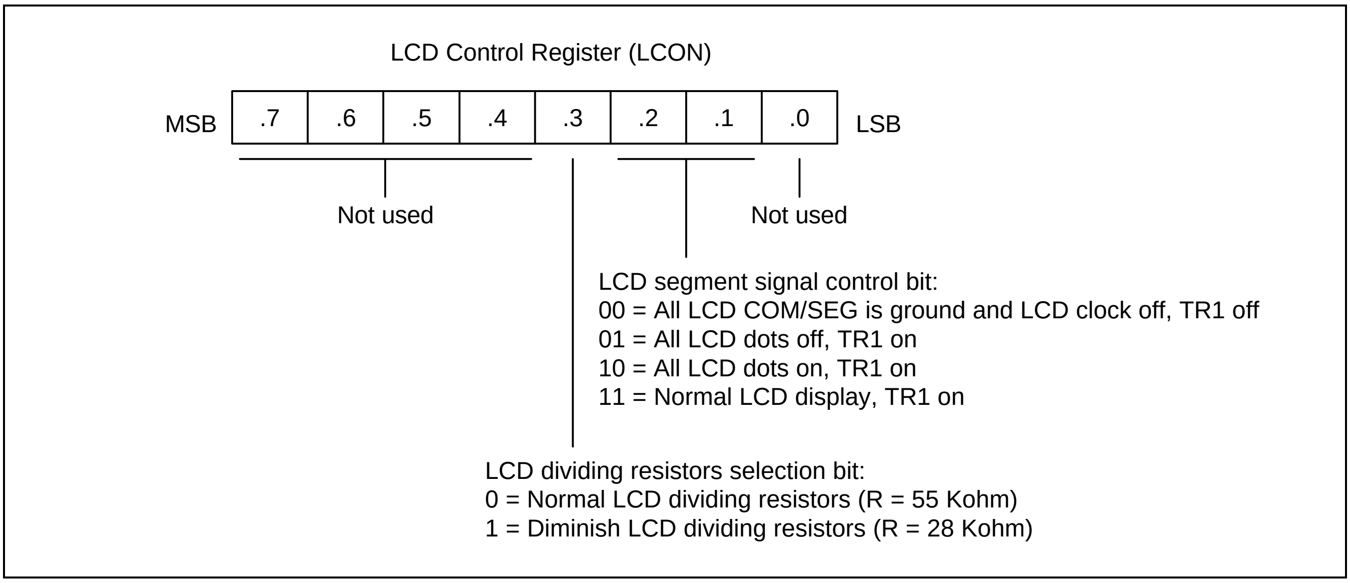


Figure 18-1. LCD Display Data RAM Organization

LCD CONTROL REGISTER (LCON)

LCON controls LCD dividing resistor and LCD display.



LCD segment signal control bit:
00 = All LCD COM/SEG is ground and LCD clock off, TR1 off
01 = All LCD dots off, TR1 on
10 = All LCD dots on, TR1 on
11 = Normal LCD display, TR1 on

LCD dividing resistors selection bit:
0 = Normal LCD dividing resistors (R = 55 Kohm)
1 = Diminish LCD dividing resistors (R = 28 Kohm)

Figure 18-2. LCD Control Register (LCON)

LCD VOLTAGE DIVIDING RESISTORS

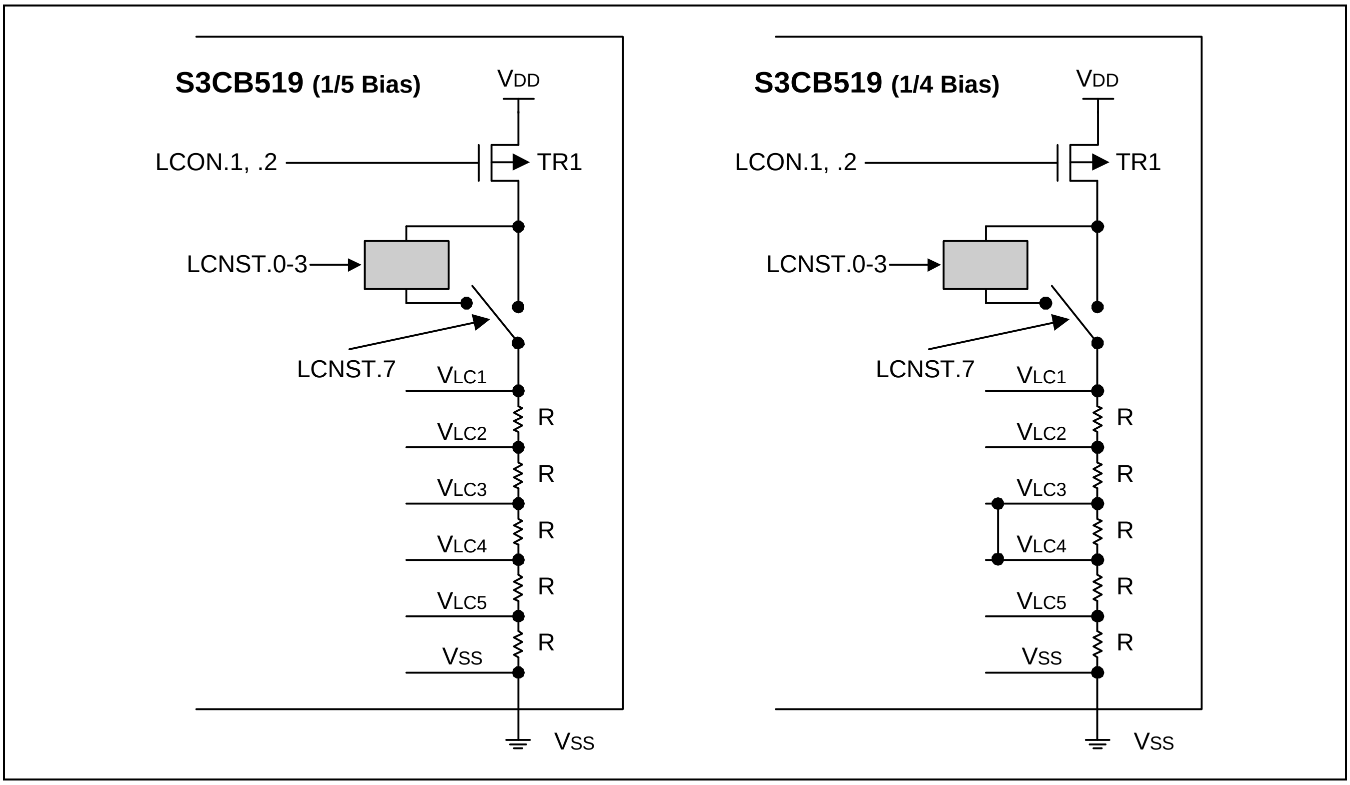
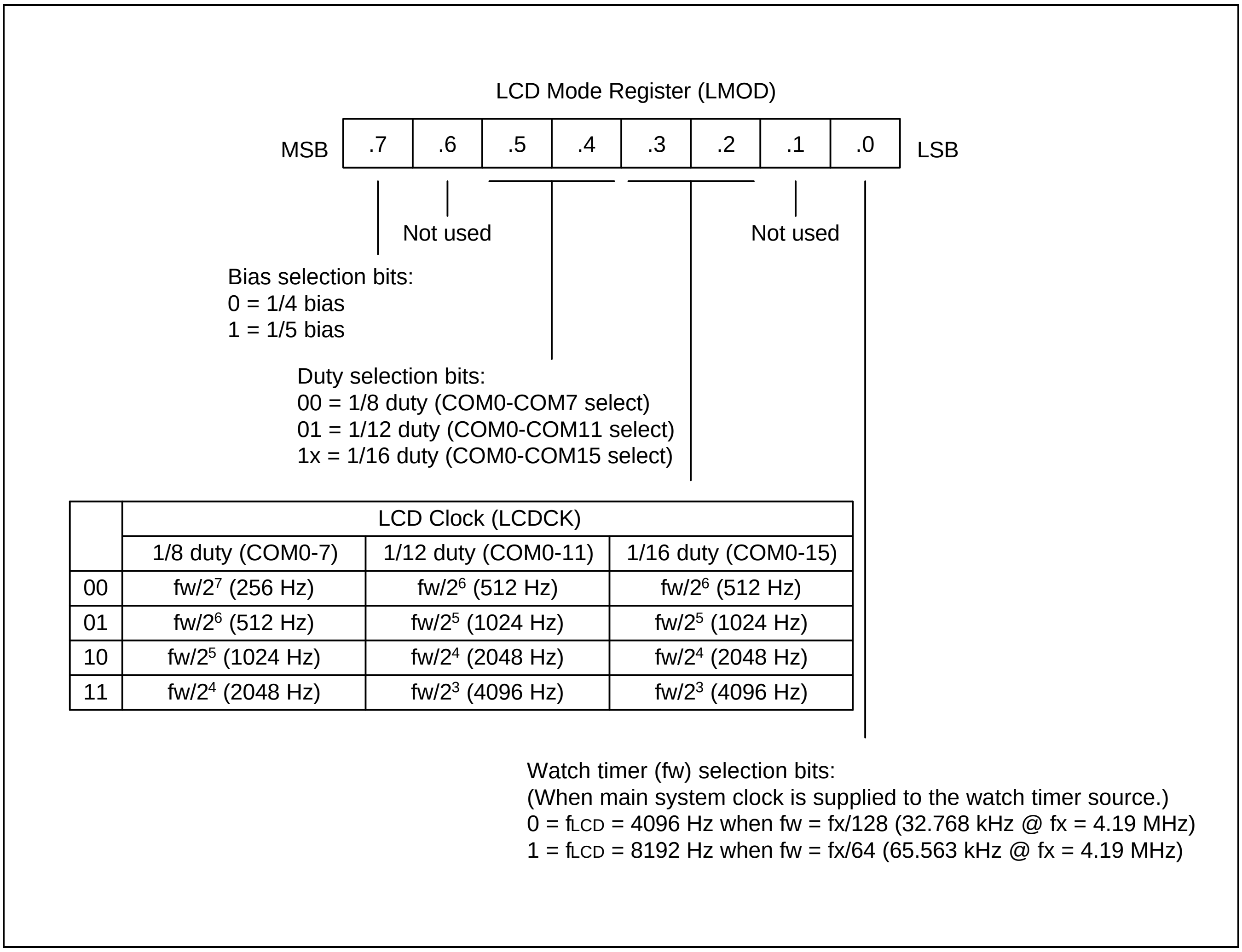


Figure 18-3. Internal Voltage Dividing Resistor Connection

LCD MODE REGISTER (LMOD)

LMOD controls LCD bias, duty, and clock.



Bias selection bits:
0 = 1/4 bias
1 = 1/5 bias

Duty selection bits:
00 = 1/8 duty (COM0-COM7 select)
01 = 1/12 duty (COM0-COM11 select)
1x = 1/16 duty (COM0-COM15 select)

	LCD Clock (LCDCK)		
	1/8 duty (COM0-7)	1/12 duty (COM0-11)	1/16 duty (COM0-15)
00	fw/2^7 (256 Hz)	fw/2^6 (512 Hz)	fw/2^6 (512 Hz)
01	fw/2^6 (512 Hz)	fw/2^5 (1024 Hz)	fw/2^5 (1024 Hz)
10	fw/2^5 (1024 Hz)	fw/2^4 (2048 Hz)	fw/2^4 (2048 Hz)
11	fw/2^4 (2048 Hz)	fw/2^3 (4096 Hz)	fw/2^3 (4096 Hz)

Watch timer (fw) selection bits:
(When main system clock is supplied to the watch timer source.)
0 = f_{LCD} = 4096 Hz when fw = fx/128 (32.768 kHz @ fx = 4.19 MHz)
1 = f_{LCD} = 8192 Hz when fw = fx/64 (65.563 kHz @ fx = 4.19 MHz)

Figure 18-4. LCD Mode Register (LMOD)

LCD CONTRAST CONTROL REGISTER (LCNST)

LCNST controls LCD contrast.

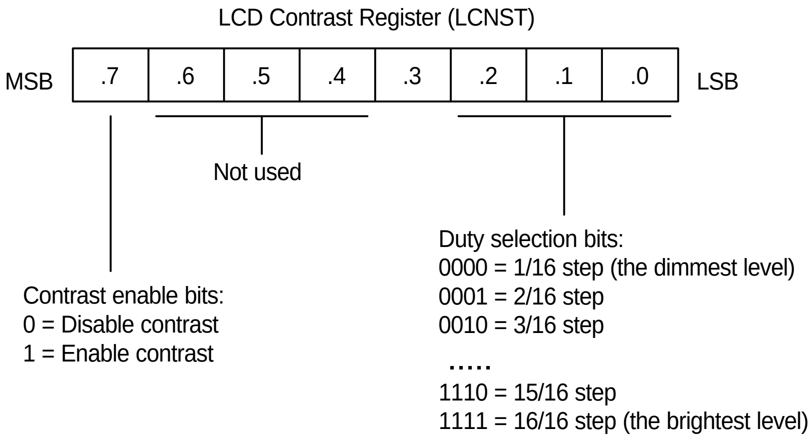


Figure 18-5. LCD Contrast Register (LCNST)

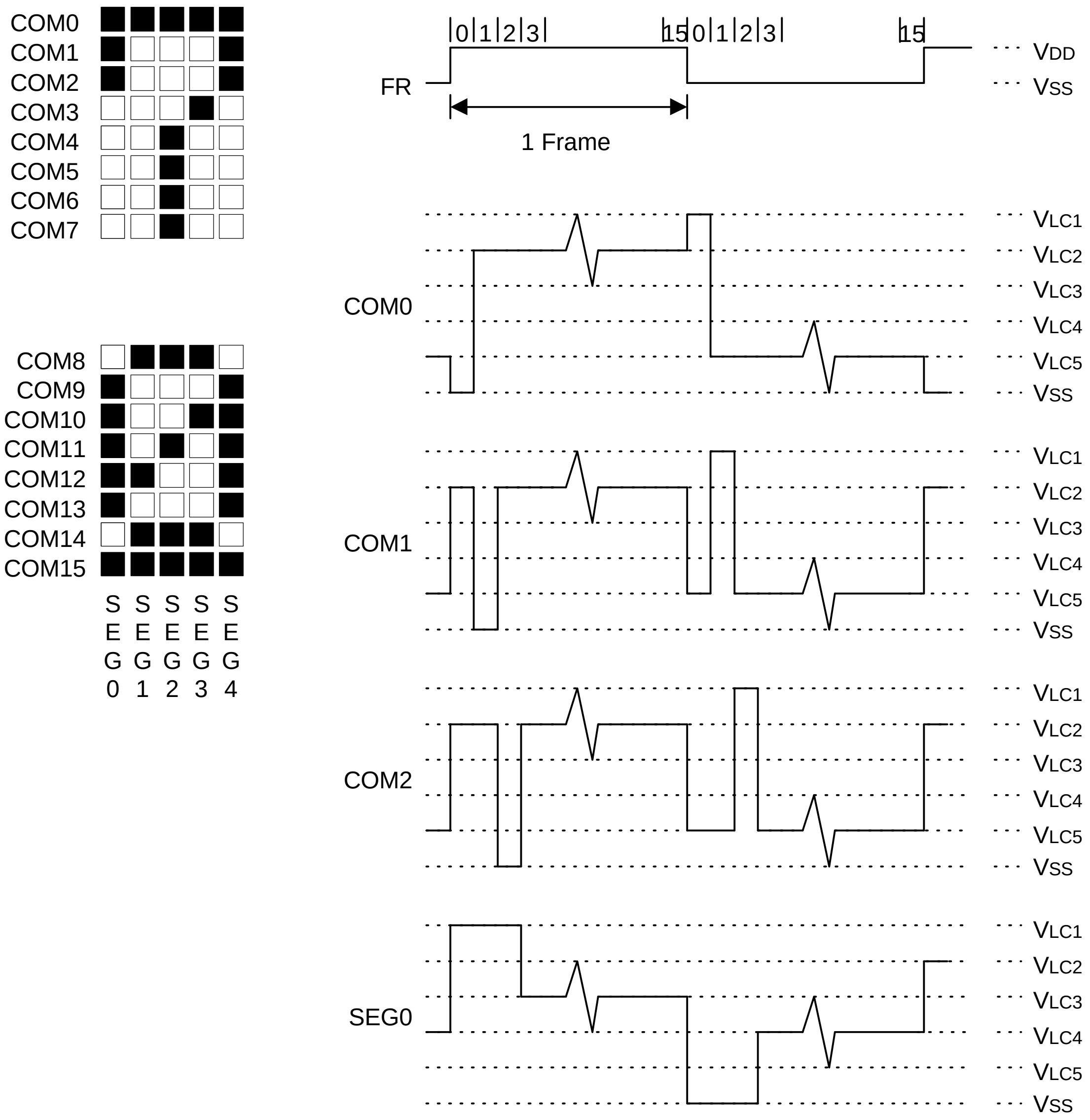


Figure 18-6. LCD Signal Waveforms (1/16 Duty, 1/5 Bias)

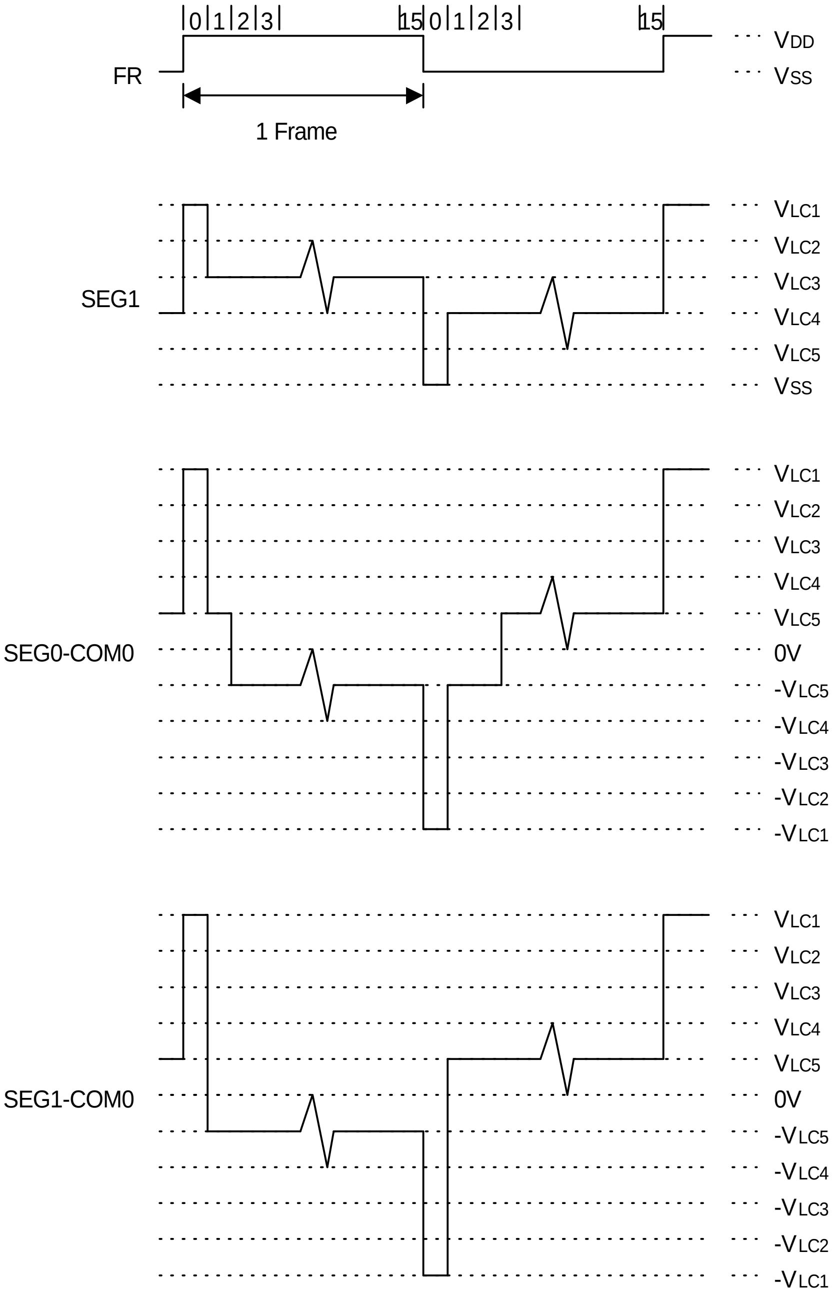


Figure 18-6. LCD Signal Waveforms (1/16 Duty, 1/5 Bias) (Continued)

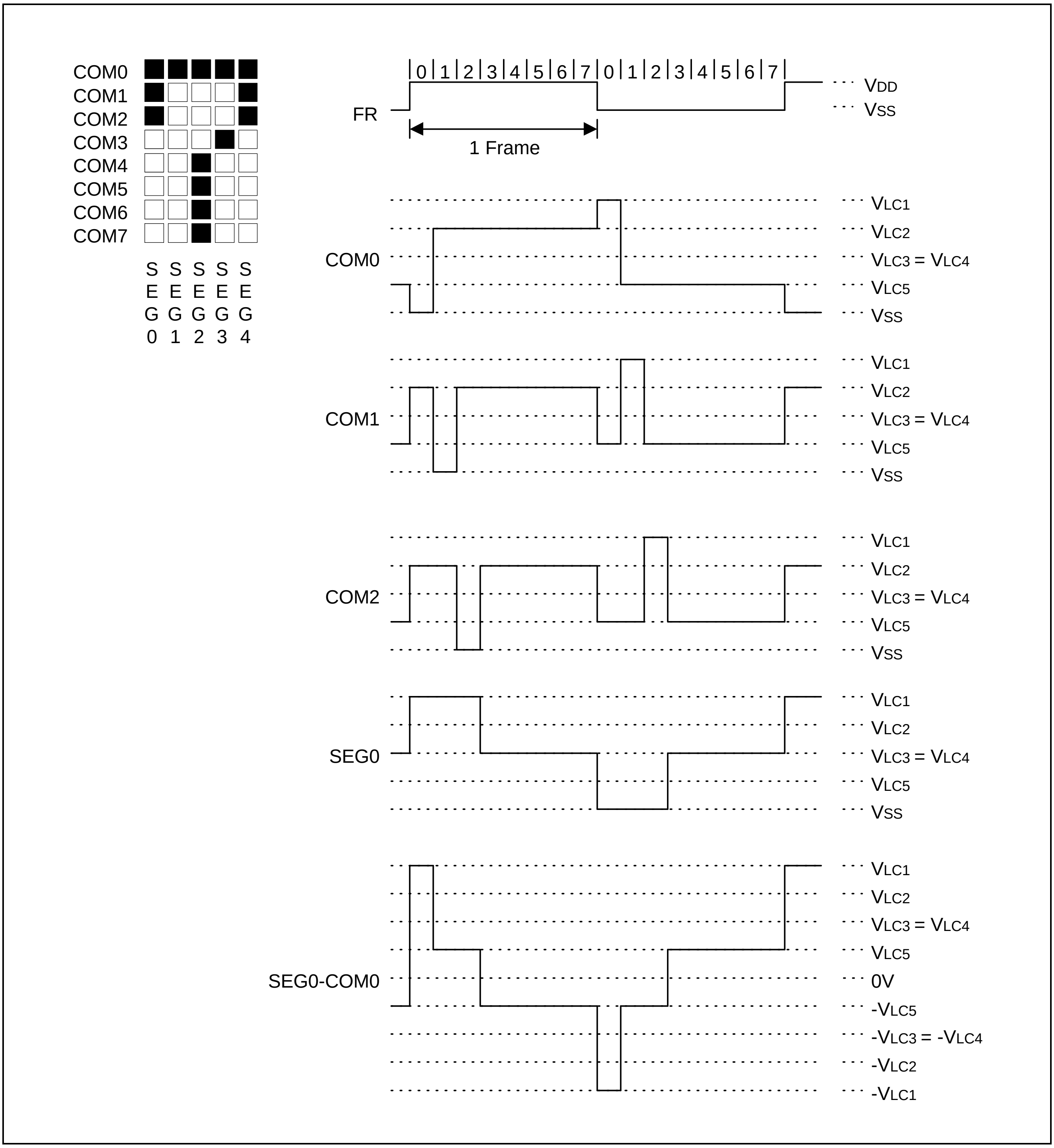


Figure 18-7. LCD Signal Waveforms (1/8 Duty, 1/4 Bias)

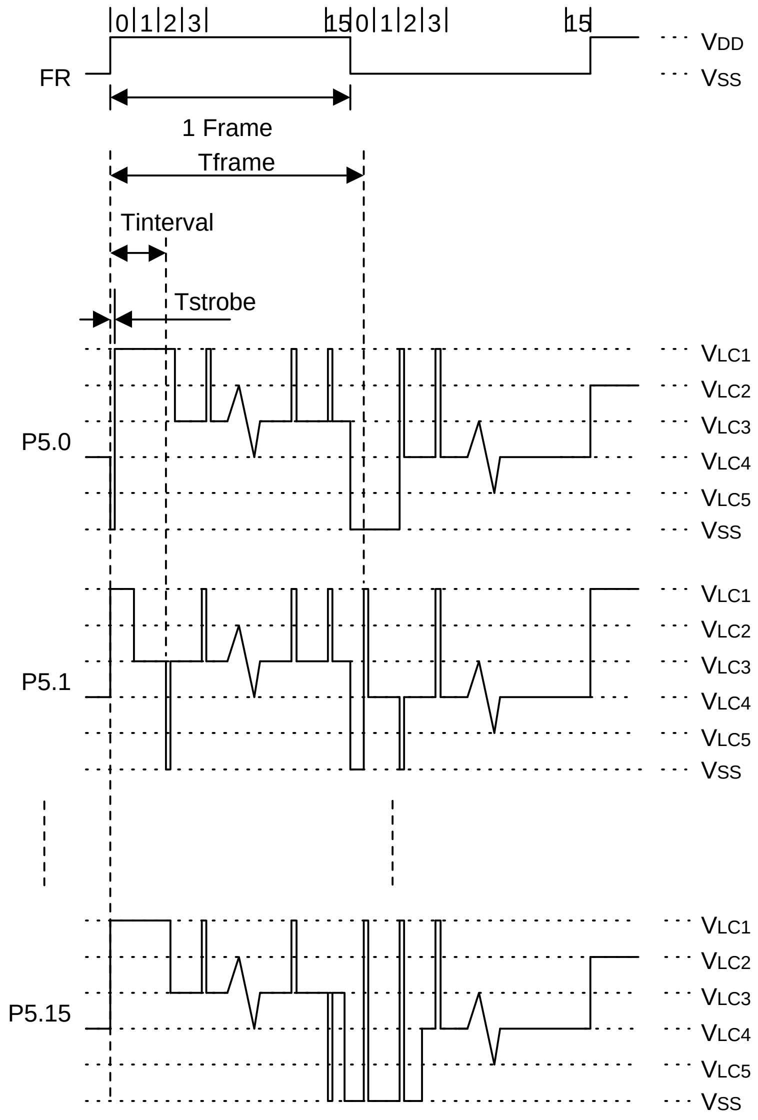
LCD KEY SCAN

When P5CON.7 is set, strobe signal is output to P5.0/SEG39-P5.15/SEG24 during normal SEG output, and the strobe signal number is selected by P5CON setting.
Key input is acquired from KS0/P1.0–KS3/P1.3 and is set in P1CON.
If any pin of P5.0–P5.15 is set only to SEG in P5CON, the selected pin does not output the key strobe signal.
If any of P1.0-P1.3 is set to anything other than the alternative mode (key scan mode), the selected pin acts as a normal I/O.
When P5.0/SEG39–P5.15/SEG24 are set as key strobe, selected key strobe is output, pin by pin, continuously with a selected interval and duration.

When KS0/P1.0–KS3/P1.3 are set as an alternative mode (key scan input), KS0–KS3 state is normally high-impedance, and when SEGx strobe is out, KS0/P1.0–KS3/P1.3 setting is changed to input pull-up state.
The data (Port 1) is input right before the strobe disappears, and if any “Low” state appears, an interrupt occurs.

When the key scan interrupt occurs, user can read the Interrupt request register for the key input state and the Port 5 data register for the key strobe state. The data of the selected pin, P5, is not changed until next strobe occurs. Port 1 data register has invalid data when in the key input state.

Port 5 data register value is ‘0’ for P5.0 strobe, ‘1’ for P5.1, ‘2’ for P5.2, …’0FH’ for P5.15 strobe.



NOTES:

1. Tframe
   When P5.0-P5.3 is used, Tframe is Tinterval x 4,
   When P5.0-P5.7 is used, Tframe is Tinterval x 8,
   When P5.0-P5.11 is used, Tframe is Tinterval x 12,
   When P5.0-P5.15 is used, Tframe is Tinterval x 16.
2. Tinterval, Tstrobe value is set by setting P5CON value.

Figure 18-8. LCD Waveform when Key Strobe Signal is Active

NOTES

19 A/D CONVERTER

OVERVIEW

The ADC is Sigma-Delta type ADC for speech and telephony applications. The ADC contains both digital IIR/FIR filters, and an on-chip voltage reference circuit is included to allow supply operations.

FEATURES

Sigma Delta ADC.

- 256X oversampling
- On chip decimation filter
- On chip voltage reference circuitry

A/D CONVERTER CONTROL REGISTER (ADCON)

User can select the A/D input clock for dividing higher crystal by controlling ADCON.
A/D converted data are 14-bit resolution and are input to ADATAH (High byte), ADATAL (Low byte) in 16-bit data format. Because A/DC use 256X over-sampling, for 8 kHz sampling, when crystal is 2.048 MHz (= 8 kHz × 256), user must select fx as AD/DA input clock.
And when crystal is 4.096 MHz (= 2 × 8 kHz × 256), user must select fx/2 as AD/DA input clock.

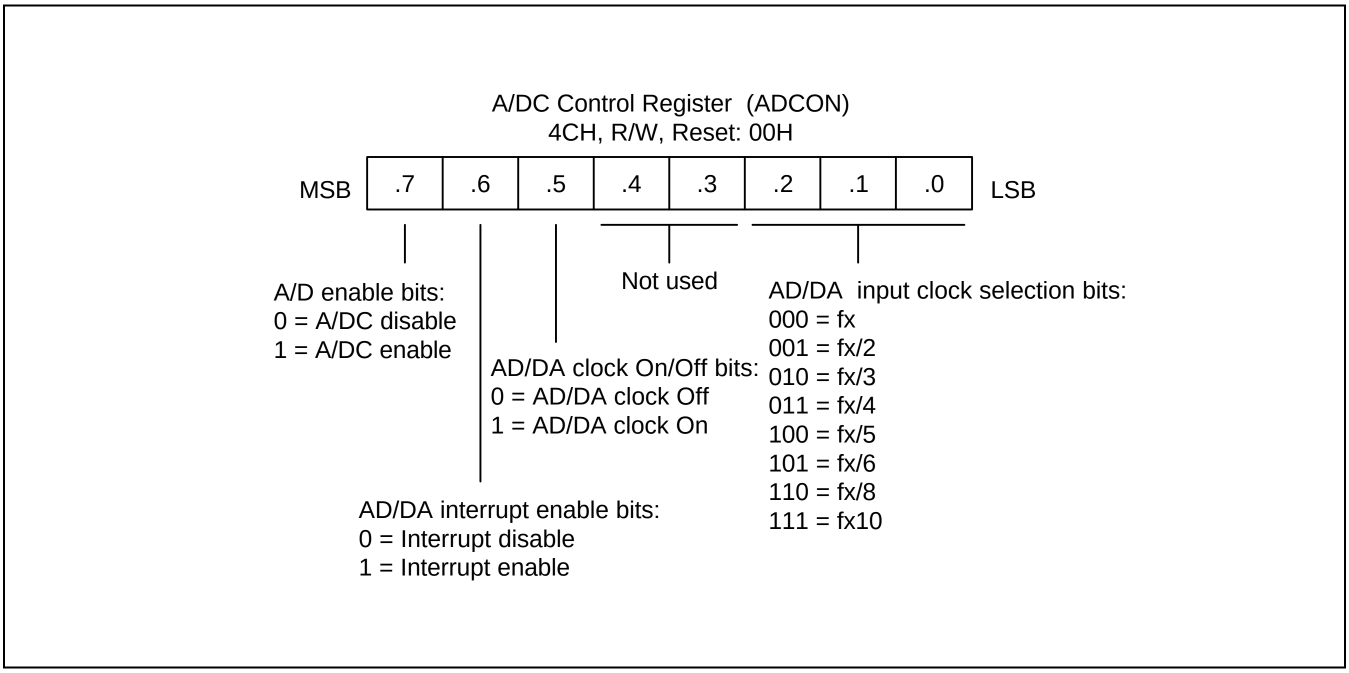
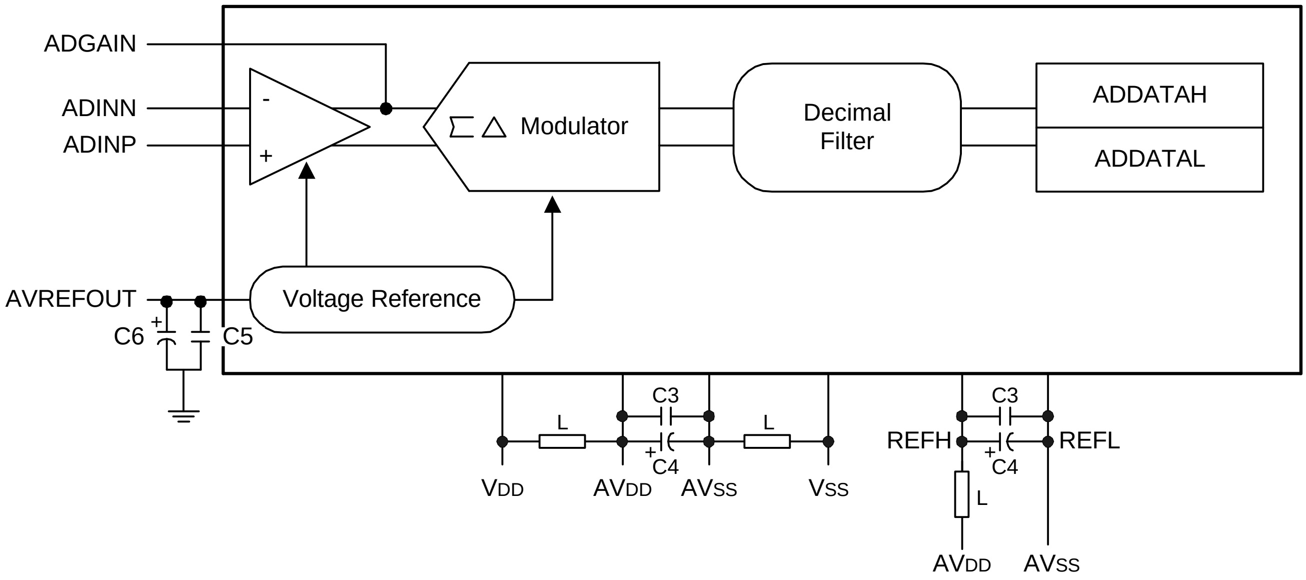


Figure 19-1. A/DC Control Register (ADCON)

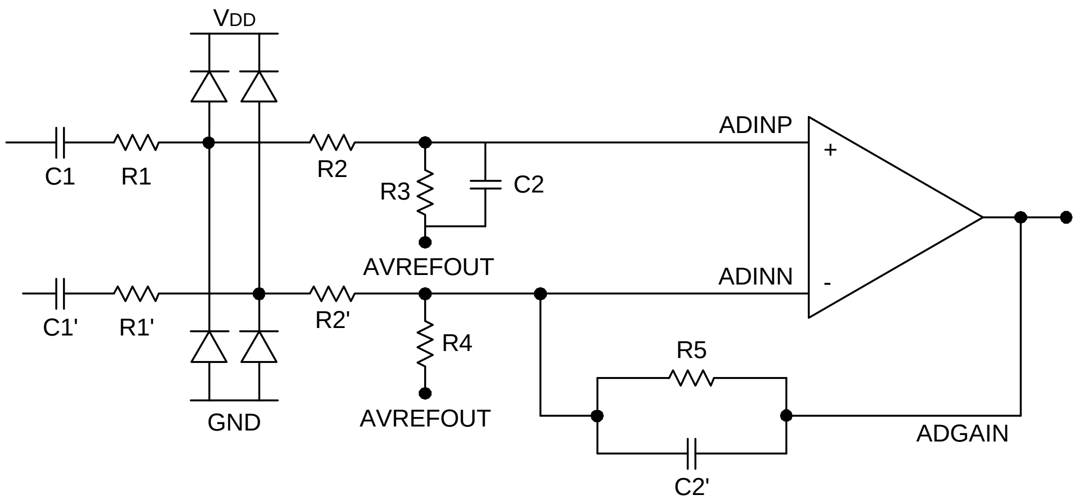


NOTE:

C3:	0.1µF	TANTALUM	CAPACITOR
C4, C6:	10 µF	CERAMIC	CAPACITOR
C5:	1 µF	TANTALUM	CAPACITOR
L:	0.1mH		

Figure 19-2. A/D Converter Block Diagram

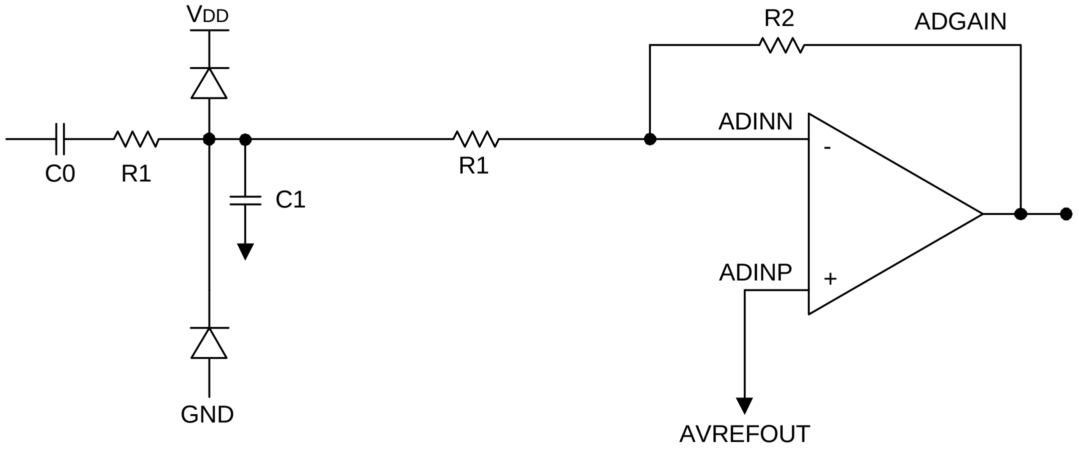
Differential-ended input application



Voltage Gain: R5/(R1 + R2)
R3 = R4 x R5/(R4 +R5)
R1 = R1'
R2 = R2'
C1 = C1'
C2 = C2'

Example:
R1 = R1' = 390 kΩ
R2 = R2' = 47 kΩ
R3 = 110 kΩ
R4 = 220 kΩ
R5 = 220 kΩ
C1 = C1' = 470 pF
C2 = C2' = 22 pF

Single-ended input application



Voltage Gain: $\frac{R2}{2 \times R1}$
R2 = 2 x R1
R1>50 kΩ
C1 = 1 x 10^{-5}/R2

Example:
R1 = 100 kΩ
R2 = 200 kΩ
C1 = 50 pF

Figure 19-3. Application Example

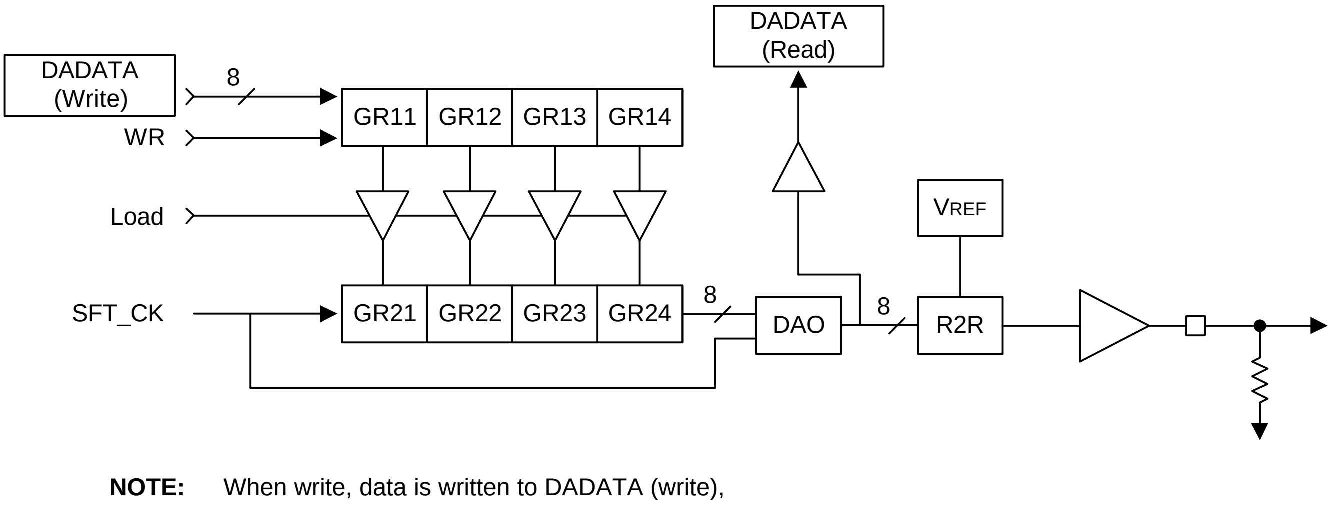
20 D/A CONVERTER

OVERVIEW

This MCU has an 8-bit Digital-to-Analog converter with R-2R structure. This DAC (Digital **–** to Analog converter) is used to generate analog voltage, $V_{DA,}$ with 256-steps (2^8). This function is controlled by the DAC mode register (DACON).
To enable the converter, the DACON.0 must be set to “1”. To generate an analog voltage (V_{DA}), load the appropriate value to DADATA. The level of the analog voltage is determined by DADATA.

When a user writes data to DATATA, the contents of GR13 is shifted to GR14, GR12 to GR13, GR11 to GR12, and DATATA to GR11.
The content of GR24 is output to DAO. When GR24 is output and some time passes, the contents of GR23 is shifted to GR24, GR22 to GR23, GR21 to GR22. After all the contents of GR21–GR24 is out to DAO, GR11–GR14 will be copied to GR21–GR24.
Four consequent DA data will be written to DADATA every AD/DA interrupt. Then the four data will be out with same interval until the next AD/DA interrupt occurs.
The interval between DAO and the next DAO is about 31 μsec when a 4.096 MHz oscillator is used, and ADC clock is fx/2.

The interval clock comes from the ADC. (See ADCON).



NOTE: When write, data is written to DADATA (write),
When read, data is read from DADATA (read).

Figure 20-1. D/A Converter Circuit Diagram

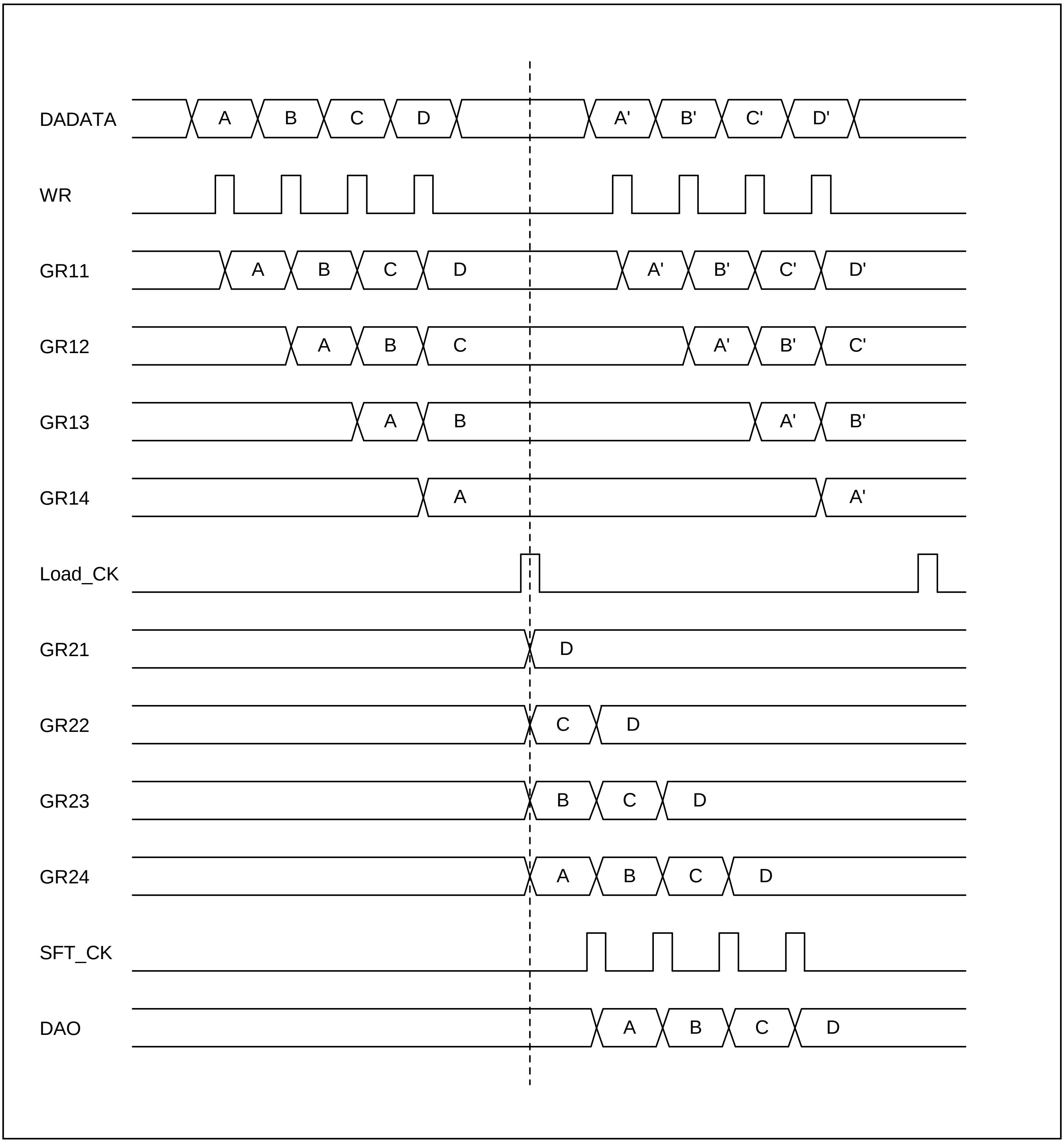


Figure 20-2. D/A Converter Timing Diagram

D/A CONVERTER DATA REGISTER (DADATA)

The DADATA specifies the digital data to generate analog voltage.
RESET initializes the DADATA value to "00H". The D/A converter output value, V_{DAO}, is calculated by following formula.

$V_{DAO}= V_{PP} \times (n / 256) + V_{BIAS}-\frac{1}{2} V_{PP}$ (n = 0-255, DADATA value), where, V_{PP} and V_{BIAS} is specified in electrical data and the V_{PP} is a regulated output voltage.

If DADATA value is 0, $V_{DAO} = V_{BIAS} - \frac{1}{2} V_{PP}$
If DADATA value is 128, $V_{DAO} = V_{BIAS}$
If DADATA value is 255, $V_{DAO} = V_{BIAS} + \frac{1}{2} V_{PP}$

D/A CONVERTER CONTROL REGISTER (DACON)

DACON values are set to logic "00H" following RESET, and this value disables DAC.

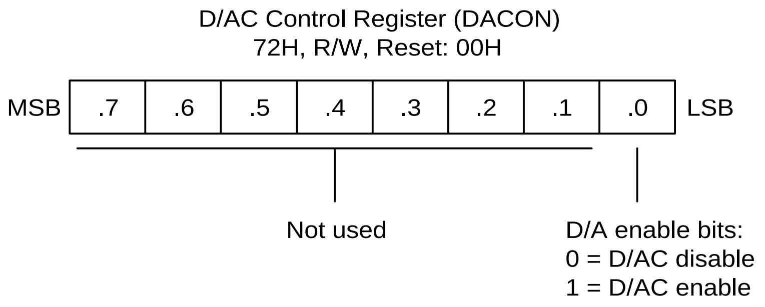


Figure 20-3. D/A Control Register (DACON)

NOTES

21 MAC816

MAC816 ARCHITECTURE OVERVIEW

MAC816 is a 16-bit fixed-point DSP coprocessor for low-end DSP applications. It is designed as one of the DSP coprocessor engines for CalmRISC, which targets towards cost-sensitive low-end multimedia DSP applications. The generic coprocessor instructions for CalmRISC are renamed according to the intended operations on MAC816, including the DSP data type, and the DSP addressing mode. Below represented is the top block diagram of MAC816.

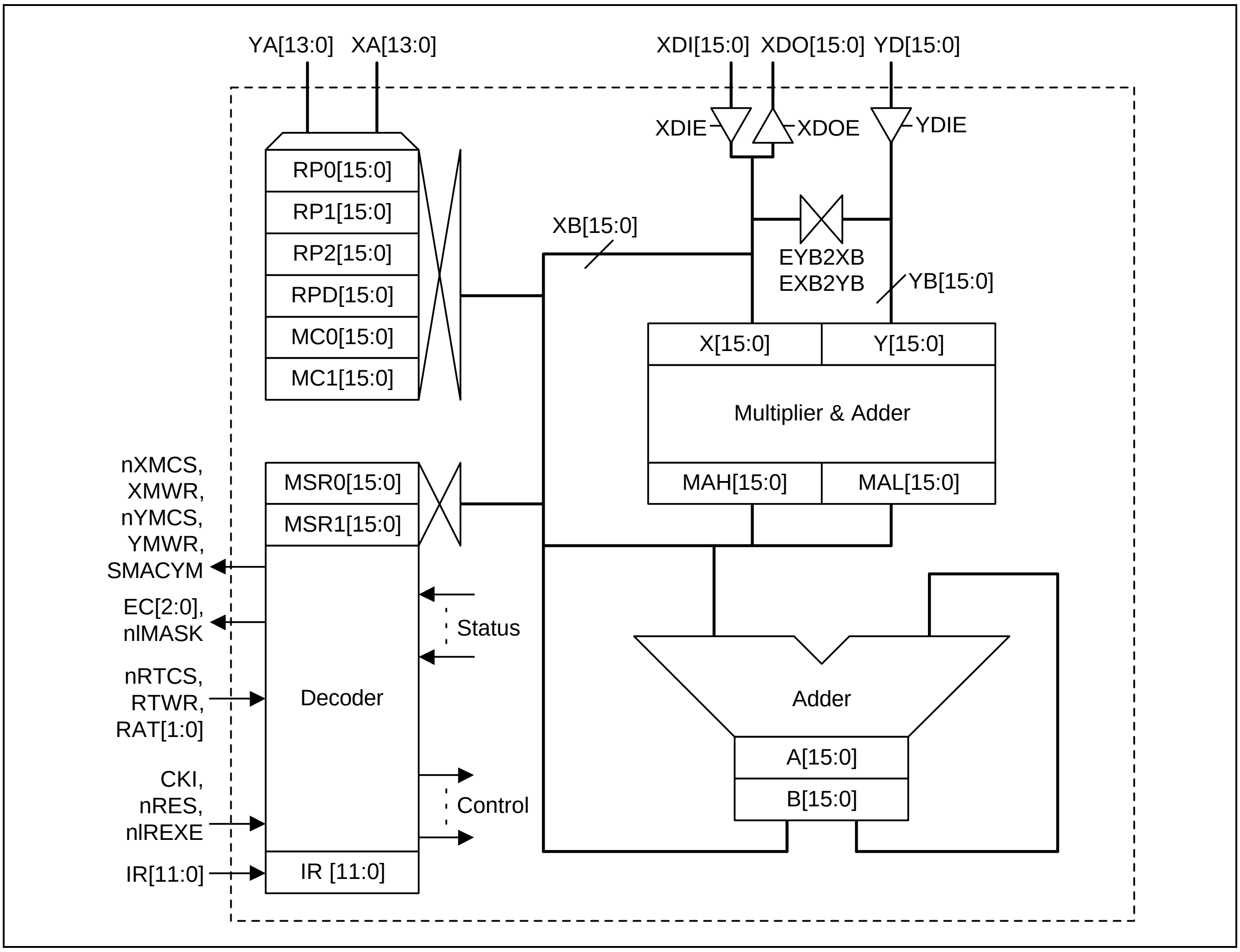


Figure 21-1. Top Block Diagram

The MAC816 building blocks consist of:

- Multiplier and Accumulator Unit (MAU)
- Arithmetic Unit (AU)
- RAM Pointer Unit (RPU)
- Interface Unit (IU)

Basically, MAU (Multiplier and Accumulator Unit) is built around an 8-bit by 16-bit parallel multiplier and a 32-bit adder for multiply-and-accumulate (MAC) operations. Hence, 16-bit by 16-bit MAC operations are performed in two cycles in MAC816. AU performs 16-bit arithmetic and shift operations for DSP. RPU of MAC816 consists of 3 data memory pointers and 2 control blocks for the pointer modulo calculation. The pointers are used for accessing the data memory for a 16-bit data operand. Since two 16-bit data operands can be fetched simultaneously in a single cycle through XD[15:0] and YD[15:0] for MAC operation, the data memory should be partitioned into two parts: X and Y memory. IU is for the communication between CalmRISC and MAC816. It decodes coprocessor interface signals from CalmRISC and controls the data paths in MAC816, according to the decoding result.

Most of MAC816 instructions are 1-word instruction, while several instructions which need 16-bit immediate value are 2-word instruction.

PROGRAMMER'S MODEL

In this chapter, the important features of MAC816 are discussed in detail. How the data memory is organized is discussed and the explanation of registers follows. Last, the host interface with CalmRISC will be explained.

DATA MEMORY ACCESSES

The total data memory address space for MAC816 is 32K-word. The 32K-word data memory space is physically divided into XM (X area memory) and YM (Y area memory). This memory is actually shared with the host processor (CalmRISC). The host processor accesses the 64K-byte data memory in byte width, otherwise MAC816 accesses it in 2-byte width. MAC816 has two types of addressing modes. RPU can generate two 15-bit addresses every instruction cycle which can be post-modified.

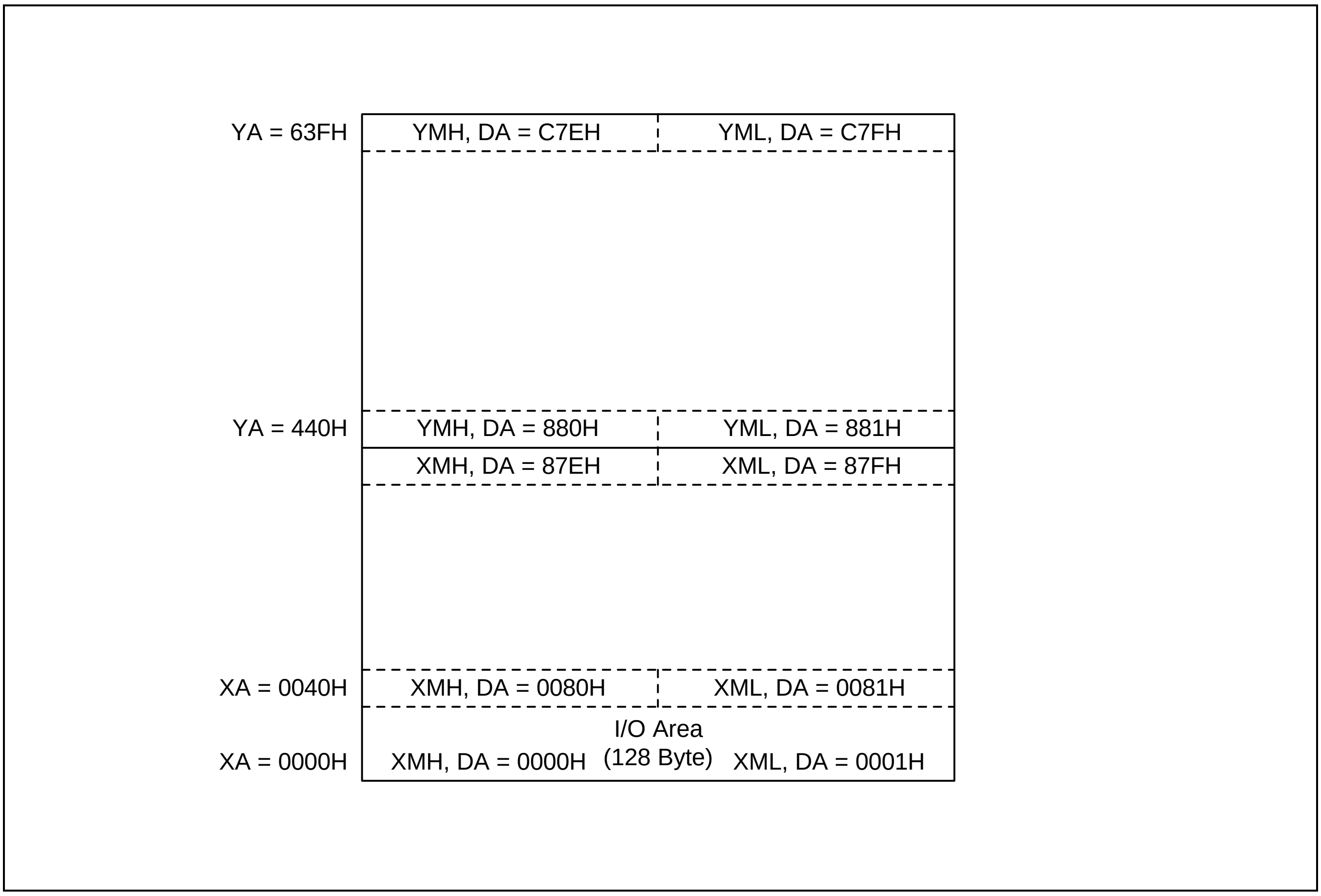


Figure 21-2. Data Memory Organization

Table 21-1. RPU(RAM Pointer Unit) Registers

Registers		Mnemonics	Description	Reset Value
Mreg1	RPi	**RP0**	**R**AM **P**ointer register **0**	Unknown
		RP1	**R**AM **P**ointer register **1**	Unknown
		RP2	**R**AM **P**ointer register **2**	Unknown
		RPD	**R**AM **P**ointer for short direct addressing	Unknown
MCi		**MC0**	**M**odulo **C**ontrol register **0** for RP0/RP1	Unknown
		MC1	**M**odulo **C**ontrol register **1** for RP2	Unknown

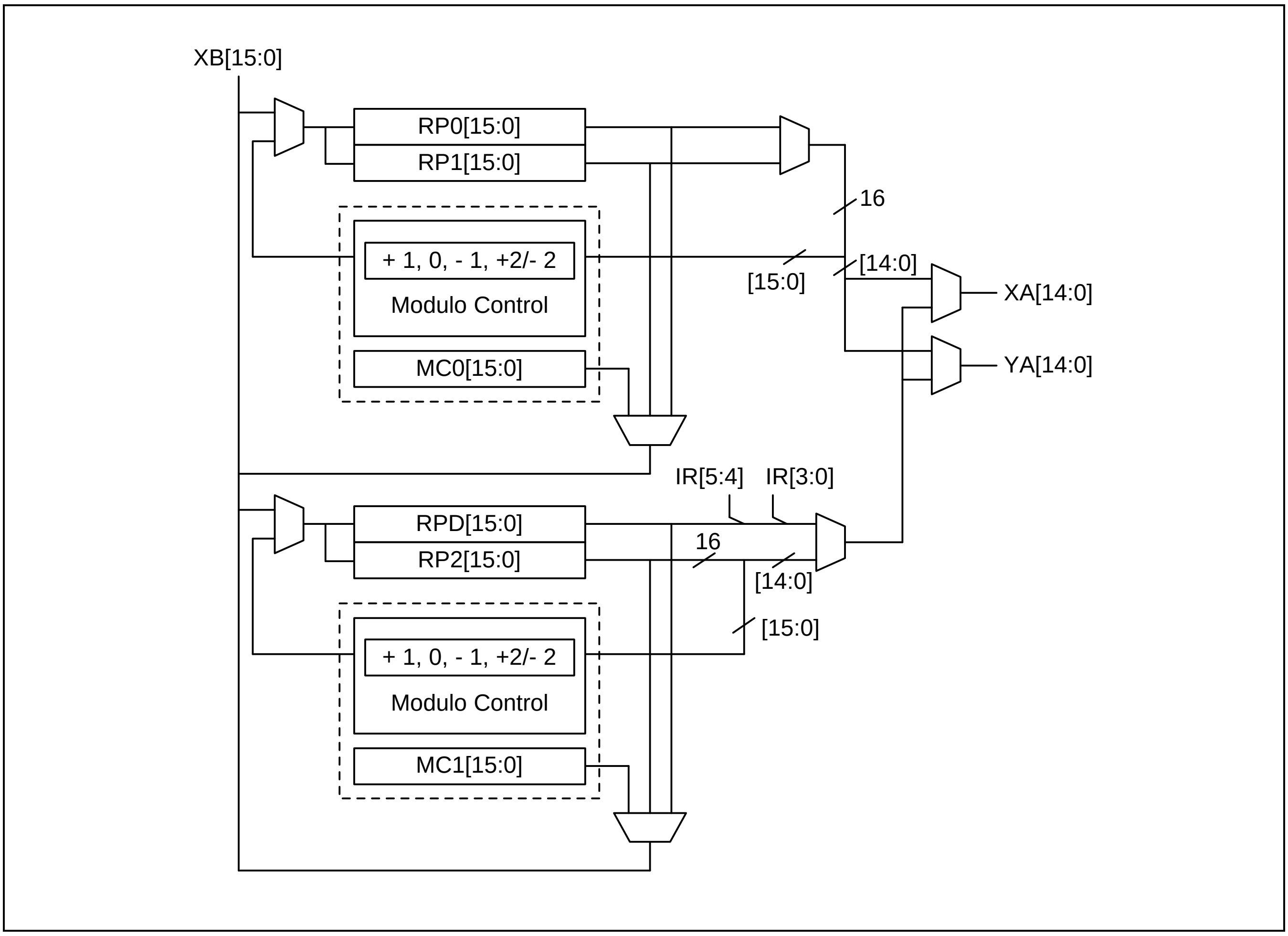


Figure 21-3. RPU (RAM Pointer Unit) Block Diagram

Short Direct Memory Addressing Mode

Six-bits embedded in the instruction code as LSBs and 9-bits from the RPD[14:6] of RPD register as MSB compose the 15-bit address to the data memory address. This can be used with some instructions operating an Ai (A/B register in AU) operand. In “load/store *mreg1*” instruction, a 4-bit embedded in the instruction code as LSBs and 11-bits from the RPD[14:4] of RPD register as MSB compose the 15-bit address to the data memory address. This can be used to load/store RAM pointer register from/to data memory.

Indirect Memory Addressing Mode

The RPi registers of RPU are used as a 15-bit address for indirect addressing XM (X area memory) or YM (Y area memory). Some instructions can simultaneously access the XM and YM, and then RP0 is used for XM and RP2 for YM. In indirect addressing mode, RPi register is modified by +1,-1,-2, and +2 after the addressing. The MSB of RPi register enables modulo opera tion of the RPi modification. The RPU registers are divided into two groups of simultaneous addressing over XA and YA: X-memory is addressed by RP0 and RP1 with MC0 and Y-memory is addressed by RP2 with MC1. RPi from both groups can be used for both XA and YA for instruction, which uses only one address register. In this instruction the XM and YM can be viewed as a single continuous data memory space.

Table 21-2. RPi register bit information

Bit position	Value	Description
[14:0]	0H–7FFFH	Data memory(XM/YM) address
[15]	0	Modulo mode disable
	1	Modulo mode enable

Modulo Control Registers (MCi)

MCi controls RP0, RP1 and RP2 register modifications after indirect memory accessing. MCi has an upper boundary value in MCi[9:0], a step size in MCi[12:10] and a modulo size information in MCi[15:13]. The upper boundary determines the upper limit of the modulo body. The modulo size information determines the lower limit and size of the modulo body as shown below. For example, assume RP0 = 87FFH and MC0 = 03FFH: If "@RP0+" is used on the operand of the instruction, the data memory contents pointed by "07FFH" is accessed, and RP0 is updated to "8400H" after memory accessing. Assume RP0 = 07FFH and MC0 = 03FFH: If "@RP0+" is used on the operand of the instruction, the data memory contents pointed by "07FFH" is accessed, and RP0 is updated to "0800H" after memory accessing.

Bit position	Value	Description
[9:0]	0H–3FFH	Upper boundary
[12:10]	000	Step size = + 2
	001	Step size = - 2
	010–111	Reserved
[15:13]	000	Maximum modulo size = 1024 (0H to 3FFH), Modulo body = RPi[14:10]:0000000000 to RPi[14:10]:MCi[9:0]
	001	Maximum modulo size = 8 (0H to 7H), Modulo body = RPi[14:3]:000 to RPi[14:3]:MCi[2:0]
	010	Maximum modulo size = 16 (0H to 0FH), Modulo body = RPi[14:4]:0000 to RPi[14:4]:MCi[3:0]
	011	Maximum modulo size = 32 (0H to 1FH), Modulo body = RPi[14:5]:00000 to RPi[14:5]:MCi[4:0]
	100	Maximum modulo size = 64 (0H to 3FH), Modulo body = RPi[14:6]:000000 to RPi[14:6]:MCi[5:0]
	101	Maximum modulo size = 128 (0H to 7FH), Modulo body = RPi[14:7]:0000000 to RPi[14:7]:MCi[6:0]
	110	Maximum modulo size = 256 (0H to 0FFH), Modulo body = RPi[14:8]:00000000 to RPi[14:8]:MCi[7:0]
	111	Maximum modulo size = 512 (0H to 1FFH), Modulo body = RPi[14:9]:000000000 to RPi[14:9]:MCi[8:0]

COMPUTATION UNIT

The computation unit contains two main units, the Multiplier and Accumulator Unit (MAU) and Arithmetic Unit (AU).

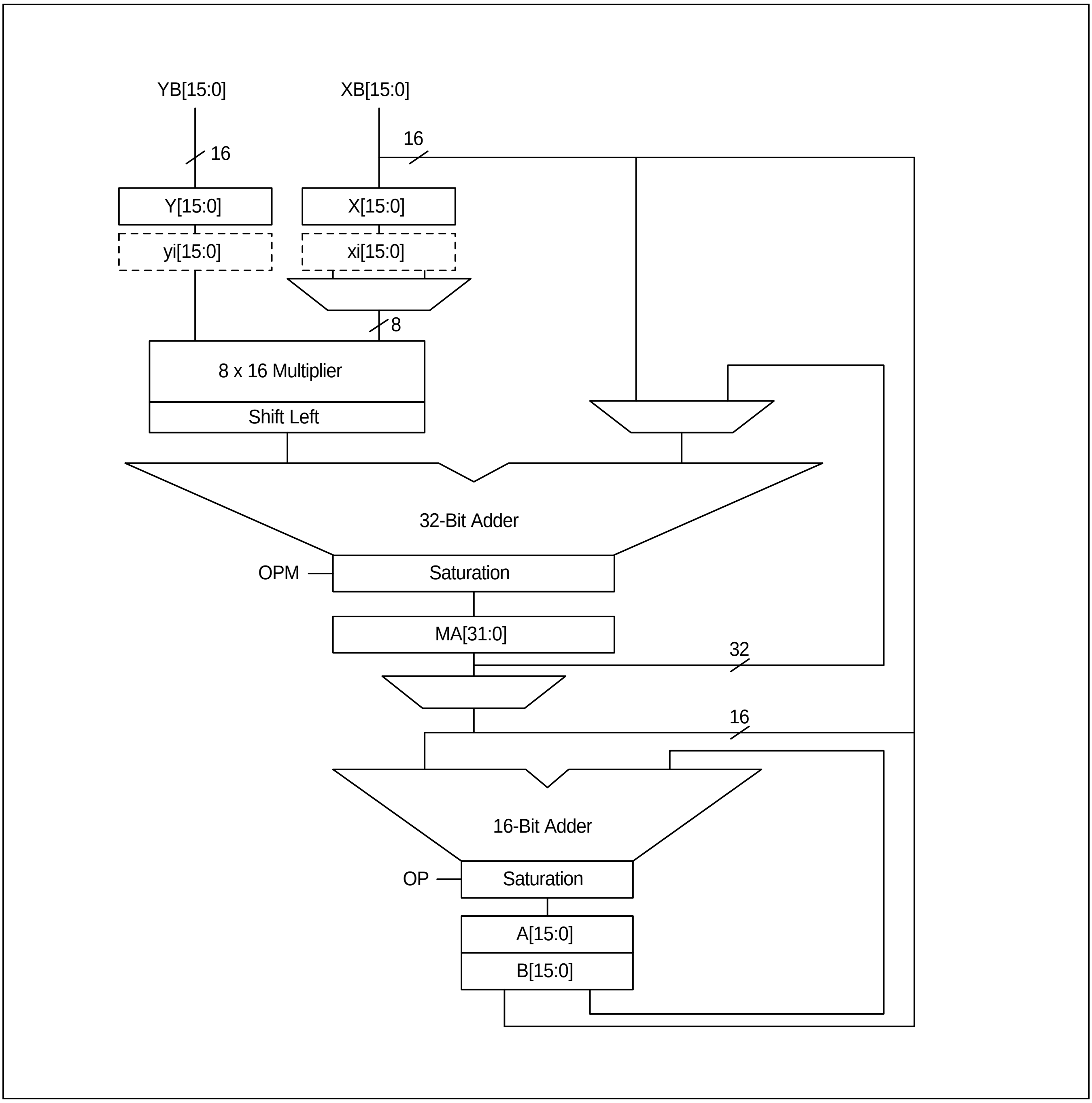


Figure 21-4. Computation Unit Block Diagram

Multiplier and Accumulator Unit (MAU)

The MAU consists of a 8 by 16 to 24 bit parallel multiplier, two 16-bit input registers(X and Y), a product output shifter, and 32-bit product and accumulator register(MA). The multiplier performs signed by signed, signed by unsigned, unsigned by signed, or unsigned by unsigned multiplication. By clearing “MSR1[2] (or M816)”, the MAU can perform 16 by 16 to 32 bit parallel multiplication in 2 cycles. After the multiplier instruction, if a read instruction of MA is followed, previous MA register value will be read out because. During 16 by 16 multiplication, in the second cycle of multiplication, the instruction of MA modification can cause illegal multiplication results. Thus, multiplier instruction should not be followed by MA register writing. The “MV” flag is set if arithmetic overflow occurs after an arithmetic operation in the MA register, and if set “OPM”, the MA register is saturated to a 32-bit positive (7FFFFFFFH) or negative (80000000H). The MA register is not updated by loading X and Y registers. Hence, the X and Y registers can be used as ~~a~~ temporary data registers. The registers in MAU are as shown in the table.

Mnemonics	Description	Reset Value
X	MAU X input register	Unknown
Y	MAU Y input register	Unknown
MAL	MAU Accumulator register, MA[15:0]	Unknown
MAH or MA	MAU Accumulator register, MA[31:16]	Unknown

Arithmetic Unit (AU)

The AU consists of 16-bit adder, 1-bit shifter, and two result registers (A and B). The AU receives one operand from Ai and another operand from XB or Ai. Operations between the two Ai registers are also possible. The source and destination Ai register of an AU instruction are always the same. The XB bus is used for transferring one of the register content, an immediate operand, or the content of a data memory location as a source operand. The AU results are stored in one of the Ai registers. The AU can perform add, subtract, compare, and shift operations. It uses two’s complement arithmetic operations. The AU evaluates the status flags of an arithmetic result. The “V” flag is set if arithmetic overflow occurs after an arithmetic operation in A or B register, and if set to “OPA” or “OPB”, the A or B register is saturated to a 16-bit positive (7FFFH) or negative (8000H). Data transfer between MAC816 and the host processor can be achieved via A or B register. The host processor (CalmRISC) can directly access A and B registers of MAC816 through “CLD GPR,imm” or “CLD imm,GPR” instruction.

STATUS REGISTERS

Status Register 0 : MSR0

MSR0 is mainly reserved for flagging an AU result , for protecting control overflow, and for indicating test results.

Bit Name	Bit	Description
C	0	Carry flag
V	1	Overflow flag
Z	2	Zero flag
N	3	Negative flag
T	4	Test result flag
OPA	5	Overflow Protection control for A register
OPB	6	Overflow Protection control for B register
–	15–7	Reserved

MSR0[0] (or C) is the carry of AU executions. MSR0[1] (or V) is the overflow flag of AU executions. It is set to 1 if and only if the carry-in into the 16-th bit position of addition/subtraction differs from the carry-out from the 16-th bit position. MSR0[2] (or Z) is the zero flag, which is set to 1 if and only if the AU result is zero. MSR0[3] (or N) is the negative flag. Basically, the most significant bit (MSB) of AU results becomes the N flag. However, if an AU instruction touches the overflow flag (V) like ADD, SUB, CP, *etc*, N flag is updated as exclusive-OR of V and the MSB of the AU result. This implies that even if an AU operation results in overflow, N flag is still valid. T flag is set to 1 if the result of “ETST *cond*.” Instruction is true. MSR0[5] (or OPA) or MSR0[6] (or OPB) enables arithmetic saturation when an arithmetic overflow occurs in A or B register.

Status Register 1 : MSR1

MSR1 consists of status flags of MAU operation, control bit for MAU, and selection bits of EC[I].

Bit Name	Bit	Description
PSH1	0	Multiplier product 1 bit shift control
OPM	1	Overflow Protection control for MA register
M816	2	Multiplication mode control
MV	3	MA overflow flag
SEC0	7–4	EC[0] selection
SEC1	11–8	EC[1] selection
SEC2	15–12	EC[2] selection

MSR1[0] (or PSH1) enables the product to shift by one bit to the left. MSR1[1] (or OPM) controls MA saturation. MSR1[2] (or M816) selects the operating mode for the multiplier. If M816=1, then the multiplier performs 8 by 16 bit to 24 bit multiplication. Otherwise (M816=0), the multiplier performs 16 by 16 bit to 32 bit multiplication in two cycles. MSR1[3] (or MV) is the overflow flag of MAU executions. It is set to 1 if an arithmetic overflow (32-bit overflow) occurs after an arithmetic operation in MAU. It is cleared by a processor reset or “ECR MV” and modified by writing to MSR1. SECi selects the combination of EC[I]. The flag information for the host processor is selected by setting SECi.

Value(of SECi)	Description
0000	EC[I] = Z, Set to 1 if Z flag is 1.
0001	EC[I] = not Z
0010	EC[I] = N
0011	EC[I] = not N
0100	EC[I] = C
0101	EC[I] = not C
0110	EC[I] = V
0111	EC[I] = not V
1000	EC[I] = T
1001	EC[I] = GT
1010	EC[I] = LE
1011	EC[I] = MV
1100	EC[I] = not MV
1101–1111	reserved

HOST INTERFACE

MAC816 is interfaced to the host processor according to CalmRISC coprocessor interface scheme explained below.

CalmRISC supports an efficient and seamless interface with coprocessors. By integrating a MAC (multiply and accumulate) with the CalmRISC core, not only microcontroller functions but also complex digital signal processing algorithms can be implemented in a single development platform (or MDS). CalmRISC has a set of dedicated signal pins, through which data/command/status are exchanged between CalmRISC and a coprocessor. Depicted below are the coprocessor signal pins and a figure of how two processors are interfaced.

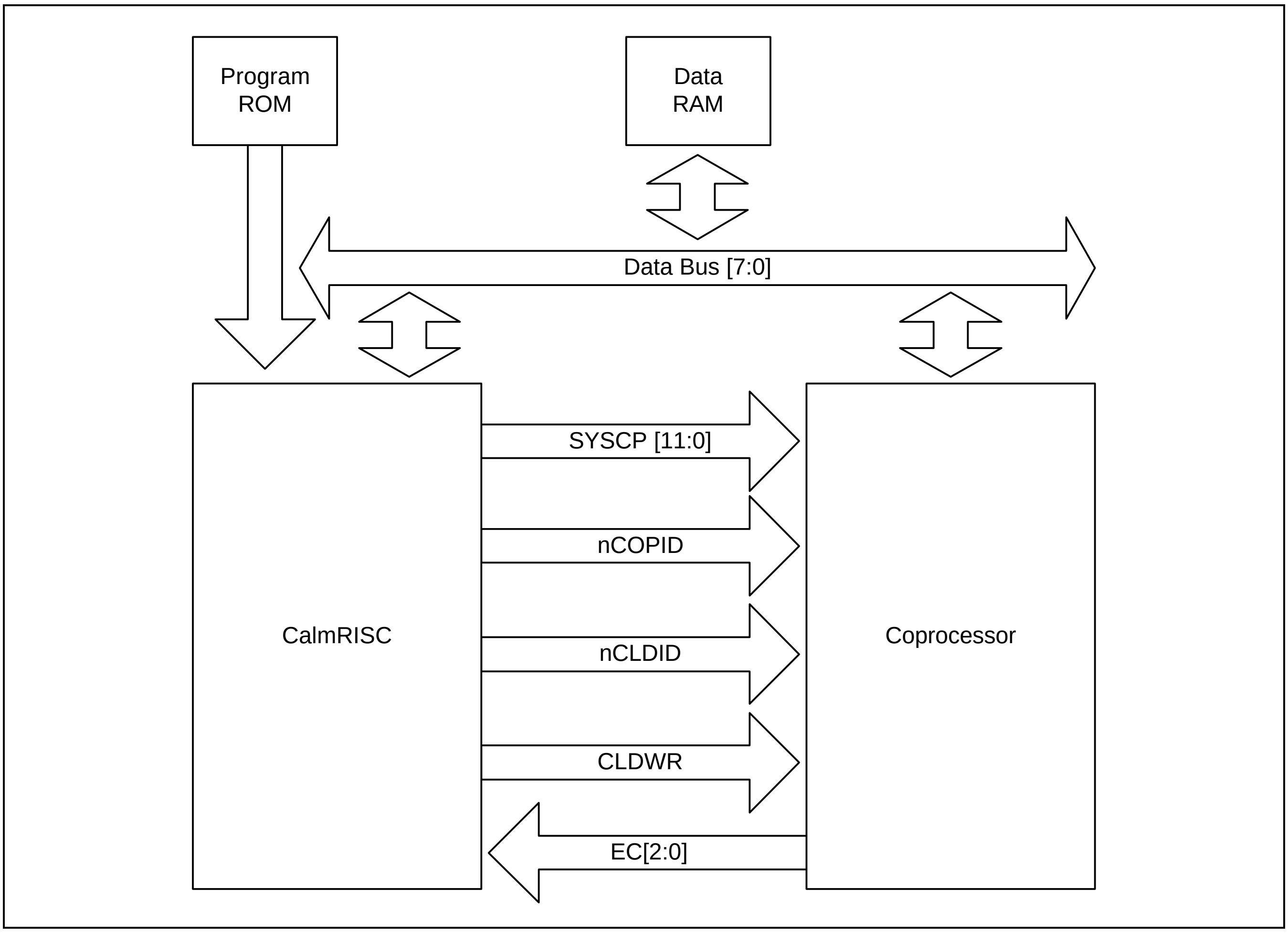


Figure 21-5. Coprocessor Interface Diagram

As shown in the coprocessor interface diagram above, the coprocessor interface signals of CalmRISC are: *SYSCP[11:0]*, *nCOPID*, *nCLDID*, *nCLDWR*, and *EC[2:0]*. The data are exchanged through the data buses, *DI[7:0]* and *DO[7:0]*. CalmRISC issues the command to a coprocessor through *SYSCP[11:0]* in COP instructions. The status of a coprocessor can be sent back to CalmRISC through EC[2:0], and these flags can be checked in the condition codes of branch instructions. The coprocessor instructions are listed in the following table.

Table 21-3. Coprocessor instructions

Mnemonic	Op 1	Op 2	Description
COP	#imm:12	–	Coprocessor operation
CLD	GPR	imm:8	Data transfer from coprocessor into GPR
CLD	imm:8	GPR	Data transfer of GPR to coprocessor
JP(or JR) CALL LNK	EC2–0	label	Conditional branch with coprocessor status flags

The coprocessor of CalmRISC does not have its own program memory (that is, passive coprocessor) as shown in Figure 7 -1. In fact, the coprocessor instructions are fetched and decoded by CalmRISC**,** which issues the command to the coprocessor through the interface signals. For example, if "COP #imm:12" instruction is fetched, then the 12-bit immediate value (imm:12) is loaded on *SYSCP[11:0]* signal with *nCOPID* active in ID/MEM stage, to request the coprocessor to perform the designated operation. The interpretation of the 12-bit immediate value is totally up to the coprocessor. The instruction set of the coprocessor is determined by arranging the 12 bit immediate field. In other words, CalmRISC only provides a set of generic coprocessor instructions, and its installation to a specific coprocessor instruction set can differ from one coprocessor to another. CLD Write instructions
("CLD imm:8, GPR") put the content of a GPR register of CalmRISC on the data bus (*DO[7:0]*) and issue the address(imm:8) of the coprocessor internal register on *SYSCP[7:0]* with *nCLDID* active and *CLDWR* active.
CLD Read instructions ("CLD GPR, imm:8" in Table 1) work similarly, except that the content of the coprocessor internal register addressed by the 8-bit immediate value is read into a GPR register through *DI[7:0]* with *nCLDID* active and *CLDWR* inactive.

The timing diagram given below is a coprocessor instruction pipeline and shows the time the coprocessor performs the required operations. Suppose I_2 is a coprocessor instruction. First, it is fetched and decoded by CalmRISC (at t = T(i-1)). Once it is identified as a coprocessor instruction, CalmRISC indicates to the coprocessor the appropriate command through the coprocessor interface signals (at t = T(i)). Then the coprocessor performs the designated tasks at t = T(i) and t = T(i+1). Hence IF from CalmRISC and then ID/MEM and EX from the coprocessor constitute the pipeline for I_2. Similarly, if I_3 is a coprocessor instruction, the coprocessor's ID/MEM and EX stages replace the corresponding stages of CalmRISC.

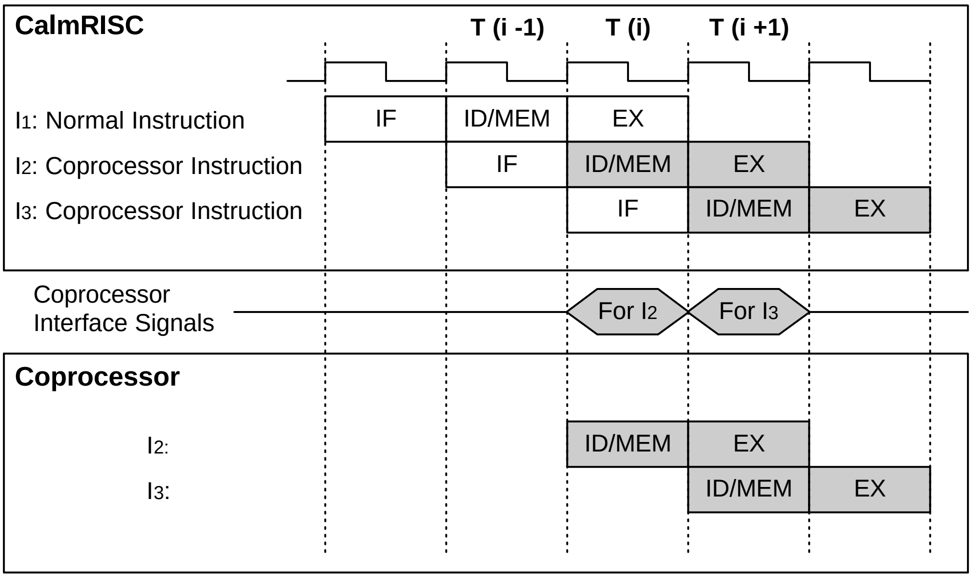


Figure 21-6. Coprocessor Instruction Pipeline

In a multi-processor system, the data transfer between processors is an important factor to determine the efficiency of the overall system. Suppose an input data stream is accepted by a processor, in order to share data with other processors, there should be some efficient mechanism to transfer the data to the processors. In CalmRISC, data is transferred through a single shared data memory. The shared data memory in a multi-processor has some inherent problems such as data hazards and deadlocks. However, the coprocessor in CalmRISC accesses the shared data memory only at the time designated by CalmRISC, a time at which CalmRISC is guaranteed not to access the data memory, and therefore there is no contention over the shared data memory. Another advantage of the proposed scheme is that the coprocessor can access the data memory in its own bandwidth.

INSTRUCTION SET

GLOSSARY

This chapter describes the MAC816 instruction set, and the details of each instruction are listed in alphabetical order . The following notations are used for the description and mnemonics of assembler.

Table 21-4. Notation and Convention

Notation	Interpretation
<opN>	Operand N. N can be omitted if there is only one operand. Typically, <op1> is the destination (and source) operand and <op2> is the source operand.
adr:N	Content of memory location specified by N-bit address
#imm:N	N-bit immediate number
&	Bit-wise AND
|	Bit-wise OR
~	Bit-wise NOT
^	Bit-wise XOR
N**M	Mth power of N
$(N)_M$	M-based number N

Table 21-5. MAC816 Registers

Notation	Operand Code	Mnemonic	Descriptions
Mreg	0000–0010	–	Reserved
	0011	MARN	MA[31:16] + MA[15], MA higher word with round-off
	0100	Y	Y[15:0], multiplier Y input register
	0101	X	X[15:0], multiplier X input register
	0110	MAL	MA[15:0], multiplier accumulator lower 16-bits
	0111	MAH	MA[31:16], multiplier accumulator higher 16-bits
	1000	RP0	RP0[15:0], RAM pointer register 0
	1001	RP1	RP0[15:0], RAM pointer register 1
	1010	RP2	RP0[15:0], RAM pointer register 2
	1011	RPD	RAM pointer for short direct addressing
	1100	MC0	Modulo control register 0 for RP0/RP1
	1101	MC1	Modulo control register 1 for RP2
	1110	MSR0	MAC816 status register 0
	1111	MSR1	MAC816 status register 1
Ai	0	A	A[15:0], AU result register A
	1	B	B[15:0], AU result register B
Am	00	A	A[15:0], AU result register A
	01	B	B[15:0], AU result register B
	10	AC	A[15:0], AU result register A with Carry
	11	BC	B[15:0], AU result register B with Carry
MAm	00	A	A[15:0], AU result register A
	01	B	B[15:0], AU result register B
	10	MAL	MA[15:0], multiplier accumulator lower 16-bits
	11	MAH	MA[31:16], multiplier accumulator higher 16-bits

Table 21-5. MAC816 Registers (Continued)

Notation	Operand Code	Mnemonic	Descriptions
Mreg2	000–011	–	Reserved
Mreg2s	100	Y	Y[15:0], multiplier Y input register
Mreg2d	101	X	X[15:0], multiplier X input register
	110	MAL	MA[15:0], multiplier accumulator lower 16-bits
	111	MAH	MA[31:16], multiplier accumulator higher 16-bits
Mreg1	00	RP0	RP0[15:0], RAM pointer register 0
	01	RP1	RP0[15:0], RAM pointer register 1
	10	RP2	RP0[15:0], RAM pointer register 2
	11	RPD	RAM pointer for short direct addressing
Mreg3	00	MC0	Modulo control register 0 for RP0/RP1
	01	MC1	Modulo control register 1 for RP2
	10	MSR0	MAC816 status register 0
	11	MSR1	MAC816 status register 1

Table 21-6. Data Transfer Registers

Notation	Register Address	Descriptions
Creg	00	A[7:0], AU result register A lower 8-bits
	01	A[15:8], AU result register A higher 8-bits
	10	B[7:0], AU result register B lower 8-bits
	11	B[15:8], AU result register B higher 8-bits

Table 21-7. Memory Access Mode Information

Notation	Operand Code	Mnemonic	Descriptions
@rpm	0000	@rp0+	Content of memory location specified by RP0, RP0 post-increment by 1 with modulo mode
	0001	@rp0-	Content of memory location specified by RP0, RP0 post-decrement by 1 with modulo mode
	0010	@rp0s	Content of memory location specified by RP0, RP0 post-modification by +2 or –2 with modulo mode
	0011	@rp0	Content of memory location specified by RP0
	0100	@rp1+	Content of memory location specified by RP1, RP1 post-increment by 1 with modulo mode
	0101	@rp1-	Content of memory location specified by RP1, RP1 post-decrement by 1 with modulo mode
	0110	@rp1s	Content of memory location specified by RP1, RP1 post-modification by +2 or –2 with modulo mode
	0111	@rp1	Content of memory location specified by RP1
	1000	@rp2+	Content of memory location specified by RP2, RP2 post-increment by 1 with modulo mode
	1001	@rp2-	Content of memory location specified by RP2, RP2 post-decrement by 1 with modulo mode
	1010	@rp2s	Content of memory location specified by RP2, RP2 post-modification by +2 or –2 with modulo mode
	1011	@rp2	Content of memory location specified by RP2
	1100–1111	-	Reserved
@rp0m	00	@rp0+	Content of memory location specified by RP0, RP0 post-increment by 1 with modulo mode
	01	@rp0-	Content of memory location specified by RP0, RP0 post-decrement by 1 with modulo mode
	10	@rp0s	Content of memory location specified by RP0, RP0 post-modification by +2 or –2 with modulo mode
	11	@rp0	Content of memory location specified by RP0
@rp2m	00	@rp2+	Content of memory location specified by RP2, RP2 post-increment by 1 with modulo mode
	01	@rp2-	Content of memory location specified by RP2, RP2 post-decrement by 1 with modulo mode
	10	@rp2s	Content of memory location specified by RP2, RP2 post-modification by +2 or –2 with modulo mode
	11	@rp2	Content of memory location specified by RP2

Table 21-8. Condition Code Information

Notation	Operand Code	Mnemonic	Descriptions
cc	0000	Z	Z = 1
	0001	NZ	Z = 0
	0010	C	C = 1
	0011	NC	C = 0
	0100	NEG	N = 1
	0101	POS	N = 0
	0110	V1	V = 1
	0111	V0	V = 0
	1000	–	Reserved
	1001	GT	N = 0 and Z = 0
	1010	LE	N = 1 and Z = 1
	1011	MV1	MV = 1
	1100	MV0	MV = 0
	1101–1111	–	Reserved

Table 21-9. Control Bit Code Information

Notation	Operand Code	Mnemonic	Descriptions
bs	000	OPM	MSR1[1]
	001	PSH1	MSR1[0]
	010	ME0	RP0[15], RP0 modulo mode enable
	011	ME1	RP1[15], RP1 modulo mode enable
	100	M816	MSR1[2]
	101	ME2	RP2[15], RP2 modulo mode enable
	110	OPA	MSR0[5]
	111	OPB	MSR0[6]

Table 21-10. AU operation code information

Notation	Operand Code	Mnemonic	Descriptions
EMOD0	00	ELD/ELDT	Load
	01	EADD/EADDT	Addition
	10	ESUB/ESUBT	Subtraction
	11	ECP/ECPT	Comparison
EMOD1	0000	ERR/ERRT	Rotate right
	0001	ERL/ERLT	Rotate left
	0010	ESR/ESRT	Arithmetic shift right
	0011	ESL/ESLT	Arithmetic shift left
	0100	EINC/EINCT	Increment
	0101	EDEC/EDECT	Decrement
	0110	ENEG/ENEGT	Negation
	0111	ECR/ECRT	Clear
	1000	ENORM/ENORMT	Normalization
	1001	EABS/EABST	Absolution
	1010–1111	–	reserved

Table 21-11. Others

Notation	Operand Code	Mnemonic	Descriptions
sXsY	00	uu	Unsigned by unsigned multiplication
	01	us	Unsigned by signed multiplication
	10	su	Signed by unsigned multiplication
	11	none	Signed by signed multiplication
rs	0	ER	Reset
	1	ES	Set
ts	0	ELD/ EMOD1/ EMOD0	Execute mnemonic always
	1	ELDT/ EMOD1T/ EMOD0T	Execute mnemonic when test result flag (MSR0[4] or T) is set. If T = 0, act as nop.

INSTRUCTION ENCODING

Table 21-12. Instruction Encoding

Instruction	11	10	9	8	7	6	5	4	3	2	1	0	2nd Word
ELD Mreg2,@rpm	00		00		0	Mreg2			rpm				–
ELD @rpm,Mreg2					1								
ELD Mreg3,@rpm			01		00		Mreg3						
ELD @rpm,Mreg3					01								
ELD Mreg1,adr:4					10		Mreg1		adr[3:0]				
ELD adr:4,Mreg1					11		Mreg1						
ESEC0 #imm:4			10		00		00		Imm[3:0]				
ESEC0 #imm:4							01						
ESEC0 #imm:4							10						
ECR MV							11						
ELD Mreg2d,Mreg2s					01		Mreg2d			Mreg2s			
EMOD0 A,#imm:5					1	EMOD0		Imm[4:0]					
ELD adr:6,MAm			11		adr[5:4]		MAm		adr[3:0]				
ELD MAm,adr:6	01		00				MAm						
EADD Am,adr:6			01				Am						
ESUB Am,adr:6			10										
ECP Am,adr:6			11										
ELD Mreg,Am	10		00		00				Mreg				
ELD Am,Mreg					01								
ELD/ELDT @rpm,Am					1	ts			rpm				
EMOD1/EMOD1T Am			01		0				EMOD1				
EMOD0/EMOD0T Am,MAm					1				MAm		EMOD0		
EMOD0/EMOD0T Am,@rpm			1	EMOD0					rpm				

Table 21-12. Instruction Encoding (Continued)

Instruction	11	10	9	8	7	6	5	4	3	2	1	0	2nd Word
ELD Mreg,#imm:16	11		00		Mreg				Imm[15:12]				Imm[11:0]
EMOD0 Am,#imm:16			01		EMOD0		Am						
EMAD @rp0m,@rp2m,sXsY			10		00		rp0m		rp2m		sXsY		–
EMSB @rp0m,@rp2m,sXsY					01								
EMUL @rp0m,@rp2m,sXsY					10								
EMUL Ai,@rp2m,sXsY					11		0	Ai					
EMUL X,@rp2m,sXsY							10						
EMUL @rp0mY,,sXsY							11		rp0m				
EMAD Ai,@rp2m,sXsY			11		00		0	Ai	rp2m				
EMAD X,@rp2m,sXsY							10						
EMAD @rp0m,Y,sXsY							11		rp0m				
EMSB Ai,@rp2m,sXsY					01		0	Ai	rp2m				
EMSB X,@rp2m,sXsY							10						
EMSB @rp0m,Y,sXsY							11		rp0m				
EMAD X,Y,sXsY					10		00		00				
EMSB X,Y,sXsY									01				
EMUL X,Y,sXsY									10				
ESR MA							01		00		xx		
ESL MA									01				
ERND MA									10				
ENOP							1	xxxxx					
ERPM rpm							00		rpm				
ER/ES bs					11		01		rs	bs			
ETST cc							10		cc				
ELD RPDN,#imm:4							11		Imm[3:0]				

NOTE: "X" means not applicable.

QUICK REFERENCE

Table 21-13. Quick Reference

Operation	Operand1	Operand2	Function	Flag

ELD
EADD
ESUB
ECP
	A	#imm:5	op1 ← op2
			op1 ← op1 + op2
			op1 ← op1 - op2
			op1 - op2
				–
				c.z,v,n
				c,z,v,n
				c,z,v,n

ELD	RPDN	#imm:4	RPD[7:4] ← op2	
ELD	Adr:6	Am/MAm	op1 ← op2	
ELD	Am/MAm	Adr:6	op1 ← op2	

EADD
ESUB
ECP
	Am	Adr:6	op1 ← op1 + op2
			op1 ← op1 - op2
			op1 - op2
				c.z,v,n
				c,z,v,n
				c,z,v,n

ELD	Mreg1	Adr:4	op1 ← op2	–
ELD	Adr:4	Mreg1	op1 ← op2	–
ELD	Am	Mreg	op1 ← op2	–
ELD	mreg	Am	op1 ← op2	–
ELD	Mreg2d	Mreg2s	op1 ← op2	–
ELD	Mreg2	@rpm	op1 ← op2	–
ELD	@rpm	Mreg2	op1 ← op2	–
ELD EADD ESUB ECP ELDT EADDT ESUBT ECPT	Am	MAm	op1 ← op2 op1 ← op1 + op2 op1 ← op1 - op2 op1 - op2 If T=1, same as ELD If T=1, same as EADD If T=1, same as ESUB If T=1, same as ECP	– c.z,v,n c,z,v,n c,z,v,n – c.z,v,n c,z,v,n c,z,v,n
ELD ELDT	@rpm	Am	op1 ← op2 If T=1, same as ELD	–
ELD EADD ESUB ECP ELDT EADDT ESUBT ECPT	Am	@rpm	op1 ← op2 op1 ← op1 + op2 op1 ← op1 - op2 op1 - op2 If T=1, same as ELD If T=1, same as EADD If T=1, same as ESUB If T=1, same as ECP	– c.z,v,n c,z,v,n c,z,v,n – c.z,v,n c,z,v,n c,z,v,n

Table 21-13. Quick Reference (Continued)

Operation	Operand1	Operand2	Function	Flag
ETST	cc	–	MSR0[4] ← cc (condition check)	–
ELD	mreg	#imm:16	op1 ← op2	–

ELD
EADD
ESUB
ECP
	A	#imm:16	op1 ← op2
			op1 ← op1 + op2
			op1 ← op1 - op2
			op1 - op2
				–
				c.z,v,n
				c,z,v,n
				c,z,v,n

ERPM	rpm	–	RP ← modified RP	–
ER	bs	–	op1 ← 0	–
ES	bs	–	op1 ← 10	–
ESEC0 ESEC1 ESEC2	MSR1	#imm:4	MSR1[7:4] ← imm[3:0] MSR1[11:8] ← imm[3:0] MSR1[15:12] ← imm[3:0]	–
ERR ERRT	Am	–	when Am!=AC/BC, op ← {op1}>>1, op1[15] ← op1[0], c ← op1[0] when Am=AC/BC, op1← {c:op1}>>1, c ← op1[0] when t=1, same as ERR	c,z,v,n c,z,v,n
ERL ERLT	Am	–	when Am!=AC/BC, op←{op1}<<1, op1[0]←op[15], c←op[15], when Am=AC/BC, op1←{op1:c}<<1, c←op[15] when t=1, same as ERL	c,z,v,n c,z,v,n
ESR ESRT	Am	–	when Am!=AC/BC, op ← {op1}>>1, c ← op1[0] when Am=AC/BC, op1 ← {c:op1}>>1, c ← op1[0] when t=1, same as ESR	c,z,v,n c,z,v,n
ESL ESLT	Am	–	when Am!=AC/BC, op1 ← {op1}<<1, op1[0] ← 0, c ← op[15], when Am=AC/BC, op1 ← {op1:c}<<1, c ← op[15] when t=1, same as ESL	c,z,v,n c,z,v,n
EINC EINCT	Am	–	when Am!=AC/BC, op1 ← op1+1 when Am=AC/BC, op1 ← op1+c when t=1, same as EINC	c,z,v,n c,z,v,n
EDEC EDECT	Am	–	when Am!=AC/BC, op1 ← op1+ffffh when Am=AC/BC, op1 ← op1+ffffh+c when t=1, same as EDEC	c,z,v,n c,z,v,n
ENEG ENEGT	Am	–	when Am!=AC/BC, op1 ← ~op1+1 when Am=AC/BC, op1 ← ~op1+c when t=1, same as ENEG	c,z,v,n c,z,v,n
EABS EABST	Am	–	when Am!=AC/BC, if op[15]=1, op1 ← ~op1+1 when Am=AC/BC, op[15]=1, op1 ← ~op1+c when t=1, same as EABS	c,z,v,n c,z,v,n
ENORM ENORMT	Am	–	when Am!=AC/BC, if op1[15]^op1[14]=0, op1 ← {op1}<<1, op1[0] ← 0, RP0 ← RP0+1 when Am=AC/BC, if op1[15]^op1[14]=0, op1 ← {op1:c}<<1, RP0 ← RP0+1 when t=1, same as ENORMT	c,z,v,n c,z,v,n
ECR ECRT	Am	–	op1 ← 0 when t=1, same as ECR	–

Table 21-13. Quick Reference (Concluded)

Operation	Operand1	Operand2	Operand3	Function	Flag
ESR	MA	–	–	op1 ← op1>>1	–
ESL	MA	–	–	op1 ← op1<<1	MV
ERND	MA	–	–	MA[31:16] ← MA[31:16] + MA[15]	MV
EMAD	MA	@rp0m	@rp2m	X-reg ← @rp0m, Y-reg ← @rp2m, MA ← MA+X*Y	MV
EMSB	MA	@rp0m	@rp2m	X-reg ← @rp0m, Y-reg ← @rp2m, MA ← MA-X*Y	MV
EMUL	MA	@rp0m	@rp2m	X-reg ← @rp0m, Y-reg ← @rp2m, MA ← (X*Y)	–
EMAD	MA	Ai	@rp2m	X-reg ← op2, Y-reg ← @rp2m, MA ← MA+X*Y	MV
EMSB	MA	Ai	@rp2m	X-reg ← op2, Y-reg ← @rp2m, MA ← MA-X*Y	MV
EMUL	MA	Ai	@rp2m	X-reg ← op2, Y-reg ← @rp2m, MA ← (X*Y)	–
EMAD	MA	X	@rp2m	Y-reg ← @rp2m, MA ← MA+X*Y	MV
EMSB	MA	X	@rp2m	Y-reg ← @rp2m, MA ← MA-X*Y	MV
EMUL	MA	X	@rp2m	Y-reg ← @rp2m, MA ← (X*Y)	–
EMAD	MA	@rp0m	Y	X-reg ← @rp0m, MA ← MA+X*Y	MV
EMSB	MA	@rp0m	Y	X-reg ← @rp0m, MA ← MA-X*Y	MV
EMUL	MA	@rp0m	Y	X-reg ← @rp0m, MA ← (X*Y)	–
EMAD	MA	X	Y	MA ← MA+X*Y	MV
EMSB	MA	X	Y	MA ← MA-X*Y	MV
EMUL	MA	X	Y	MA ← (X*Y)	–

MAC816 INSTRUCTION DESCRIPTION

EABS — Absolute

Format: EABS <op>
<op>: Am

Operation: If the MSB of <op> is 1, <op> ← ~<op> +1 when <op> is A or B.
If the MSB of <op> is 1, <op> ← ~<op> +C when <op> is AC or BC.
EABS adds the values 0 and the 2’s complement of <op>.

Flags: **C:** set if the borrow of result is zero. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: exclusive OR of V and MSB of result.

EABST — Absolute conditional

Format: EABST <op>
<op>: Am

Operation: If T=1, then same as EABS, else no operation

Flags: If T=1, then same as EABS, else no operation

EADD — Add

Format: EADD <op1>, <op2>
<op1>: Am: A, B, AC, BC
<op2>: adr:6, @rpm, Ai, Mreg, #imm:16, #imm:5

Operation: <op1> ← <op1 + <op2> when <op1> is A or B.
<op1> ← <op1 + <op2> + C when <op1> is AC or BC.
EADD adds the values in <op1> and <op2> and stores the result in <op1>.

Flags:
C: set if the carry of result is 1. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: exclusive OR of V and MSB of result.

NOTE: If <op1> is B, <op2> can not be #imm:5.

EADDT — Add conditional

Format: EADDT <op1>, <op2>
<op1>: Am: A, B, AC, BC
<op2>: @rpm, Ai, MAH,MAL

Operation: If T=1, then same as EADD, else no operation

Flags: If T=1, then same as EADD, else no operation

ECP — Compare

Format: ECP <op1>, <op2>
<op1>: Am
<op2>: adr:6, @rpm, Ai, Mreg, #imm:16, #imm:5

Operation: <op1> + ~<op2> +1 when <op1> is A or B.
<op1> + ~<op2> +C when <op1> is AC or BC.
ECP compares the values of <op1> and <op2> by subtracting <op2> from <op1>.
Contents of <op1> and <op2> are not changed.

Flags:
C: set if the borrow of result is zero. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: exclusive OR of V and MSB of result.

NOTE: If <op1> is B, <op2> can not be #imm:5.

ECPT — Compare conditional

Format: ECPT <op1>, <op2>
<op1>: Am: A, B, AC, BC
<op2>: @rpm, Ai, MAH,MAL

Operation: If T=1, then same as ECP, else no operation

Flags: If T=1, then same as ECP, else no operation

ECR — Clear

Format: ECRT <op>
<op>: Ai, MV

Operation: <op> ← 0
ECRT clears Ai or MV.

ECRT — Clear

Format: ECRT <op>
<op>: Ai

Operation: If T=1, <op> ← 0
ECRT clears Ai when T=1.

EDEC — Decrement

Format: EDEC <op>
<op>: Am

Operation: <op> ← <op> + 0xffff when <op> is A or B.
<op> ← <op> + 0xffff + C when <op> is AC or BC.
EDEC decrements the value in <op>.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: exclusive OR of V and MSB of result.

EDECT — Decrement conditional

Format: EDECT <op>
<op>: Am

Operation: If T=1, then same as EDEC, else no operation

Flags: If T=1, then same as EDEC, else no operation

EINC — Increment

Format: EINC <op>
<op>: Am

Operation: <op> ← <op> + 1 when <op> is A or B.
<op> ← <op> + C when <op> is AC or BC.
EINC increments the value in <op>.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: exclusive OR of V and MSB of result.

EINCT — Increment conditional

Format: EINCT <op>
<op>: Am

Operation: If T=1, then same as EINC, else no operation

Flags: If T=1, then same as EINC, else no operation

ELD Adr — Load Adr

Format: ELD <op1>, <op2>
<op1>,<op2>: adr:6, MAi / adr:4,Mreg1

Operation: <op1> ← <op2>
ELD Adr loads a value specified by <op2> into the memory location determined by <op1>

ELD Ai — Load Ai

Format: ELD <op1>, <op2>
<op1>: Ai: A, B
<op2>: adr:6, @rpm, Ai, Mreg, #imm:5, #imm:16

Operation: Ai ← <op2>
ELD Ai loads a value specified by <op2> into the register designated by Ai.

NOTE: If <op1> is B, <op2> can not be #imm:5.

ELD Mreg — Load Mreg

Format: ELD <op1>, <op2>
<op1>: Mreg
<op2>: Ai

Operation: Mreg ← Ai
ELD Mreg loads a value specified by <op2> into the register designated by Mreg.

ELD Mreg1 — Load Mreg1

Format: ELD <op1>, <op2>
<op1>: Mreg1: RP0, RP1, RP2, RPD
<op2>: adr:4

Operation: Mreg1 ← adr:4
ELD Mreg1 loads the content of memory location determined by adr:4 into the register designated by Mreg1.

ELD Mreg2 — Load Mreg2

Format: ELD <op1>, <op2>
<op1>: Mreg2: X, Y, MAH, MAL
<op2>: @rpm

Operation: Mreg2 ← @rpm, rpi ← post-modified rpi
ELD Mreg2 loads the content of memory location determined by @rpm into the register designated by Mreg2.

ELD Mreg3 — Load Mreg3

Format: ELD <op1>, <op2>
<op1>: Mreg3: MC0, MC1, MSR0, MSR1
<op2>: @rpm

Operation: Mreg3 ← @rpm
ELD Mreg3 loads the content of memory location determined by @rpm into the register designated by Mreg3.

ELD @rpm — Load into memory indexed

Format: ELD <op1>, <op2>
<op1>: @rpm
<op2>: Ai, Mreg2, Mreg3

Operation: @rpm ← <op2>, rpi ← post-modified rpi
ELD @rpm loads the value of <op2> into the memory location determined by @rpm.

EMAD — Multiplication and Addition

Format: EMAD <op1>, <op2>,sXsY
<op1>,<op2>: @rp0m,@rp2m / Ai,@rp2m / X,@rp2m / @rp0m,Y / X,Y

Operation: X ← <op1>, Y ← <op2>, MA ← MA + {sign,X}*{sign,Y}
EMAD multiplies the values in <op1> and <op2> and adds the result in MA.

Flags: **MV:** Set if the arithmetic overflow occurs in MA after this instruction.

EMSB — Multiplication and Subtraction

Format: EMSB <op1>, <op2>,sXsY
<op1>,<op2>: @rp0m,@rp2m / Ai,@rp2m / X,@rp2m / @rp0m,Y / X,Y

Operation: X ← <op1>, Y ← <op2>, MA ← MA – {sign,X}*{sign,Y}
EMAD multiplies the values in <op1> and <op2> together and subtracts the result in MA.

Flags: **MV:** Set if the arithmetic overflow occurs in MA after this instruction.

EMUL — Multiply

Format: EMUL <op1>, <op2>,sXsY
<op1>,<op2>: @rp0m,@rp2m / Ai,@rp2m / X,@rp2m / @rp0m,Y / X,Y

Operation: X ← <op1>, Y ← <op2>, MA ← {sign,X}*{sign,Y}
EMUL multiplies the values in <op1> and <op2> and stores the result in MA.

ENEG — Negate

Format: ENEG <op>
<op>: Am

Operation: <op> ← ~<op> +1 when <op> is A or B.
<op> ← ~<op> +C when <op> is AC or BC.
ESUB adds the values 0 and the 2's complement of <op> to to negate <op>.

Flags:
C: set if the borrow of result is zero. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: exclusive OR of V and MSB of result.

ENEGT — Negate conditional

Format: ENEGT <op>
<op>: Am

Operation: If T=1, then same as ENEG, else no operation

Flags: If T=1, then same as ENEG, else no operation

ENOP — No operation

Format: ENOP

Operation: No operation

Flags: No operation

ENORM — Normalization step

Format: ENORM <op>

<op>: Am

Operation: If <op>[15] == <op>[14], <op> ← <op> << 1, RP0 ← RP0+1 when <op> is A or B.

If <op>[15] == <op>[14], <op> ← {<op>,C} <<1, RP0 ← RP0+1 when <op> is AC or BC.

Flags:

C: <op>[15] ^ <op>[14]

Z: set if result is zero. Reset if not

V: reset to zero.

N: set if the MSB of result is 1. Reset if not

ENORMT — Normalization step conditional

Format: ENORMT <op>
<op>: Am

Operation: If T=1, then same as ENORM, else no operation

Flags: If T=1, then same as ENORM, else no operation

ER — Bit Reset

Format: ER bs

Operation: bs ← 0
ES resets the specified bit.

ERL — Rotate Left

Format: ERL <op>
<op>: Am

Operation: <op> ← {<op>[14:0],<op>[15]}, C ← <op>[15] when Am is A or B.
<op> ← {<op>[14:0],C}, C ← <op>[15] when Am is AC or BC.
ERL rotates the value of <op> to the left and stores the result back into <op>.
The original MSB of <op> is copied into carry (C).

Flags:
C: set if the MSB of <op> (before shifting) is 1. Reset if not
Z: set if result is zero. Reset if not
V: reset to zero.
N: set if the MSB of result is 1. Reset if not

ERLT — Rotate Left conditional

Format: ERLT <op>
<op>: Am

Operation: If T=1, then same as ERL, else no operation

Flags: If T=1, then same as ERL, else no operation

ERND — Round off

Format: ERND MA

Operation: MA[31:16] ← MA[31:16] + MA[15], MA[15:0] ← 0
ERND adds 0x8000 to the lower 16-bit position of MA and stores the result in MA.

Flags: **MV:** set if overflow is generated. Reset if not

ERPM — Modify Ram pointer

Format: ERPM rpm

Operation: rpi ← modified rpi
ERPM modifies a rpi by rpm.

NOTE: It does not generate a cycle of RAM access.

ERR — Rotate Right

Format: ERR <op>
<op>: Am

Operation: <op> ← {<op>[0], <op>[15:1]}, C ← <op>[0] when Am is A or B.
<op> ← {C, <op>[15:1]}, C ← <op>[0] when Am is AC or BC.
RR rotates the value of <op> to the right and stores the result back into <op>.
The original LSB of <op> is copied into carry (C).

Flags:
C: set if the LSB of <op>(before shifting) is 1. Reset if not
Z: set if result is zero. Reset if not
V: reset to zero.
N: set if the MSB of result is 1. Reset if not

ERRT — Rotate Right conditional

Format: ERRT <op>
<op>: Am

Operation: If T=1, then same as ERR, else no operation

Flags: If T=1, then same as ERR, else no operation

ES — Bit Set

Format: ES bs

Operation: bs ← 1
ES sets the specified bit.

ESEC0 / ESEC1 / ESEC2 — Set SECi

Format:

ESEC0 #imm:4
ESEC1 #imm:4
ESEC2 #imm:4

Operation:

ESEC0: SEC0[3:0] ← #imm:4
ESEC1: SEC1[3:0] ← #imm:4
ESEC2: SEC2[3:0] ← #imm:4

ESL — Shift Left

Format: ESL <op>
<op>:Am

Operation: <op> ← {<op>[14:0],0}, C ← <op>[15] when <op> is A or B.
<op> ← {<op>[14:0],C}, C ← <op>[15] when <op> is AC or BC.
ESL shifts to the left by 1 bit. The MSB of the original <op> is copied into carry(C).

Flags:
C: set if the MSB of <op>(before shifting) is 1. Reset if not
Z: set if result is zero. Reset if not
V: set if overflow is generated. Reset if not.
N: exclusive OR of V and MSB of result.

ESLT — Shift Left conditional

Format: ESLT <op>
<op>: Am

Operation: If T=1, then same as ESL, else no operation

Flags: If T=1, then same as ESL, else no operation

ESR — Shift Right

Format: ESR <op>
<op>:Am

Operation: <op> ← {<op>[15],<op>[15:1]}, C ← <op>[0] when <op> is A or B.
<op> ← {C,<op>[15:1]}, C ← <op>[0] when <op> is AC or BC.
ESR shifts to the right by 1 bit. The LSB of the original <op> is copied into carry(C).

Flags: **C:** set if the LSB of <op>(before shifting) is 1. Reset if not
Z: set if result is zero. Reset if not
V: set to zero
N: set if result is negative. Reset if not

ESRT — Shift Right conditional

Format: ESRT <op>
<op>: Am

Operation: If T=1, then same as ESR, else no operation

Flags: If T=1, then same as ESR, else no operation

ESUB — Subtract

Format: ESUB <op1>, <op2>
<op1>: Am
<op2>: adr:6, @rpm, Ai, Mreg, #imm:16, #imm:5

Operation: <op1> ← <op1> + ~<op2> +1 when <op1> is A or B.
<op1> ← <op1> + ~<op2> +C when <op1> is AC or BC.
ESUB adds the values in <op1> and the 2's complement of <op2>, to perform subtraction on <op1> and <op2>.

Flags:
C: set if the borrow of result is zero. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: exclusive OR of V and MSB of result.

NOTE: If <op1> is B, <op2> can not be #imm:5.

ESUBT — Subtract conditional

Format: ESUBT <op1>, <op2>
<op1>: Am
<op2>: @rpm, Ai, MAH, MAL

Operation: If T=1, then same as ESUB, else no operation

Flags: If T=1, then same as ESUB, else no operation

ETST — Test Condition

Format: ETST cc
cc: Z, NZ, C, NC, NEG, POS, V1, V0, GT, LE, MV1, MV0

Operation: T ← test result
ETST tests the specified condition of a flag.

Flags: **T:** set if test result is true. Reset if not

NOTES

22 ELECTRICAL DATA

OVERVIEW

Table 22-1. Absolute Maximum Ratings

(T_A = 25 °C)

Parameter	Symbol	Conditions	Rating	Unit
Supply voltage	V_{DD}	–	-0.3 to + 6.5	V
Input voltage	V_I	–	-0.3 to V_{DD} + 0.3	V
Output voltage	V_O	–	-0.3 to V_{DD} + 0.3	V
Output current high	I_{OH}	One I/O pin active	-18	mA
		All I/O pins active	-60	
Output current low	I_{OL}	One I/O pin active	+ 30	mA
		Total pin current for port	+ 100	
Operating temperature	T_A	–	-40 to + 85	°C
Storage temperature	T_{STG}	–	-65 to + 150	°C

Table 22-2. D.C. Electrical Characteristics

(T_A = -40 °C to + 85 °C, V_{DD} = 2.2 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Operating voltage	V_{DD}	fxx = 8.2 MHz	3.0	–	5.5	V
		fxx = 4.1 MHz	2.2	–	5.5	
Input high voltage	V_{IH1}	All input pins except V_{IH2}	0.8 V_{DD}	–	V_{DD}	V
	V_{IH2}	X_{IN}, XT_{IN}	V_{DD}-0.1			
Input low voltage	V_{IL1}	All input pins except V_{IL2}	–	–	0.2 V_{DD}	V
	V_{IL2}	X_{IN}, XT_{IN}			0.1	

Table 22-2. D.C. Electrical Characteristics (Continued)

(T_A = -40 °C to + 85 °C, V_{DD} = 2.2 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Output high voltage	V_{OH1}	V_{DD} = 5 V; I_{OH} = -1 mA All output pins except V_{OH2}	V_{DD}-1.0	–	–	V
	V_{OH2}	V_{DD} = 5 V; I_{OH} = -15 mA Port 5	V_{DD}-1.0	–	–	
Output low voltage	V_{OL1}	V_{DD} = 4.5-5.5 V; I_{OL} = 15 mA	–	0.4	2	V
Input high leakage current	I_{LIH1}	V_{IN} = V_{DD} All input pins except I_{LIH2}	–	–	3	uA
	I_{LIH2}	V_{IN} = V_{DD} X_{IN}, XT_{IN}, X_{OUT}, XT_{OUT}			20	
Input low leakage current	I_{LIL1}	V_{IN} = 0 V All input pins except I_{LIL2}	–	–	-3	
	I_{LIL2}	V_{IN} = 0 V X_{IN}, XT_{IN}, X_{OUT}, XT_{OUT}, RESET			-20	
Output high leakage current	I_{LOH}	V_{OUT} = V_{DD} All I/O pins and Output pins	–	–	3	
Output low leakage current	I_{LOL}	V_{OUT} = 0 V All I/O pins and Output pins	–	–	-3	
Pull-up resistor	R_{L1}	V_{IN} = 0 V; V_{DD} = 5 V ± 10% All port, T_A = 25 °C	30	50	70	kΩ
	R_{L2}	V_{IN} = 0 V; V_{DD} = 5 V ± 10% T_A = 25 °C, RESET only	110	210	310	

Table 22-2. D.C. Electrical Characteristics (Concluded)

(T_A = -40 °C to + 85 °C, V_{DD} = 2.2 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
$\lvert V_{DD}–COMi \rvert$ voltage drop (I=0-16)	V_{DC}	V_{DD} = 2.7 to 5.5 V -15 uA per common pin LCNST = 00000000b	–	–	120	mV
$\lvert V_{DD}–SEGi \rvert$ voltage drop (I=0-55)	V_{DS}	V_{DD} = 2.7 to 5.5 V -15 uA per segment pin LCNST = 00000000b	–	–	120	mV
LCD voltage dividing resistor	R_{LCD1}	V_{LCD} = 2.7 to 5.5 V; LCON.3 = 0	40	55	70	KΩ
	R_{LCD2}	V_{LCD} = 2.7 to 5.5 V; LCON.3 = 1	20	28	35	
Total contrast resistor	R_{CNST}	V_{LCD} = 2.7 to 5.5 V; LCNST = 10000000b	–	140	–	
VLC Output voltage	V_{LC1}	V_{LCD} = 2.7 to 5.5 V	V_{DD}-0.2	V_{DD}	V_{DD}+0.2	V
	V_{LC2}	LCD clock = 0 Hz	0.8V_{DD}-0.2	0.8 V_{DD}	0.8V_{DD}+0.2	
	V_{LC3}	LCNST = 00000000b	0.6V_{DD}-0.2	0.6 V_{DD}	0.6V_{DD}+0.2	
	V_{LC4}		0.4V_{DD}-0.2	0.4 V_{DD}	0.4V_{DD}+0.2	
	V_{LC5}		0.2V_{DD}-0.2	0.2 V_{DD}	0.2V_{DD}+0.2	
Supply current [1]	I_{DD1}	Run mode; V_{DD} = 5 V ± 10% **6 MHz** crystal oscillator	–	4	8	mA
		4 MHz crystal oscillator		2.7	5.4	
		V_{DD} = 3 V ± 10% **6 MHz** crystal oscillator	–	2	4	mA
		4 MHz crystal oscillator		1.3	2.6	
	I_{DD2}	Idle mode: V_{DD} = 5 V ± 10 % **6 MHz** crystal oscillator	–	1.2	2.5	mA
		4 MHz crystal oscillator		1.0	2.0	
		Idle mode: V_{DD} = 3 V± 10 % **6 MHz** crystal oscillator		0.5	1.5	mA
		4 MHz crystal oscillator		0.4	1.0	
	I_{DD3}	Sub-run mode; V_{DD} = 3 V± 10 % Main stop, 32 kHz sub-osc.	–	17	34	uA
	I_{DD4}	Sub-idle mode; V_{DD} = 3 V± 10 % Main stop, 32 kHz	–	4.8	10	uA
	I_{DD5}	Stop mode ; V_{DD} = 5 V ± 10 %	–	0.2	3	uA
		V_{DD} = 3 V ± 10 %		0.1	2	

NOTE: Supply current does not include current drawn through internal pull-up resistors or external output current loads and ADC, DAC, BLD, LCD voltage dividing resistor.

Table 22-3. A.C. Electrical Characteristics

(T_A = -40 °C to + 85 °C, V_{DD} = 2.2 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Interrupt input high, low width	t_{INTH}, t_{INTL}	P0, P1 V_{DD} = 5 V	–	200	–	ns
RESET input low width	t_{RSL}	V_{DD} = 5 V ± 10 %	5	–	–	us

NOTE: User must keep a larger value than the min value.

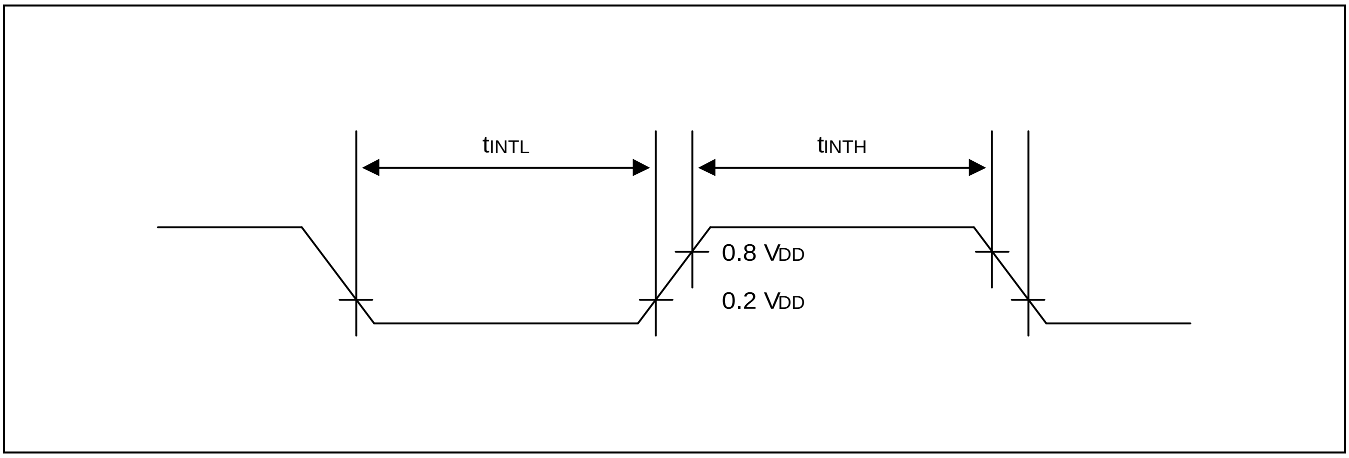


Figure 22-1. Input Timing for External Interrupts (Port 0, Port 1)

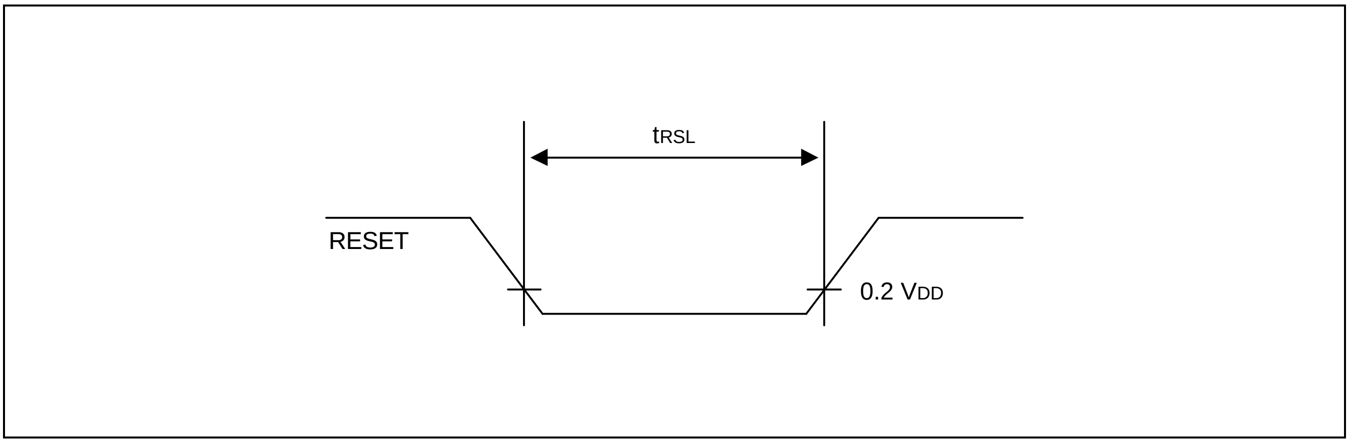
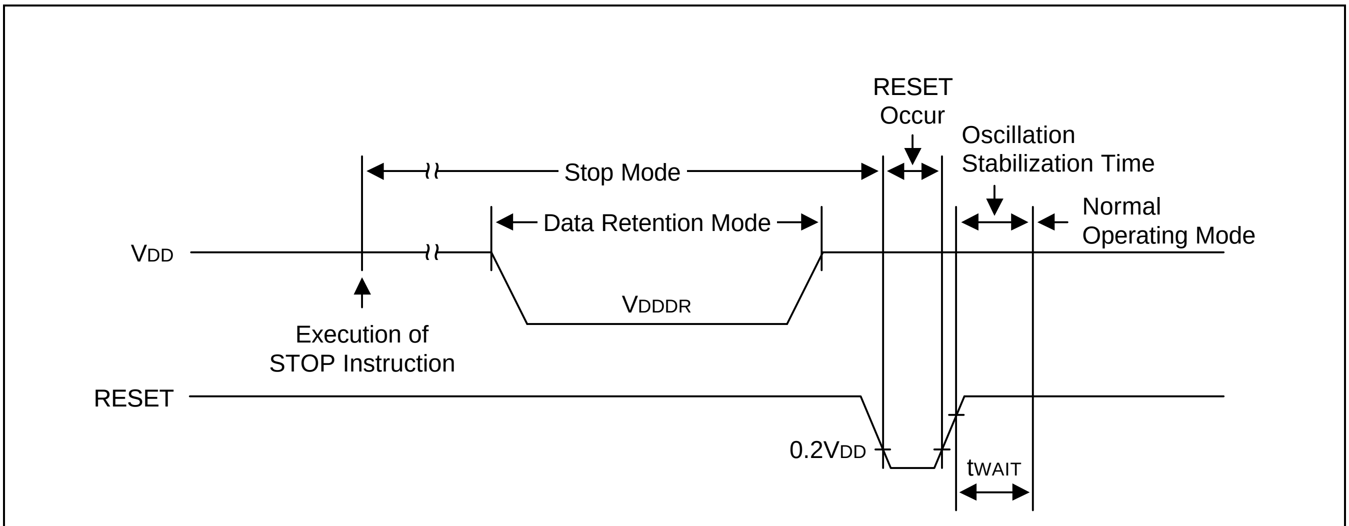


Figure 22-2. Input Timing for RESET

Table 22-4. Data Retention Supply Voltage in Stop Mode

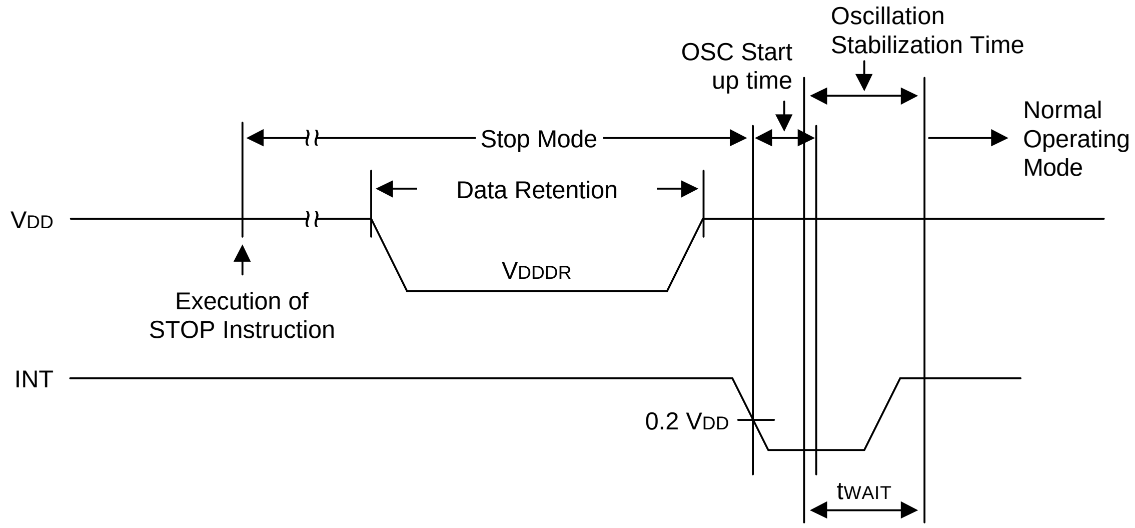
(T_A = -40 °C to + 85 °C)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Data retention supply voltage	V_{DDDR}	–	2.2	–	5.5	V
Data retention supply current	I_{DDDR}	V_{DDDR} = 2.2 V	–	–	2	uA



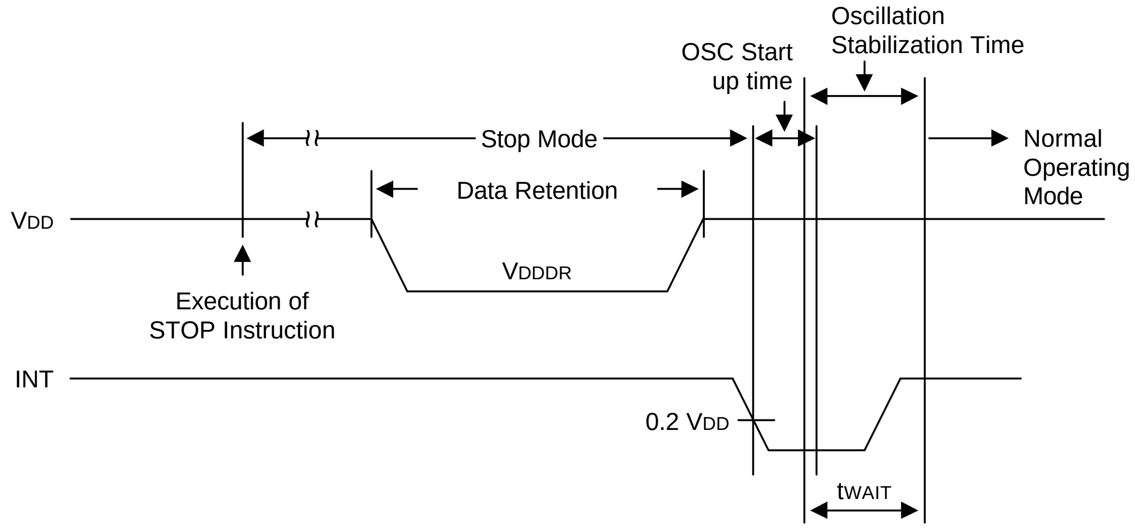
NOTE: tWAIT is same as 2048 x 32 x 1/fxx

Figure 22-3. Stop Mode Release Timing When Initiated by a RESET



NOTE: tWAIT is same as 2048 x 32 x 1/fxx. The value of 2048 which is selected for the clock source of the basic timer can be changed. And then the value of tWAIT will be changed.

Figure 22-4. Stop Mode (Main) Release Timing Initiated by Interrupts



NOTE: tWAIT is same as 256 x 32 x 1/fxx. The oscillator strat up time is less than 100 ms. The value of 256 which is selected for the clock source of basic timer must be kept within this value.

Figure 22-5. Stop Mode (Sub) Release Timing Initiated by Interrupts

Table 22-5. Synchronous SIO Electrical Characteristics

(T_A = -40 °C to + 85 °C V_{DD} = 4.5 V to 5.5 V, V_{SS} = 0 V, fxx = 10 MHz oscillator)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
SCK Cycle time	t_{CYC}	–	200	–	–	ns
Serial Clock High Width	t_{SCKH}	–	60	–	–	
Serial Clock Low Width	t_{SCKL}	–	60	–	–	
Serial Output data delay time	t_{OD}	–	–	–	50	
Serial Input data setup time	t_{ID}	–	40	–	–	
Serial Input data Hold time	t_{IH}	–	100	–	–	

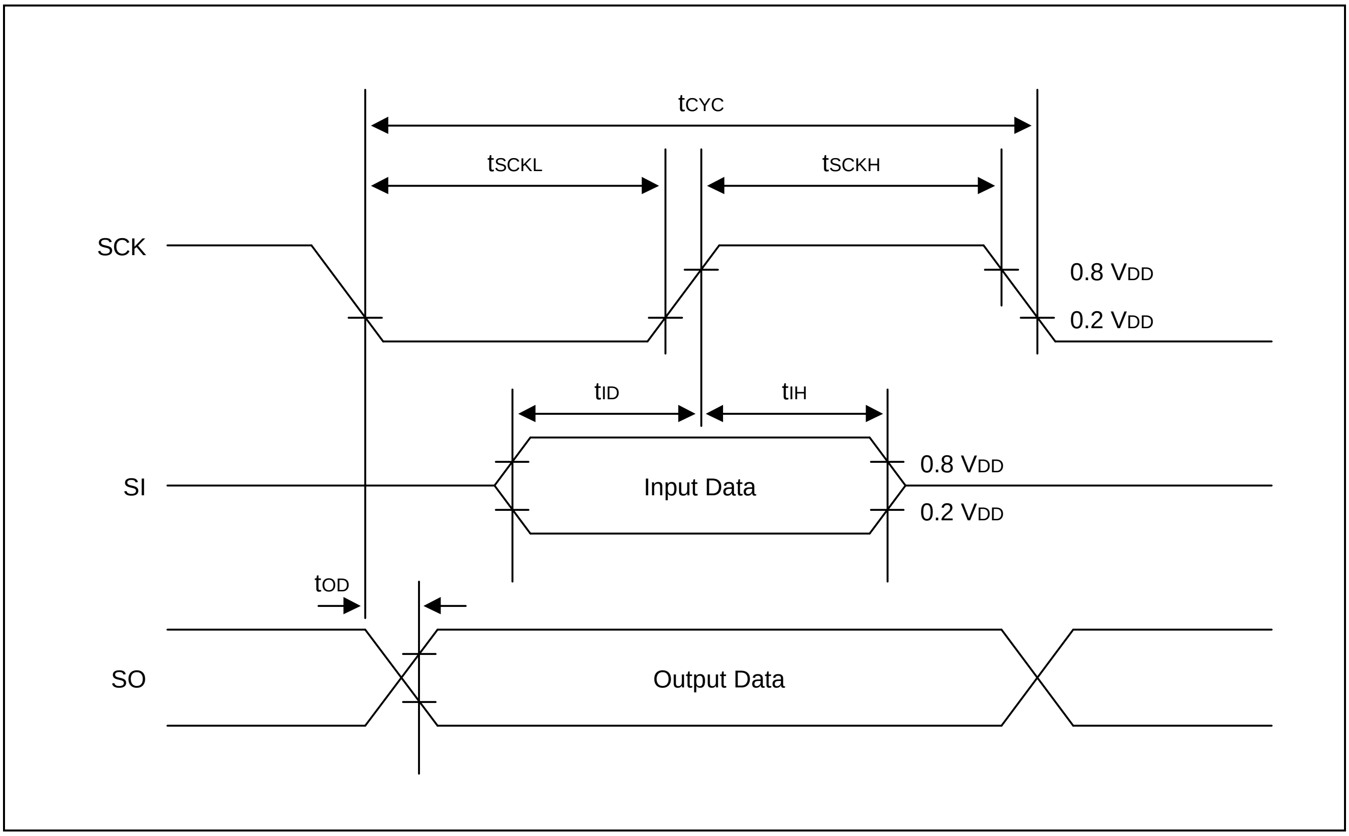


Figure 22-6. Serial Data Transfer Timing

Table 22-6. BLD Electrical Characteristics

(T_A = 25 °C, V_{DD} = 2.2 V to 5.5 V, V_{SS} = 0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
BLD Voltage	VB0	Internal V_{DD} mode	Typ-0.15	2.4	Typ+0.15	V
	VB1			2.7		
	VB2			3.0		
	VB3			3.3		
	VB4		Typ-0.3	4.0	Typ+0.3	
	VB5			4.5		
	VB6	External Input mode, V_{DD} = 2.2 V–3.0 V	Typ-0.15	1.2	Typ+0.15	
	VB7	External Input mode, V_{DD} = 3.0 V–5.5 V	Typ-0.3	1.2	Typ+0.3	
BLD Current	IBLD	V_{DD} = 5.5 V	–	50	100	uA
BLD Response	TB	V_{DD} = 5.5 V	–	1/fw (note)	–	us

NOTE: The fw must be greater than 10 µsec.

Table 22-7. ADC Electrical Characteristics

(T_A = -40 °C to + 85 °C, V_{DD} = 3.0 V to 5.5 V, V_{SS} = 0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
ADC Current	IADC	V_{DD} = 3.3 V	–	1.5	3	mA
Sampling Frequency	–	–	–	8	11	kHz
Resolution	–	Measurement Bandwidth: 20 Hz–4 kHz, Full scale input sine wave: 1 kHz, Sampling frequency: 8 kHz	–	14	–	bits
Signal to Distortion ratio	–		70	75	–	dB
Offset Error	–		–	–	±20	mV
Input Voltage Range	–	V_{DD} = 3.3 V	–	2	–	V_{PP}

NOTE: All the data in this ADC characteristics is measured in the condition of V_{DD} = 3.3 V

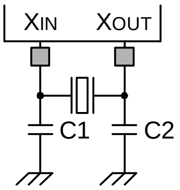
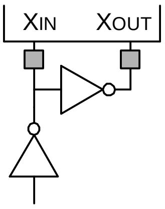
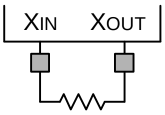
Table 22-8. DAC Electrical Characteristics

(T_A = -40 °C to + 85 °C, V_{DD} = 2.4 V to 5.5 V, V_{SS} = 0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
DAC Current	IDAC	V_{DD} = 5.5 V	–	1.5	3.0	mA
Resolution	–	–	–	8	–	bits
Absolute Accuracy	–		-3	–	3	LSB
Differential Linearity Error	DLE		-1.5	–	1.5	LSB
Output Delay	–		–	–	250	us
Output Load Resistance	Ro		–	10	–	kΩ
Output Level (peak to peak)	–	T_A = -30 °C to + 60 °C	1.2	1.5	1.88	V_{PP}
Regulator Bias voltage	–	V_{DD} = 3.3 V	–	V_{DD}/2	–	V
Output Interval	–	OSC = 4.096 MHz; AD/DA clock input = 8 kHz	–	31	–	us

Table 22-9. Main Oscillator Frequency ($f_{OSC}1$)

(T_A = -40 °C to + 85 °C V_{DD} = 2.2 V to 5.5 V)

Oscillator	Clock Circuit	Test Condition	Min	Typ	Max	Unit
Crystal/Ceramic		V_{DD} = 2.2 V–5.5 V	0.4	–	4.1	MHz
		V_{DD} = 2.4 V–5.5 V			6.2	
		V_{DD} = 3.0 V–5.5 V			8.2	
External clock		V_{DD} = 2.2 V–5.5 V	0.4	–	4.1	MHz
		V_{DD} = 2.4 V–5.5 V			6.2	
		V_{DD} = 3.0 V–5.5 V			8.2	
RC		R = 20 Kohm, V_{DD} = 5 V	–	2	–	MHz

NOTE: Oscillation frequency and Xin input frequency data are for oscillator characteristics only.

Table 22-10. Main Oscillator Clock Stabilization Time (T_{ST1})

(T_A = -40 °C + 85 °C, V_{DD} = 4.5 V to 5.5 V)

Oscillator	Test Condition	Min	Typ	Max	Unit
Crystal	V_{DD} = 4.5 V to 5.5 V	–	–	10	ms
Ceramic	Stabilization occurs when V_{DD} is equal to the minimum oscillator voltage range. V_{DD} = 4.5 V to 5.5 V	–	–	4	ms
External clock	X_{IN} input high and low level width (t_{XH}, t_{XL})	50	–	–	ns

NOTE: Oscillation stabilization time (T_{ST1}) is the time required for the CPU clock to return to its normal oscillation frequency after a power-on occurs, or when Stop mode is ended by a RESET signal.

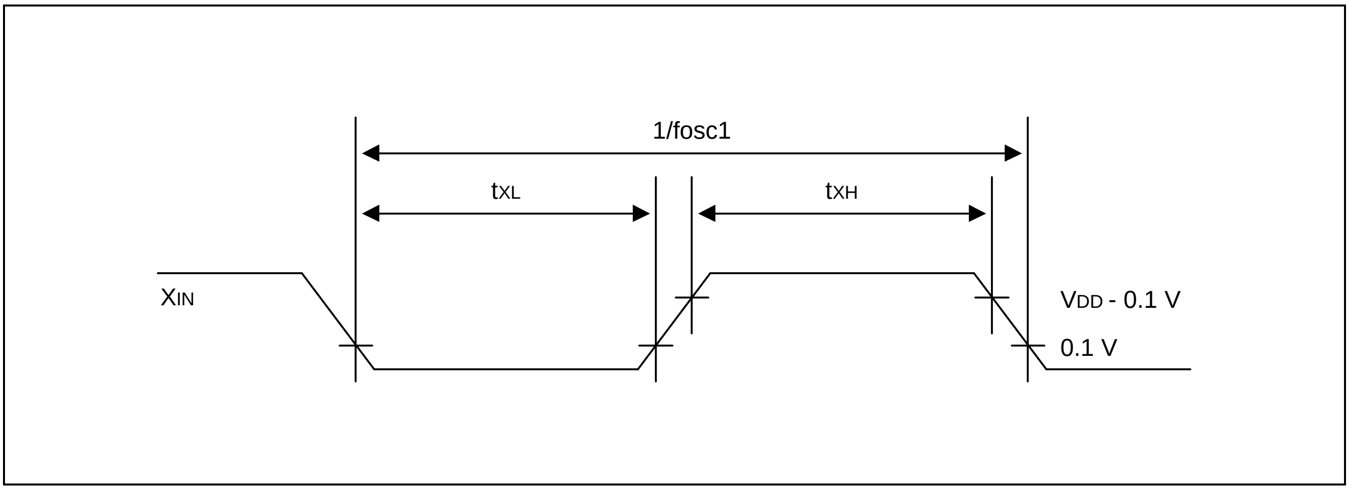
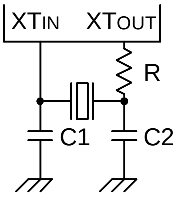


Figure 22-7. Clock Timing Measurement at X_{IN}

Table 22-11. Sub Oscillator Frequency (f_{OSC2})

(T_A = -40 °C + 85 °C, V_{DD} = 2.2 V to 5.5 V)

Oscillator	Clock Circuit	Test Condition	Min	Typ	Max	Unit
Crystal		Crystal oscillation frequency C1 = 22 pF, C2 = 33 pF R = 39 KΩ XT_{IN} and XT_{OUT} are connected with R and C by soldering.	32	32.768	35	kHz

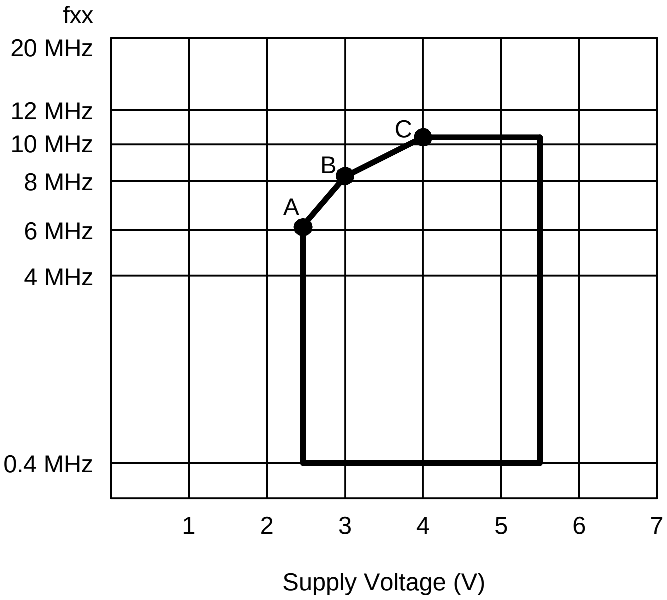
NOTE: Oscillation frequency and XTin input frequency data are for oscillator characteristics only.

Table 22-12. Sub Oscillator (Crystal) Start up Time (t_{ST2})

(T_A = -40 °C + 85 °C, V_{DD} = 2.2 V to 5.5 V)

Oscillator	Test Condition	Min	Typ	Max	Unit
Normal mode	V_{DD} = 4.5 V to 5.5 V	–	1	2	sec
	V_{DD} = 2.2 V to 4.5 V	–	–	10	
Strong mode	V_{DD} = 3.0 V to 5.5 V	–	–	6	
	V_{DD} = 2.2 V to 3.0 V	–	–	2	

NOTE: Oscillation stabilization time (t_{ST2}) is the time required for the oscillator to it's normal oscillation when stop mode is released by interrupts.



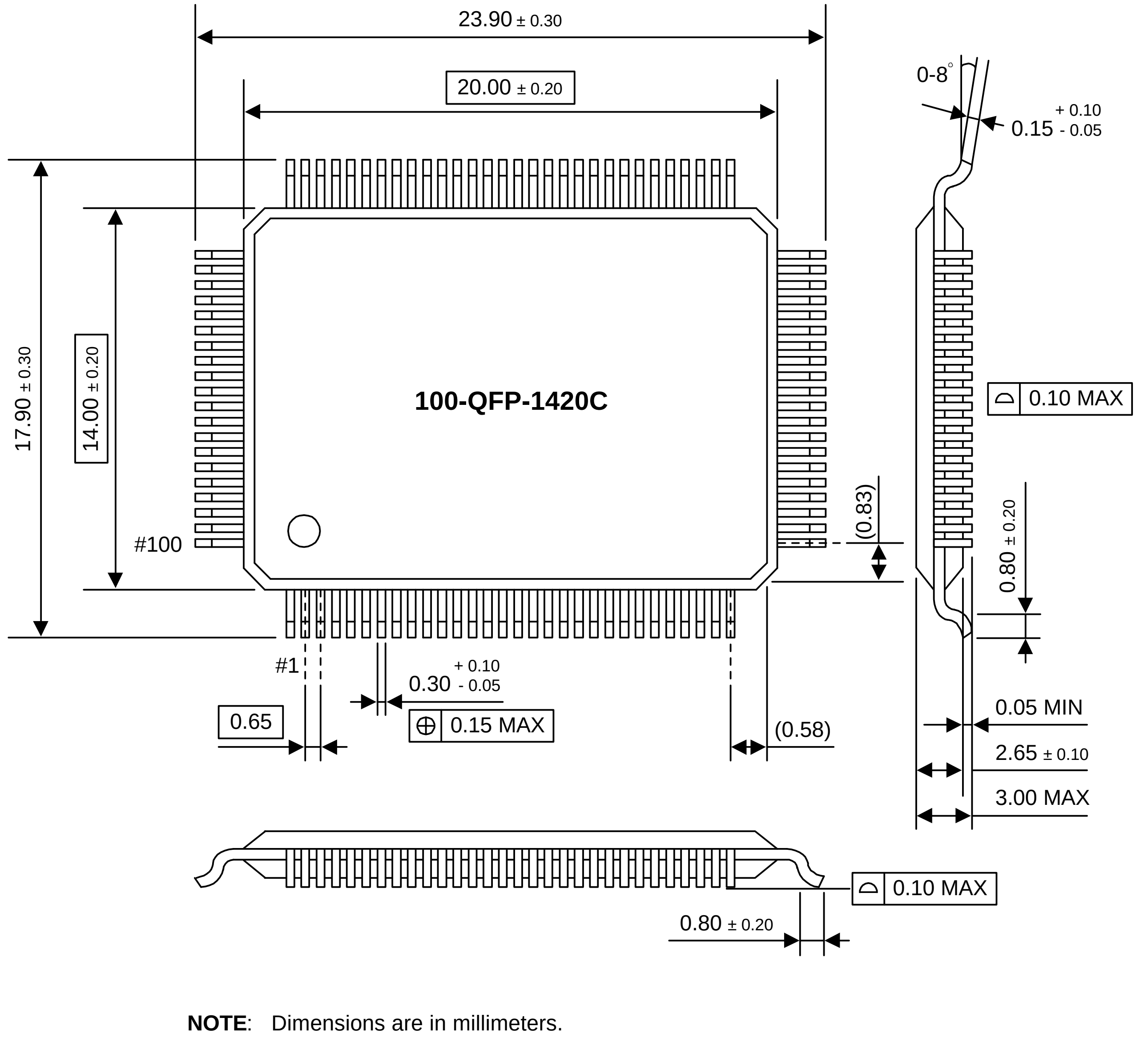
Minimum instruction clock = 1/(1 x oscillator frequency)
A = 2.4 V: 6.2 MHz
B = 3.0 V: 8.2 MHz
C = 4.0 V: 10.24 MHz

Figure 22-8. Operating Voltage Range

23 MECHANICAL DATA

OVERVIEW

The S3CB519/FB519 microcontroller is currently available in a 100-pin QFP package.



NOTE: Dimensions are in millimeters.

Figure 23-1. 100-QFP-1420C Package Dimensions

24 S3FB519

OVERVIEW

The S3FB519 single-chip CMOS microcontroller is the FLASH version of the S3CB519 microcontroller.
It has an on-chip FLASH ROM instead of masked ROM. The FLASH ROM is accessed by serial data format.

The S3FB519 is fully compatible with the S3CB519, both in function and in pin configuration. Because of its simple programming requirements, the S3FB519 is ideal for use as an evaluation chip for the S3CB519.

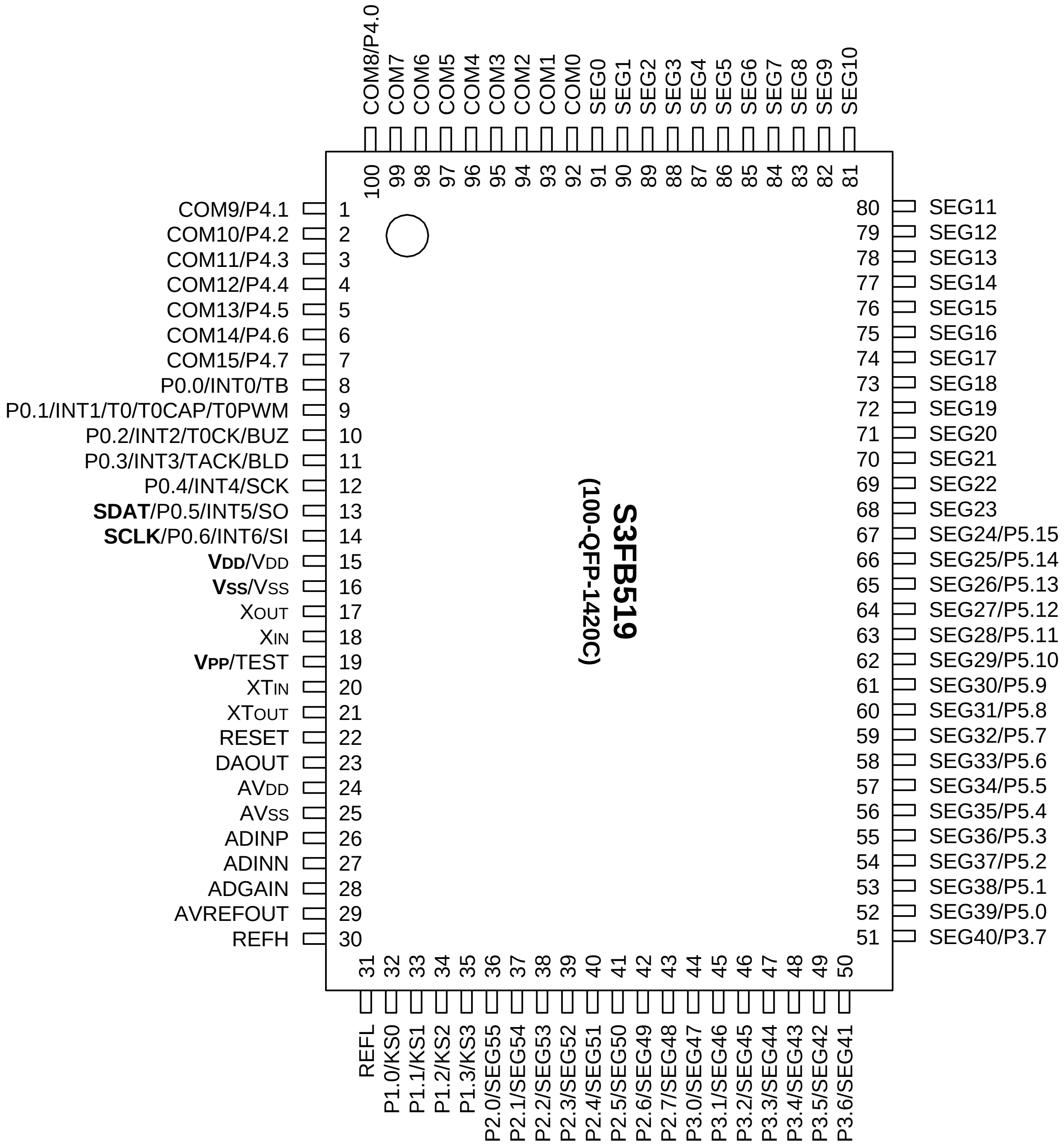


Figure 24-1. S3FB519 Pin Assignments (100-QFP)

Table 24-1. Descriptions of Pins Used to Read/Write the FLASH ROM

Main Chip	During Programming			
Pin Name	Pin Name	Pin No.	I/O	Function
P0.5	SDAT	13	I/O	Serial data pin. Output port when reading and input port when writing. Can be assigned as a Input/push-pull output port.
P0.6	SCLK	14	I/O	Serial clock pin. Input only pin.
TEST	V_{PP} (TEST)	19	I	Power supply pin for FLASH ROM cell writing (indicates that FLASH enters into the writing mode). When 12.5 V is applied, FLASH is in writing mode and when 5 V is applied, FLASH is in reading mode. When FLASH is operating, hold GND.
RESET	RESET	22	I	Chip initialization
V_{DD}/V_{SS}	V_{DD}/V_{SS}	15/16	I	Logic power supply pin. V_{DD} should be tied to +5 V during programming.

NOTE: Pin No. is for 100-QFP type package.

Table 24-2. Comparison of S3FB519 and S3CB519 Features

Characteristic	S3FB519	S3CB519
Program Memory	32-Kbyte FLASH ROM	32-Kbyte mask ROM
Operating Voltage (V_{DD})	2.2 V to 5.5 V	2.2 V to 5.5 V
FLASH Programming Mode	V_{DD} = 5 V, V_{PP} (TEST) = 12.5V	
Pin Configuration	100-QFP	100-QFP
FLASH ROM Programmability	User program	Programmed at the factory

NOTES