S3C80M4/F80M4

8-BIT CMOS
MICROCONTROLLERS
USER'S MANUAL

Revision 1

SAMSUNG
ELECTRONICS

Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. Samsung assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained herein.

Samsung reserves the right to make changes in its products or product specifications with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

This publication does not convey to a purchaser of semiconductor devices described herein any license under the patent rights of Samsung or others.

Samsung makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Samsung assume any liability arising out of the application or use of any product or circuit and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

"Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by the customer's technical experts.

Samsung products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, for other applications intended to support or sustain life, or for any other application in which the failure of the Samsung product could create a situation where personal injury or death may occur.

Should the Buyer purchase or use a Samsung product for any such unintended or unauthorized application, the Buyer shall indemnify and hold Samsung and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorized use, even if such claim alleges that Samsung was negligent regarding the design or manufacture of said product.

S3C80M4/F80M4 8-Bit CMOS Microcontrollers
User's Manual, Revision 1
Publication Number: 21-S3-C80M4/F80M4-052005

Samsung Electronics' microcontroller business has been awarded full ISO-14001 certification (BSI Certificate No. FM24653). All semiconductor products are designed and manufactured in accordance with the highest quality standards and objectives.

Samsung Electronics Co., Ltd.
San #24 Nongseo-Ri, Giheung- Eup
Yongin-City, Gyeonggi-Do, Korea
C.P.O. Box #37, Suwon 449-900

TEL: (82)-(031)-209-1934
FAX: (82)-(031)-209-1889

Home-Page URL: Http://www.samsungsemi.com
Printed in the Republic of Korea

Preface

The S3C80M4/F80M4 *Microcontroller User's Manual* is designed for application designers and programmers who are using the S3C80M4/F80M4 microcontroller for application development. It is organized in two main parts:

Part I	Programming Model	Part II	Hardware Descriptions

Part I contains software-related information to familiarize you with the microcontroller's architecture, programming model, instruction set, and interrupt structure. It has six chapters:

Chapter 1	Product Overview	Chapter 4	Control Registers
Chapter 2	Address Spaces	Chapter 5	Interrupt Structure
Chapter 3	Addressing Modes	Chapter 6	Instruction Set

Chapter 1, "Product Overview," is a high-level introduction to S3C80M4/F80M4 with general product descriptions, as well as detailed information about individual pin characteristics and pin circuit types.

Chapter 2, "Address Spaces," describes program and data memory spaces, the internal register file, and register addressing. Chapter 2 also describes working register addressing, as well as system stack and user-defined stack operations.

Chapter 3, "Addressing Modes," contains detailed descriptions of the addressing modes that are supported by the S3C8-series CPU.

Chapter 4, "Control Registers," contains overview tables for all mapped system and peripheral control register values, as well as detailed one-page descriptions in a standardized format. You can use these easy-to-read, alphabetically organized, register descriptions as a quick-reference source when writing programs.

Chapter 5, "Interrupt Structure," describes the S3C80M4/F80M4 interrupt structure in detail and further prepares you for additional information presented in the individual hardware module descriptions in Part II.

Chapter 6, "Instruction Set," describes the features and conventions of the instruction set used for all S3C8-series microcontrollers. Several summary tables are presented for orientation and reference. Detailed descriptions of each instruction are presented in a standard format. Each instruction description includes one or more practical examples of how to use the instruction when writing an application program.

A basic familiarity with the information in Part I will help you to understand the hardware module descriptions in Part II. If you are not yet familiar with the S3C8-series microcontroller family and are reading this manual for the first time, we recommend that you first read Chapters 1-3 carefully. Then, briefly look over the detailed information in Chapters 4, 5, and 6. Later, you can reference the information in Part I as necessary.

Part II "hardware Descriptions," has detailed information about specific hardware components of the S3C80M4/F80M4 microcontroller. Also included in Part II are electrical, mechanical, flash, and development tools data. It has 10 chapters:

Chapter 7	Clock Circuit	Chapter 12	8-bit PWM Timer
Chapter 8	RESET and Power-Down	Chapter 13	Electrical Data
Chapter 9	I/O Ports	Chapter 14	Mechanical Data
Chapter 10	Basic Timer	Chapter 15	S3F80M4 Flash MCU
Chapter 11	8-bit Timer 0	Chapter 16	Development Tools

Two order forms are included at the back of this manual to facilitate customer order for S3C80M4/F80M4 microcontrollers: the Mask ROM Order Form, and the Mask Option Selection Form. You can photocopy these forms, fill them out, and then forward them to your local Samsung Sales Representative.

Table of Contents

Part I — Programming Model

Chapter 1 Product Overview

Chapter 2 Address Spaces

Table of Contents (Continued)

Chapter 3 Addressing Modes

Chapter 4 Control Registers

Chapter 5 Interrupt Structure

Chapter 6 Instruction Set

Table of Contents (Continued)

Part II Hardware Descriptions

Chapter 7 Clock Circuit

Chapter 8 RESET and Power-Down

Chapter 9 I/O Ports

Chapter 10 Basic Timer

Table of Contents (Continued)

Chapter 11 8-bit Timer 0

Chapter 12 8-bit Pulse Width Modulation

Chapter 13 Electrical Data

Chapter 14 Mechanical Data

Chapter 15 S3F80M Flash MCU

Chapter 16 Development Tools

List of Figures

List of Figures (Continued)

List of Figures (Concluded)

List of Tables

List of Tables (Continued)

List of Programming Tips

Description **Page Number**

Chapter 2: Address Spaces

Chapter 7: Clock Circuit

List of Register Descriptions

List of Instruction Descriptions

List of Instruction Descriptions (Continued)

1 PRODUCT OVERVIEW

S3C8-SERIES MICROCONTROLLERS

Samsung's S3C8 series of 8-bit single-chip CMOS microcontrollers offers a fast and efficient CPU, a wide range of integrated peripherals, and various mask-programmable ROM sizes. Among the major CPU features are:

— Efficient register-oriented architecture

— Selectable CPU clock sources

— Idle and Stop power-down mode release by interrupt

— Built-in basic timer with watchdog function

A sophisticated interrupt structure recognizes up to eight interrupt levels. Each level can have one or more interrupt sources and vectors. Fast interrupt processing (within a minimum of four CPU clocks) can be assigned to specific interrupt levels.

S3C80M4/F80M4 MICROCONTROLLER

The S3C80M4/F80M4 single-chip CMOS microcontroller is fabricated using the highly advanced CMOS process, Its design is based on the SAM88RC CPU core. Stop and Idle (Power-down) modes were implemented to reduce power consumption.

The S3C80M4 is a microcontroller with a 4K-byte mask-programmable ROM embedded.
The S3F80M4 is a microcontroller with a 4K-byte Flash ROM embedded.

Using a proven modular design approach, Samsung engineers have successfully developed the S3C80M4/F80M4 by integrating the following peripheral modules with the powerful SAM8 core:

— Two programmable I/O ports, including one 8-bit port, one 7-bit port (Total 15 pins).

— Four bit-programmable pins for external interrupts.

— One 8-bit basic timer for oscillation stabilization and watchdog functions (system reset).

— One 8-bit timer/counter.

— 8-bit high-speed PWM.

FLASH

The S3F80M4 microcontroller is available in Flash version. The S3F80M4 microcontroller has an on-chip FLASH ROM instead of a masked ROM. The S3F80M4 is comparable to the S3C80M4, both in function and in pin configuration.

FEATURES

CPU

- SAM88 RC CPU core

Memory

- Program Memory (ROM)
  - 4K × 8 bits program memory
- Data Memory (RAM)
  - 128 × 8 bits data memory

Instruction Set

- 78 instructions
- Idle and stop instructions added for power-down modes

15 I/O Pins

- 15 normal I/O pins
- Bit programmable ports

Interrupts

- 6 interrupt levels and 6 interrupt sources

8-Bit Basic Timer

- Watchdog timer function
- 4 kinds of clock source

8-Bit Timer/Counter 0

- Programmable 8-bit internal timer
- External event counter function

8-Bit High-Speed PWM

- 8-bit PWM 1-ch
- 6-bit base +2-bit extension

Oscillation Sources

- Crystal, ceramic, or RC for main clock
- Main clock frequency: 0.4 MHz – 10 MHz

Two Power-Down Modes

- Idle: only CPU clock stops
- Stop: selected system clock and CPU clock stop

Power Consumption

- RUM Mode: 4mA at 10MHz, 5V
- Stop Mode: 100uA at 5V

Instruction Execution Times

- 400nS at 10 MHz fosc(minimum)

Operating Temperature Range

- –25°C to +85°C

Operating Voltage Range

- 2.4 V to 5.5 V at 0.4 – 4.2MHz
- 2.7 V to 5.5 V at 0.4 – 10MHz

Package Type

- 20-DIP-300A, 20-SOP-375
- 16-DIP-300A, 16-SOP-375

IVC

- Internal Voltage Converter for 5V operation

BLOCK DIAGRAM

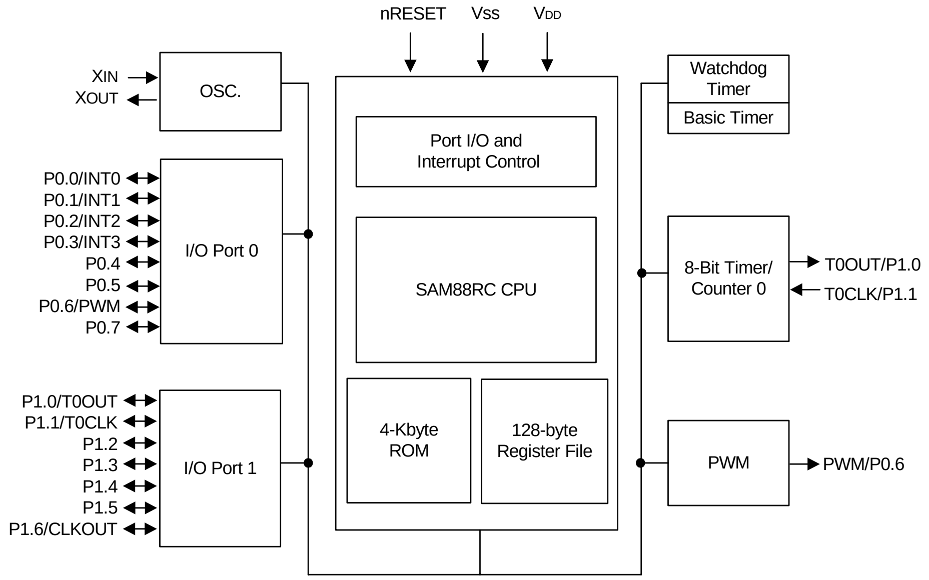


Figure 1-1. Block Diagram

PIN ASSIGNMENT

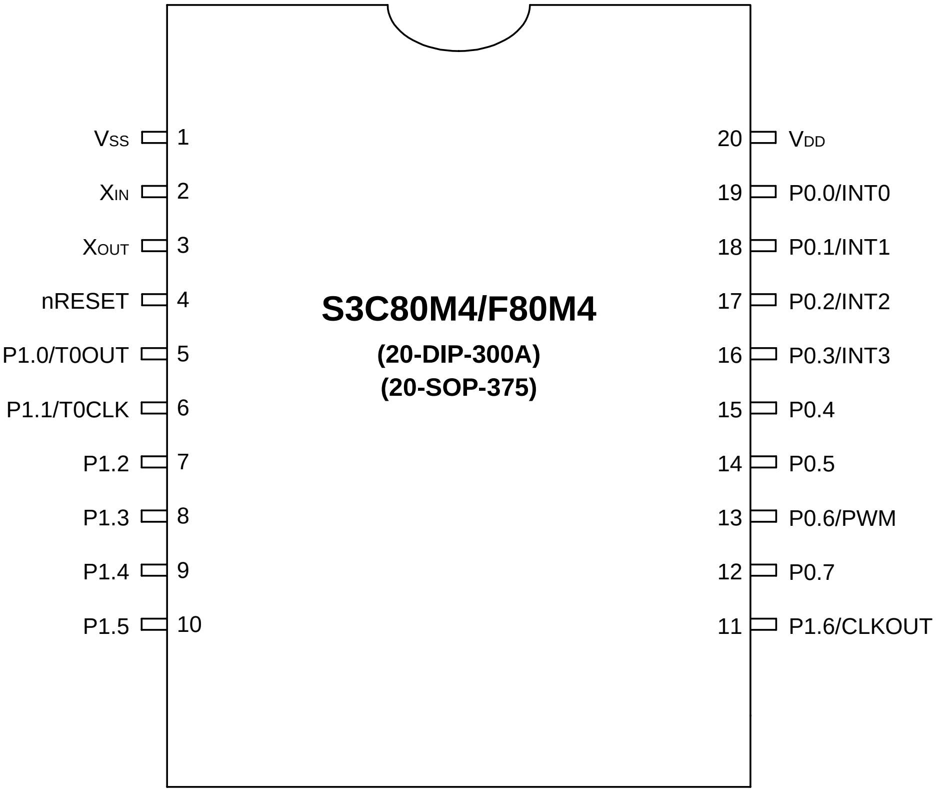


Figure 1-2. S3C80M4/F80M4 Pin Assignments (20-DIP-300A, 20-SOP-375)

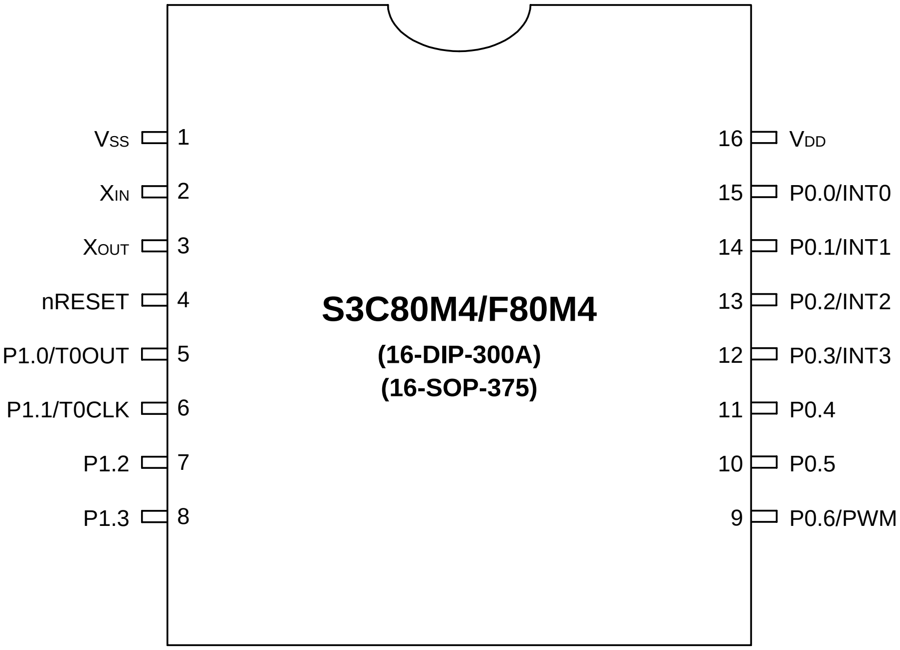


Figure 1-3. S3C80M4/F80M4 Pin Assignments (16-DIP-300A, 16-SOP-375)

PIN DESCRIPTIONS

Table 1-1. S3C80M4/F80M4 Pin Descriptions

Pin Names	Pin Type	Pin Description	Circuit Type	Pin Numbers (note)	Share Pins
P0.0–P0.7	I/O	I/O port with bit-programmable pins; Schmitt trigger input or push-pull output and software assignable pull-ups. Alternately used for external interrupt input (noise filters, interrupt enable and pending control). Port0 pins can also be used as PWM output.	D-4	19–13 (15–9) 12	INT0–INT3 PWM
P1.0 P1.1 P1.2 P1.3	I/O	I/O port with bit-programmable pins; Schmitt trigger input or push-pull, open-drain output and software assignable pull-ups.	E-4	5–8 (5–8)	T0OUT T0CLK
P1.4 P1.5 P1.6	I/O	I/O port with bit-programmable pins; Input or push-pull, open-drain output and software assignable pull-ups.	E-2	9–11	CLKOUT
INT0–INT3	I/O	External interrupts input pins.	D-4	19–16 (15–12)	P0.0–P0.3
T0CLK	I/O	Timer 0 external clock input.	E-4	6(6)	P1.1
T0OUT	I/O	Timer 0 clock output.	E-4	5(5)	P1.0
CLKOUT	I/O	CPU clock output.	E–2	11	P1.6
PWM	I/O	8-Bit high speed PWM output.	D-4	15(13)	P0.6
nRESET	I	System reset pin.	B	4(4)	–
X_{IN}, X_{OUT}	–	Main oscillator pins.	–	2,3 (2,3)	–
V_{DD}, V_{SS}	–	Power input pins. A capacitor must be connected between V_{DD} and V_{SS}.	–	1,20 (1,16)	–

NOTE: Parentheses indicate pin number for 16-DIP-300A/16-SOP-375 package.

PIN CIRCUITS

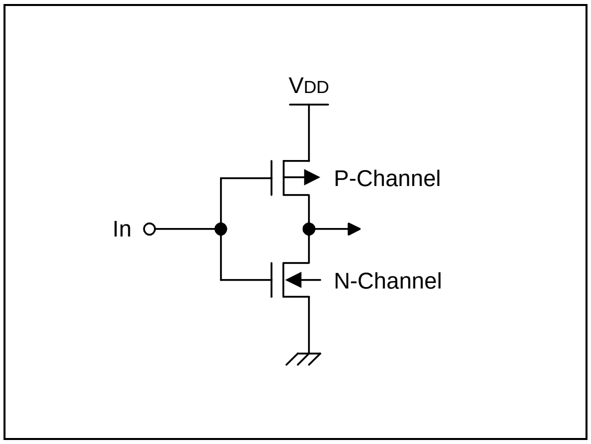


Figure 1-4. Pin Circuit Type A

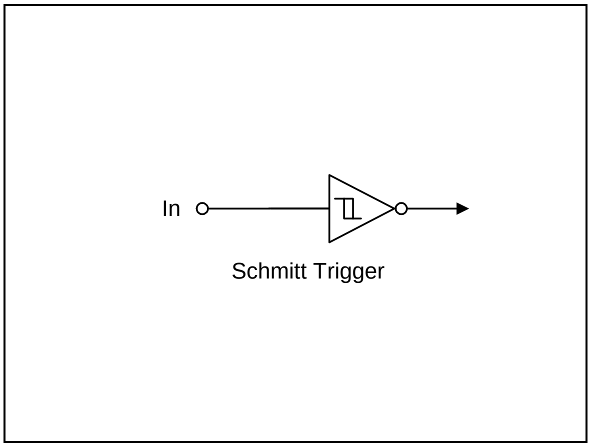


Figure 1-5. Pin Circuit Type B

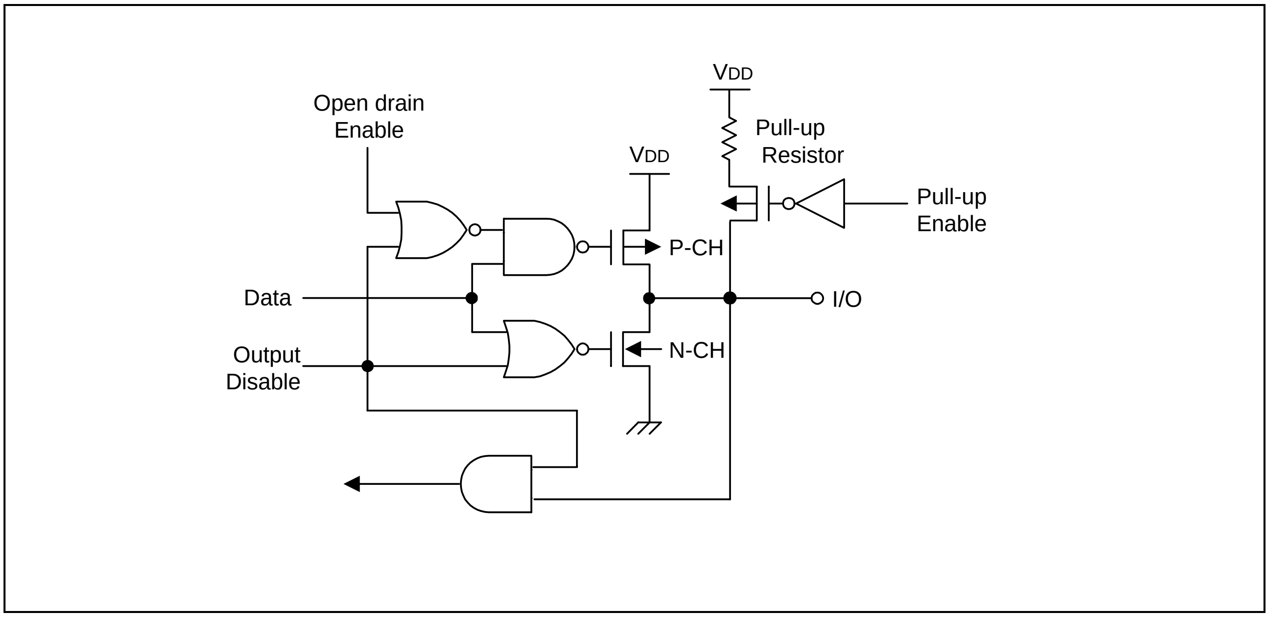


Figure 1-6. Pin Circuit Type E-2 (P1.4–P1.6)

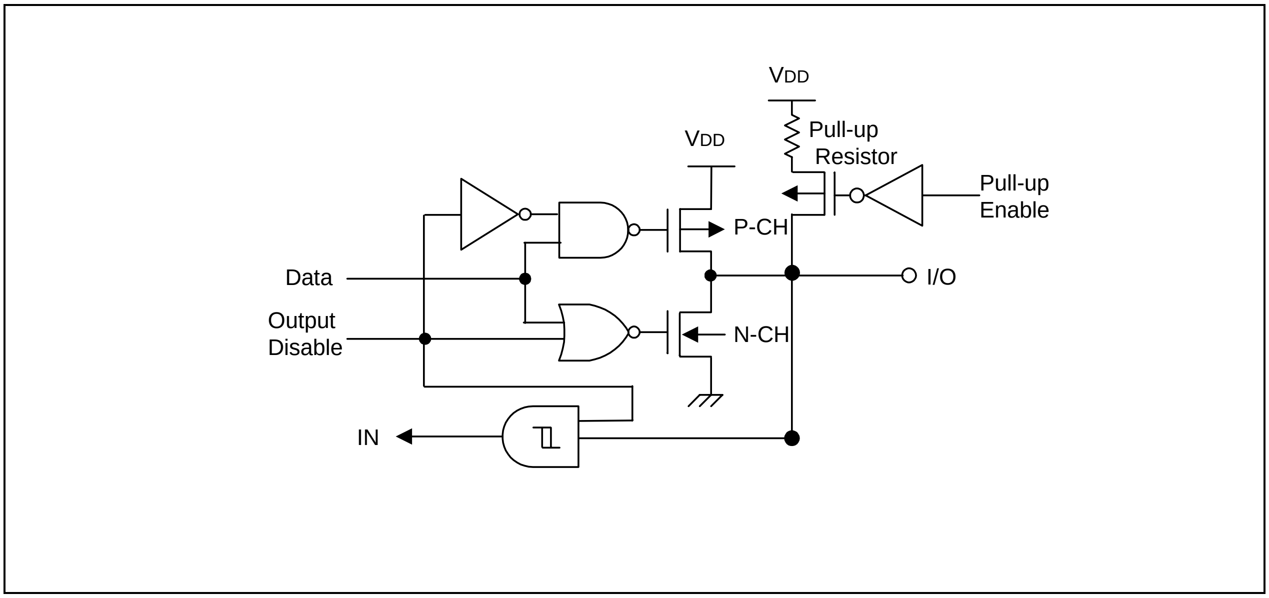


Figure 1-7. Pin Circuit Type D-4 (P0)

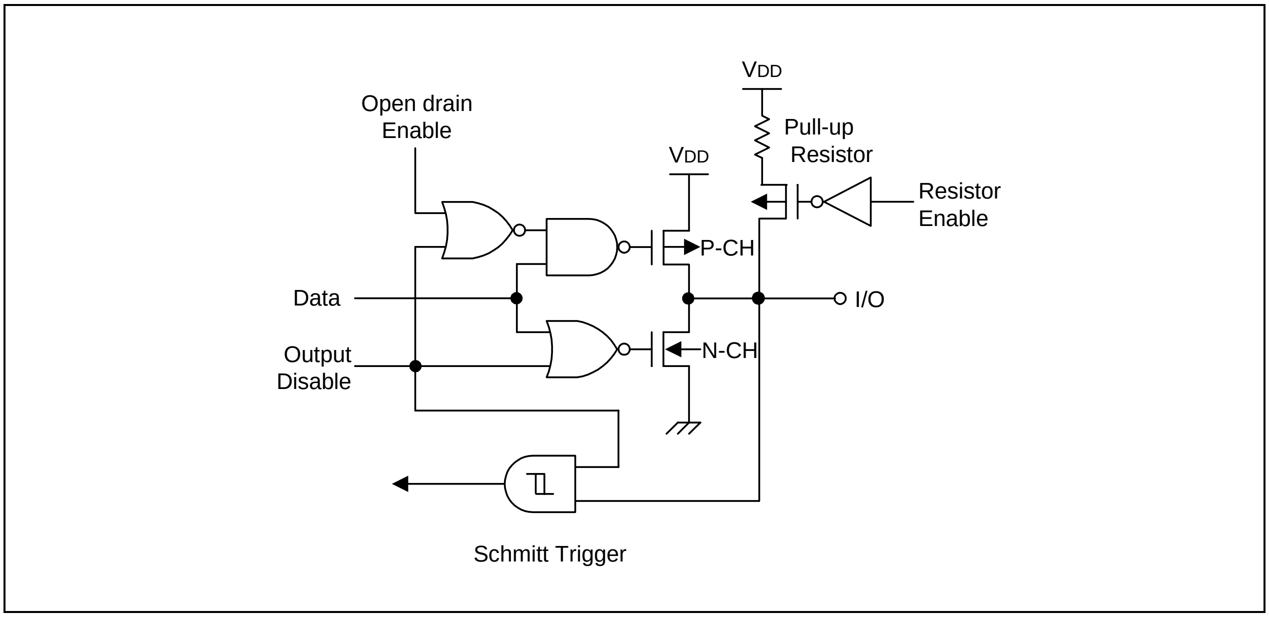


Figure 1-8. Pin Circuit Type E-4 (P1.0-P1.3)

2 ADDRESS SPACES

OVERVIEW

The S3C80M4 microcontroller has two types of address space:

— Internal program memory (ROM)

— Internal register file

A 16-bit address bus supports program memory operations. A separate 8-bit register bus carries addresses and data between the CPU and the register file.

The S3C80M4 has an internal 4-Kbyte mask-programmable ROM.

The 256-byte physical register space is expanded into an addressable area of 320 bytes using addressing modes.

PROGRAM MEMORY (ROM)

Program memory (ROM) stores program codes or table data. The S3C80M4/F80M4 has 4K bytes internal mask-programmable program memory.

The first 256 bytes of the ROM (0H–0FFH) are reserved for interrupt vector addresses. Unused locations in this address range can be used as normal program memory. If you use the vector address area to store a program code, be careful not to overwrite the vector addresses stored in these locations.

The ROM address at which a program execution starts after a reset is 0100H in the S3C80M4.

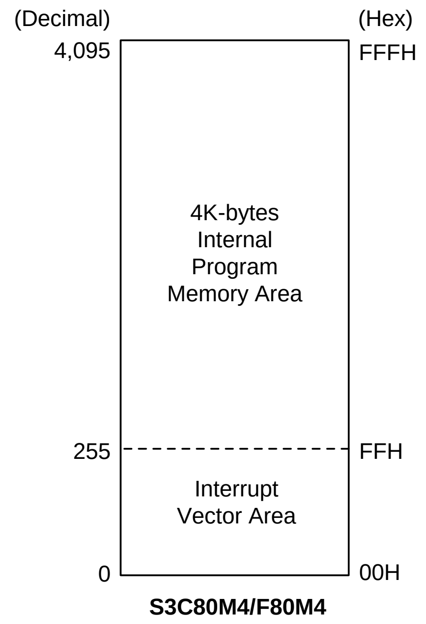


Figure 2-1. Program Memory Address Space

REGISTER ARCHITECTURE

In the S3C80M4/F80M4 implementation, the upper 64-byte area of register files is expanded two 64-byte areas, called *set 1* and *set 2*. The upper 32-byte area of set 1 is further expanded two 32-byte register banks (bank 0 and bank 1), and the lower 32-byte area is a single 32-byte common area.

In case of S3C80M4/F80M4 the total number of addressable 8-bit registers is 175. Of these 175 registers, 13 bytes are for CPU and system control registers, 18 bytes are for peripheral control and data registers, 16 bytes are used as a shared working registers, and 128 registers are for general-purpose use, page 0.

You can always address set 1 register locations, regardless of which of the ten register pages is currently selected. Set 1 locations, however, can only be addressed using register addressing modes.

The extension of register space into separately addressable areas (sets, banks, and pages) is supported by various addressing mode restrictions, the select bank instructions, SB0 and SB1.

Specific register types and the area (in bytes) that they occupy in the register file are summarized in Table 2-1.

Table 2-1. S3C80M4/F80M4 Register Type Summary

Register Type	Number of Bytes
General-purpose registers (including the 16-byte common working register area, one 128-byte prime register area)	144
CPU and system control registers	13
Mapped clock, peripheral, I/O control, and data registers	18
Total Addressable Bytes	175

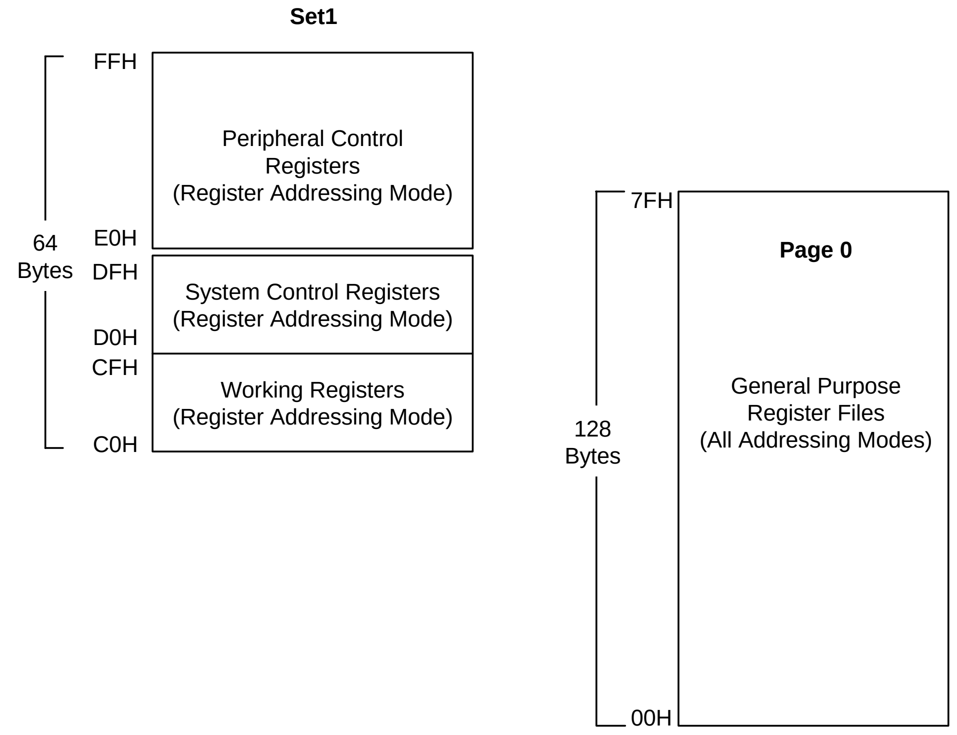
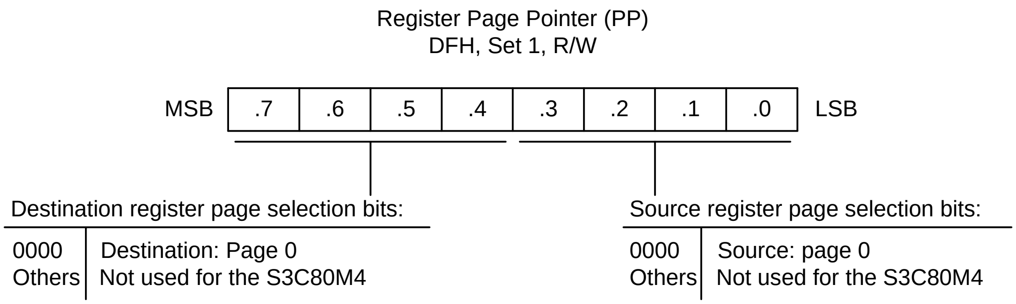

Set1
FFH
Peripheral Control Registers (Register Addressing Mode)
E0H
64 Bytes
DFH
System Control Registers (Register Addressing Mode)
D0H
CFH
Working Registers (Register Addressing Mode)
C0H
7FH
Page 0
General Purpose Register Files (All Addressing Modes)
128 Bytes
00H

Figure 2-2. Internal Register File Organization

REGISTER PAGE POINTER (PP)

The S3C8-series architecture supports the logical expansion of the physical 256-byte internal register file (using an 8-bit data bus) into as many as 16 separately addressable register pages. Page addressing is controlled by the register page pointer (PP, DFH). In the S3C80M4 microcontroller, the register page pointer must be changed to address other pages.

After a reset, the page pointer's source value (lower nibble) and the destination value (upper nibble) are always "0000", automatically selecting page 0 as the source and destination page for register addressing.



NOTE: In the S3C80M4 microcontroller, the internal register file is configured as eleven pages (Pages 0). The pages 0 is used for general purpose register file.

Figure 2-3. Register Page Pointer (PP)

☞ PROGRAMMING TIP — Using the Page Pointer for RAM clear (Page 0, Page 1)

```
          LD      PP,#00H           ;  Destination  ←  0, Source  ←  0
          SRP     #0C0H
          LD      R0,#0FFH          ;  Page 0 RAM clear starts
RAMCL0    CLR     @R0
          DJNZ    R0,RAMCL0
          CLR     @R0               ;  R0 = 00H

          LD      PP,#10H           ;  Destination  ←  1, Source  ←  0
          LD      R0,#0FFH          ;  Page 1 RAM clear starts
RAMCL1    CLR     @R0
          DJNZ    R0,RAMCL1
          CLR     @R0               ;  R0 = 00H
```

NOTE: You should refer to page 6-39 and use DJNZ instruction properly when DJNZ instruction is used in your program.

REGISTER SET 1

The term *set 1* refers to the upper 64 bytes of the register file, locations C0H–FFH.

The upper 32-byte area of this 64-byte space (E0H–FFH) is expanded two 32-byte register banks, *bank 0* and *bank 1*. The set register bank instructions, SB0 or SB1, are used to address one bank or the other. A hardware reset operation always selects bank 0 addressing.

The upper two 32-byte areas (bank 0 and bank 1) of set 1 (E0H–FFH) contains 68 mapped system and peripheral control registers. The lower 32-byte area contains 16 system registers (D0H–DFH) and a 16-byte common working register area (C0H–CFH). You can use the common working register area as a “scratch” area for data operations being performed in other areas of the register file.

Registers in set 1 locations are directly accessible at all times using Register addressing mode. The 16-byte working register area can only be accessed using working register addressing (For more information about working register addressing, please refer to Chapter 3, “Addressing Modes.”)

REGISTER SET 2

The same 64-byte physical space that is used for set 1 locations C0H–FFH is logically duplicated to add another 64 bytes of register space. This expanded area of the register file is called set *2*. For the S3C80M4, the set 2 address range (C0H–FFH) is not accessible.

The logical division of set 1 and set 2 is maintained by means of addressing mode restrictions. You can use only Register addressing mode to access set 1 locations. In order to access registers in set 2, you must use Register Indirect addressing mode or Indexed addressing mode.

The set 2 register area is commonly used for stack operations.

PRIME REGISTER SPACE

The lower 128 bytes (00H–7FH) of the S3C80M4's one 128-byte register pages is called *prime register area.* Prime registers can be accessed using any of the seven addressing modes (see Chapter 3, "Addressing Modes.")

The prime register area is immediately addressable following a reset.

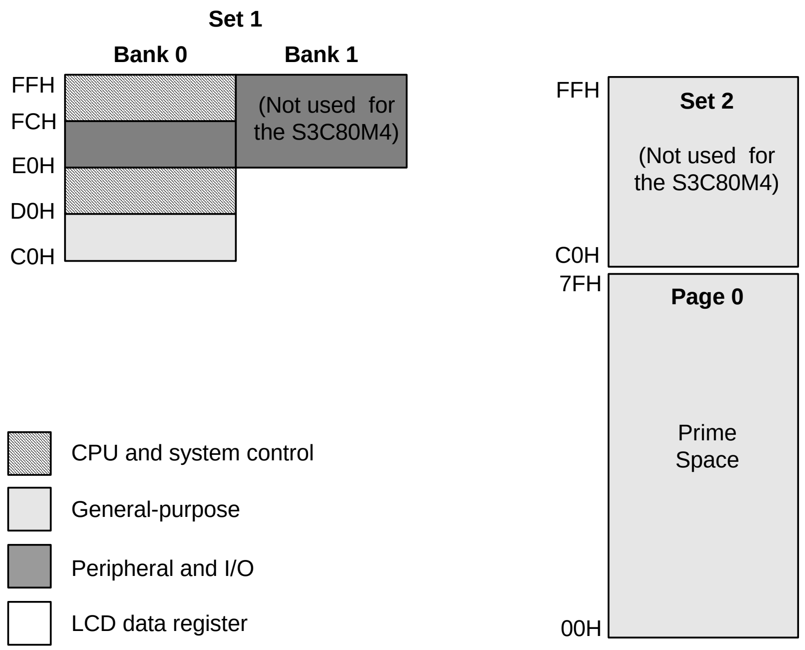


Figure 2-4. Set 1, Set2, Prime Area Register Map

WORKING REGISTERS

Instructions can access specific 8-bit registers or 16-bit register pairs using either 4-bit or 8-bit address fields. When 4-bit working register addressing is used, the 256-byte register file can be seen by the programmer as one that consists of 32 8-byte register groups or "slices." Each slice comprises of eight 8-bit registers.

Using the two 8-bit register pointers, RP1 and RP0, two working register slices can be selected at any one time to form a 16-byte working register block. Using the register pointers, you can move this 16-byte register block anywhere in the addressable register file, except the set 2 area.

The terms slice and block are used in this manual to help you visualize the size and relative locations of selected working register spaces:

— One working register *slice* is 8 bytes (eight 8-bit working registers, R0–R7 or R8–R15)

— One working register *block* is 16 bytes (sixteen 8-bit working registers, R0–R15)

All the registers in an 8-byte working register slice have the same binary value for their five most significant address bits. This makes it possible for each register pointer to point to one of the 24 slices in the register file. The base addresses for the two selected 8-byte register slices are contained in register pointers RP0 and RP1.

After a reset, RP0 and RP1 always point to the 16-byte common area in set 1 (C0H–CFH).

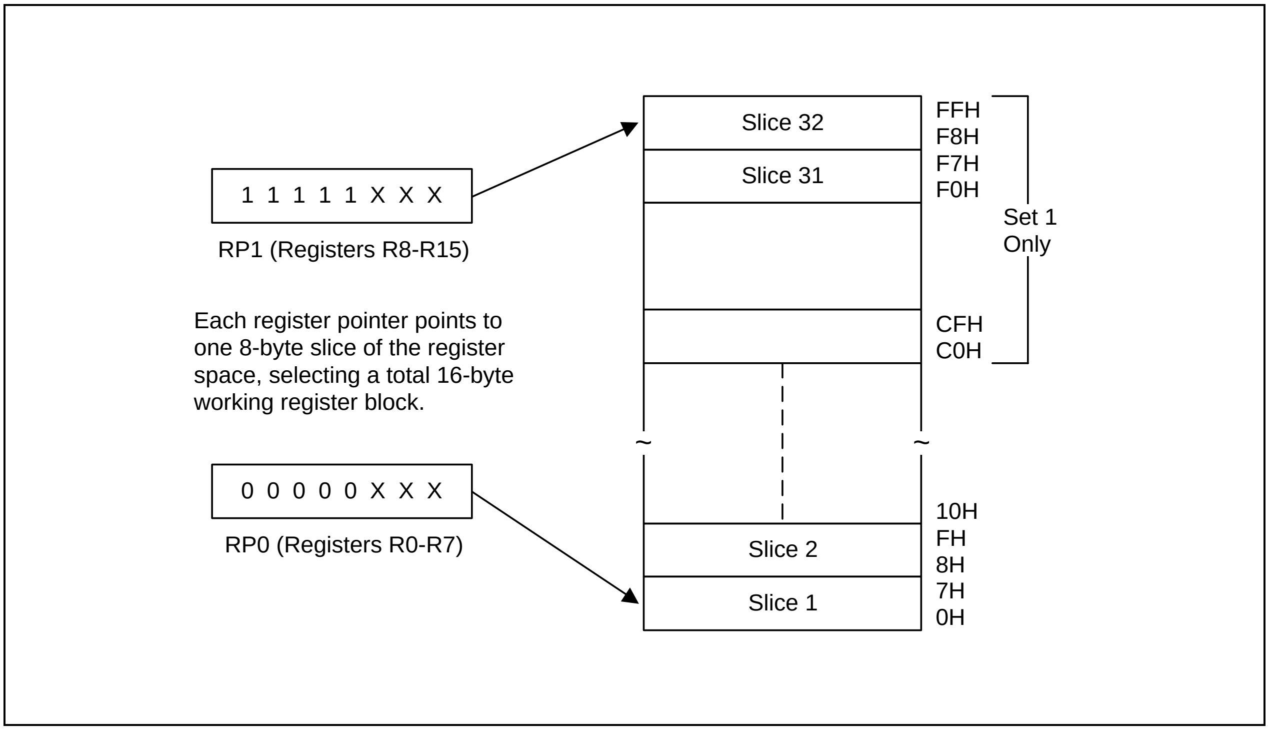


Figure 2-5. 8-Byte Working Register Areas (Slices)

USING THE REGISTER POINTS

Register pointers RP0 and RP1, mapped to addresses D6H and D7H in set 1, are used to select two movable 8-byte working register slices in the register file. After a reset, they point to the working register common area: RP0 points to addresses C0H–C7H, and RP1 points to addresses C8H–CFH.

To change a register pointer value, you load a new value to RP0 and/or RP1 using an SRP or LD instruction. (see Figures 2-6 and 2-7).

With working register addressing, you can only access those two 8-bit slices of the register file that are currently pointed to by RP0 and RP1. You cannot, however, use the register pointers to select a working register space in set 2, C0H–FFH, because these locations can be accessed only using the Indirect Register or Indexed addressing modes.

The selected 16-byte working register block usually consists of two contiguous 8-byte slices. As a general programming guideline, it is recommended that RP0 point to the "lower" slice and RP1 point to the "upper" slice (see Figure 2-6). In some cases, it may be necessary to define working register areas in different (non-contiguous) areas of the register file. In Figure 2-7, RP0 points to the "upper" slice and RP1 to the "lower" slice.

Because a register pointer can point to either of the two 8-byte slices in the working register block, you can flexibly define the working register area to support program requirements.

☞ PROGRAMMING TIP — Setting the Register Pointers

```
SRP     #70H           ;  RP0  ←  70H, RP1  ←  78H
SRP1    #48H           ;  RP0  ←  no change, RP1  ←  48H,
SRP0    #0A0H          ;  RP0  ←  A0H, RP1  ←  no change
CLR     RP0            ;  RP0  ←  00H, RP1  ←  no change
LD      RP1,#0F8H      ;  RP0  ←  no change, RP1  ←  0F8H
```

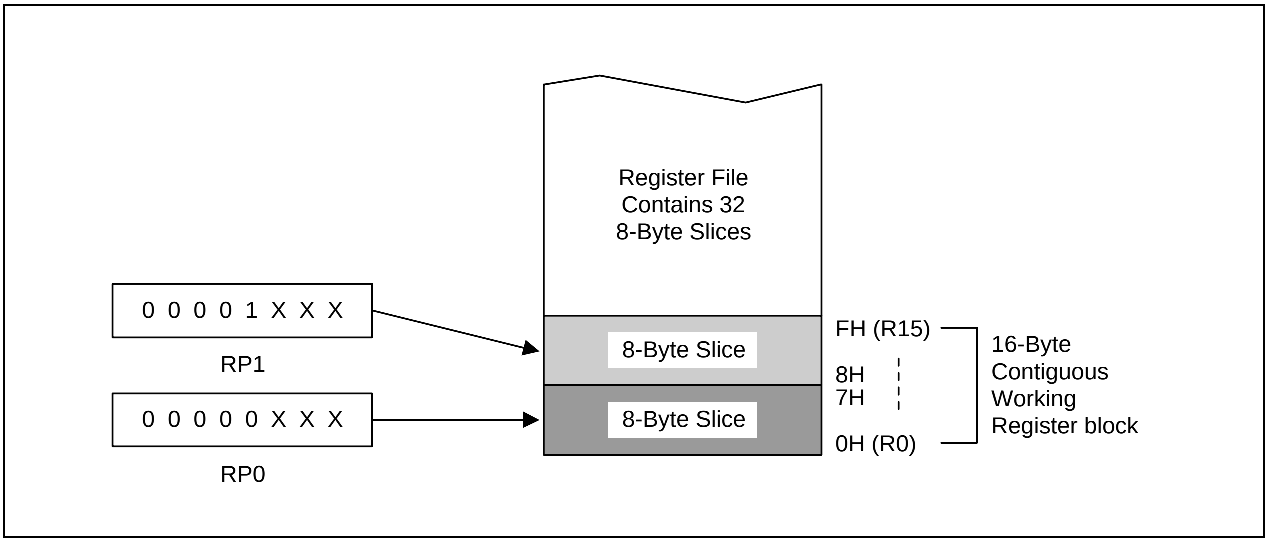


Figure 2-6. Contiguous 16-Byte Working Register Block

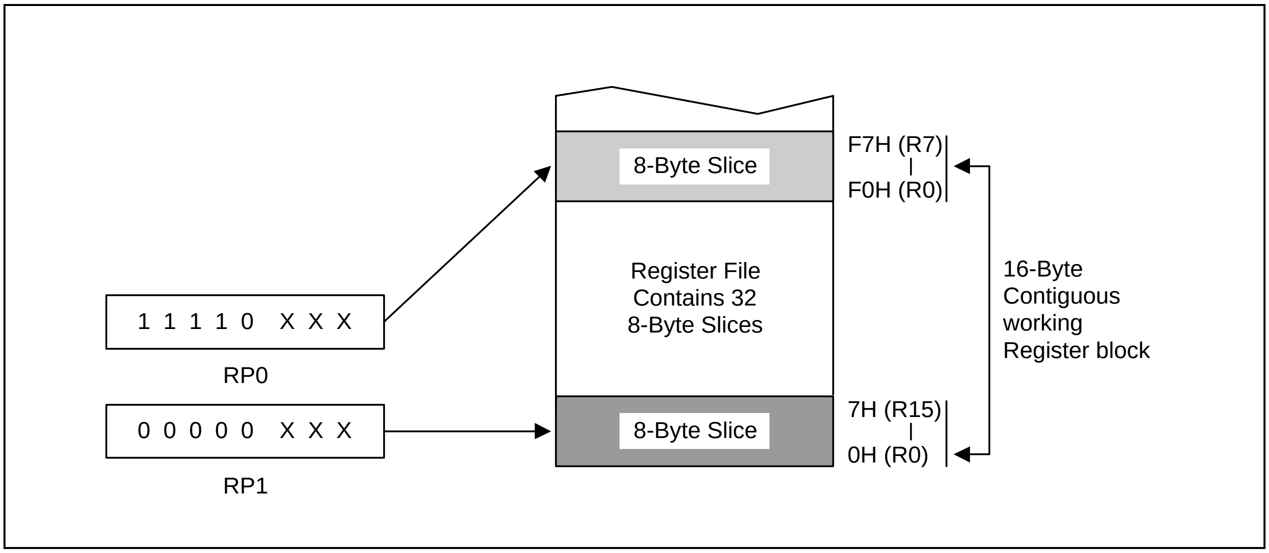


Figure 2-7. Non-Contiguous 16-Byte Working Register Block

☞ PROGRAMMING TIP — Using the RPs to Calculate the Sum of a Series of Registers

Calculate the sum of registers 80H–85H using the register pointer. The register addresses from 80H through 85H contain the values 10H, 11H, 12H, 13H, 14H, and 15H, respectively:

```
SRP0      #80H          ;  RP0  ←  80H
ADD       R0,R1         ;  R0  ←  R0  +  R1
ADC       R0,R2         ;  R0  ←  R0  +  R2 + C
ADC       R0,R3         ;  R0  ←  R0  +  R3 + C
ADC       R0,R4         ;  R0  ←  R0  +  R4 + C
ADC       R0,R5         ;  R0  ←  R0  +  R5 + C
```

The sum of these six registers, 6FH, is located in the register R0 (80H). The instruction string used in this example takes 12 bytes of instruction code and its execution time is 36 cycles. If the register pointer is not used to calculate the sum of these registers, the following instruction sequence would have to be used:

```
ADD       80H,81H       ;  80H  ←  (80H)  +  (81H)
ADC       80H,82H       ;  80H  ←  (80H)  +  (82H)  +  C
ADC       80H,83H       ;  80H  ←  (80H)  +  (83H)  +  C
ADC       80H,84H       ;  80H  ←  (80H)  +  (84H)  +  C
ADC       80H,85H       ;  80H  ←  (80H)  +  (85H)  +  C
```

Now, the sum of the six registers is also located in register 80H. However, this instruction string takes 15 bytes of instruction code rather than 12 bytes, and its execution time is 50 cycles rather than 36 cycles.

REGISTER ADDRESSING

The S3C8-series register architecture provides an efficient method of working register addressing that takes full advantage of shorter instruction formats to reduce execution time.

With Register (R) addressing mode, in which the operand value is the content of a specific register or register pair, you can access any location in the register file except for set 2. With working register addressing, you use a register pointer to specify an 8-byte working register space in the register file and an 8-bit register within that space.

Registers are addressed either as a single 8-bit register or as a paired 16-bit register space. In a 16-bit register pair, the address of the first 8-bit register is always an even number and the address of the next register is always an odd number. The most significant byte of the 16-bit data is always stored in the even-numbered register, and the least significant byte is always stored in the next (+1) odd-numbered register.

Working register addressing differs from Register addressing as it uses a register pointer to identify a specific 8-byte working register space in the internal register file and a specific 8-bit register within that space.

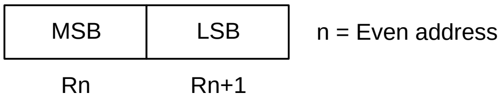
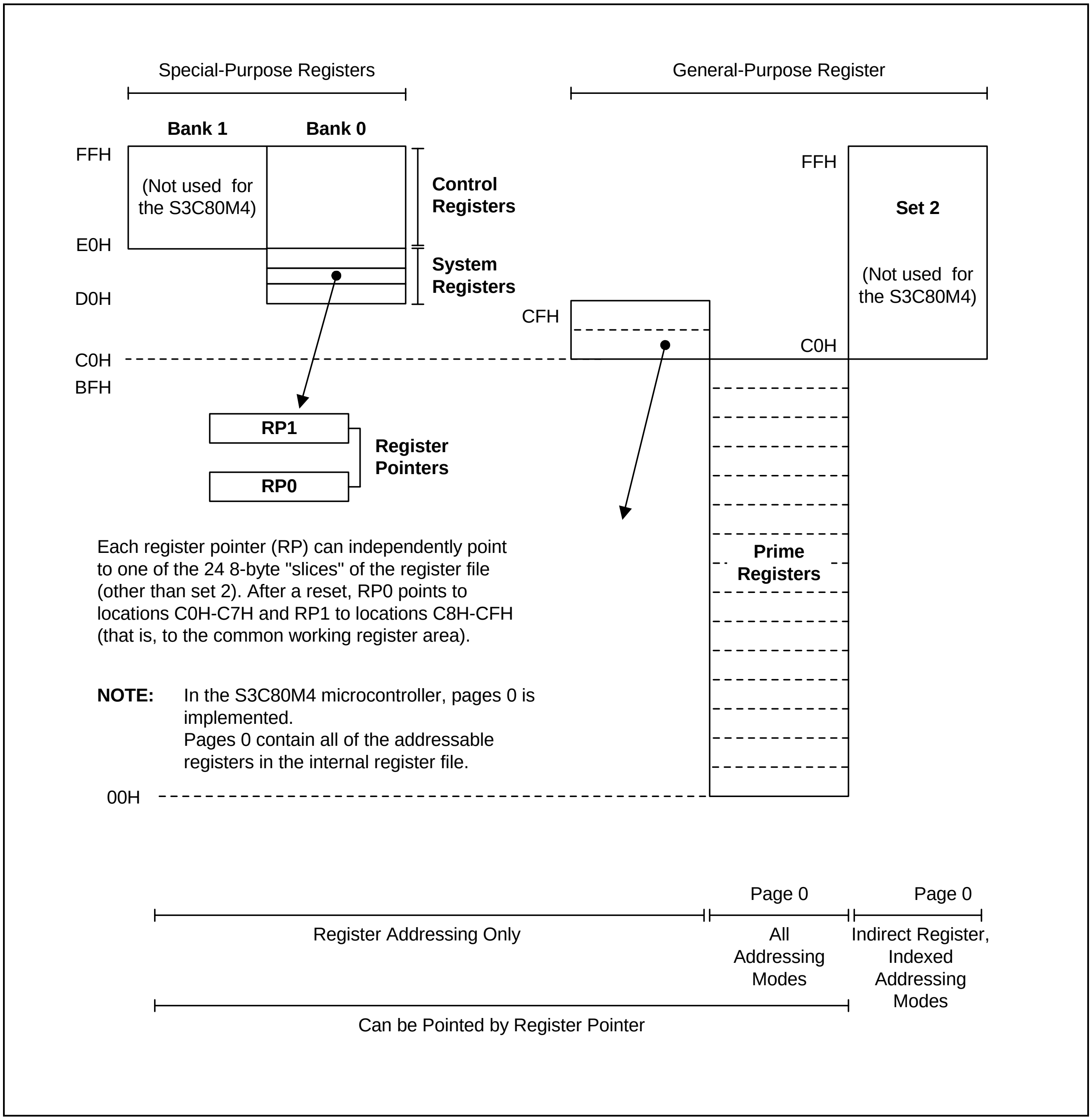


Figure 2-8. 16-Bit Register Pair



Each register pointer (RP) can independently point to one of the 24 8-byte "slices" of the register file (other than set 2). After a reset, RP0 points to locations C0H-C7H and RP1 to locations C8H-CFH (that is, to the common working register area).

NOTE: In the S3C80M4 microcontroller, pages 0 is implemented. Pages 0 contain all of the addressable registers in the internal register file.

Figure 2-9. Register File Addressing

COMMON WORKING REGISTER AREA (C0H–CFH)

After a reset, register pointers RP0 and RP1 automatically select two 8-byte register slices in set 1, locations C0H–CFH, as the active 16-byte working register block:

RP0 → C0H–C7H

RP1 → C8H–CFH

This 16-byte address range is called *common area*. That is, locations in this area can be used as working registers by operations that address any location on any page in the register file. Typically, these working registers serve as temporary buffers for data operations between different pages.

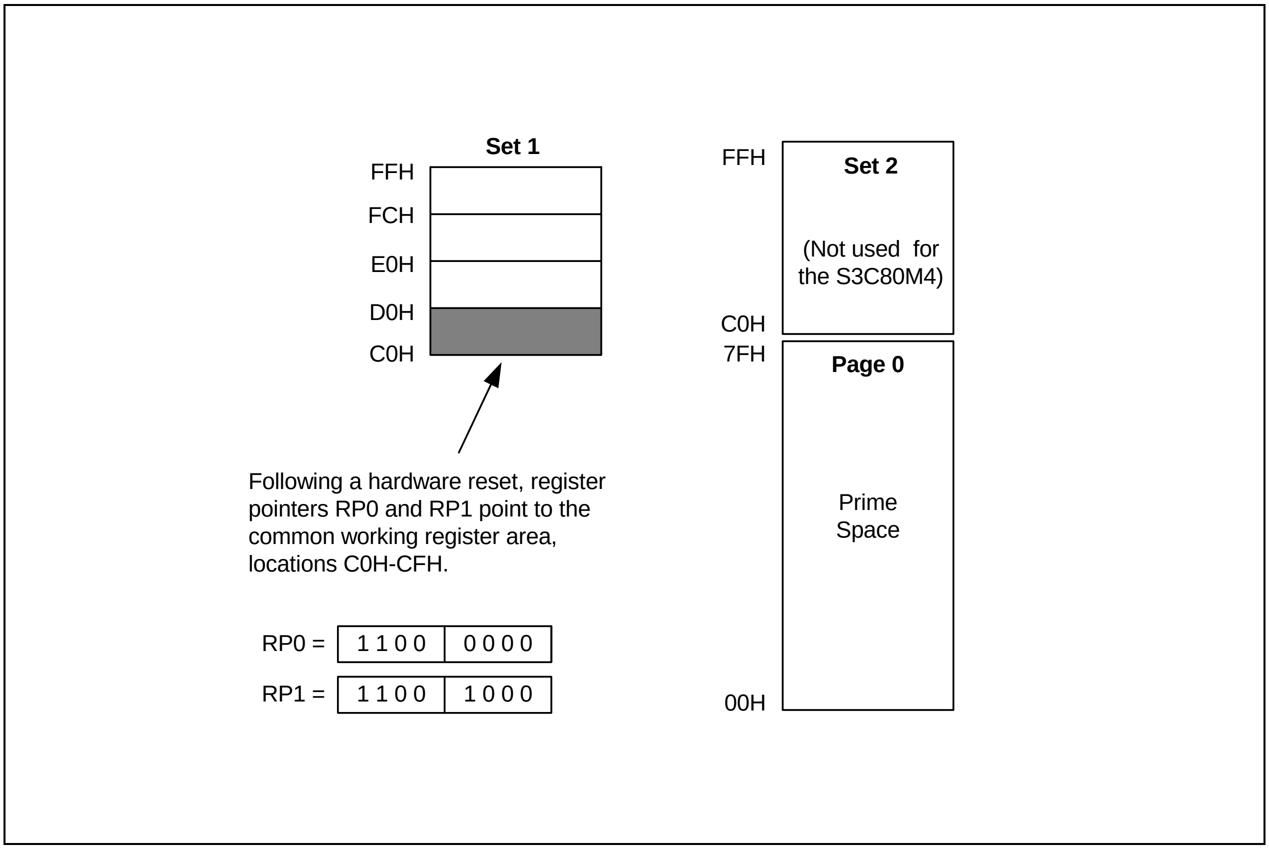

Set 1
FFH
FCH
E0H
D0H
C0H
FFH
Set 2
(Not used for the S3C80M4)
C0H
7FH
Page 0
Prime Space
00H
Following a hardware reset, register pointers RP0 and RP1 point to the common working register area, locations C0H-CFH.
RP0 = 1 1 0 0 | 0 0 0 0
RP1 = 1 1 0 0 | 1 0 0 0

Figure 2-10. Common Working Register Area

☞ PROGRAMMING TIP — Addressing the Common Working Register Area

As the following examples show, you should access working registers in the common area, locations C0H–CFH, using working register addressing mode only.

Examples

```
1. LD       0C2H,40H                 ;   Invalid addressing mode!
   Use working register addressing instead:
   SRP      #0C0H
   LD       R2,40H                   ;   R2 (C2H) →   the value in location 40H

2. ADD      0C3H,#45H                ;   Invalid addressing mode!
   Use working register addressing instead:
   SRP      #0C0H
   ADD      R3,#45H                  ;   R3 (C3H) →   R3 + 45H
```

4-BIT WORKING REGISTER ADDRESSING

Each register pointer defines a movable 8-byte slice of working register space. The address information stored in a register pointer serves as an addressing "window" that makes it possible for instructions to access working registers very efficiently using short 4-bit addresses. When an instruction addresses a location in the selected working register area, the address bits are concatenated in the following way to form a complete 8-bit address:

— The high-order bit of the 4-bit address selects one of the register pointers ("0" selects RP0, "1" selects RP1).

— The five high-order bits in the register pointer select an 8-byte slice of the register space.

— The three low-order bits of the 4-bit address select one of the eight registers in the slice.

As shown in Figure 2-11, the result of this operation is that the five high-order bits from the register pointer are concatenated with the three low-order bits from the instruction address to form the complete address. As long as the address stored in the register pointer remains unchanged, the three bits from the address will always point to an address in the same 8-byte register slice.

Figure 2-12 shows a typical example of 4-bit working register addressing. The high-order bit of the instruction "INC R6" is "0", which selects RP0. The five high-order bits stored in RP0 (01110B) are concatenated with the three low-order bits of the instruction's 4-bit address (110B) to produce the register address 76H (01110110B).

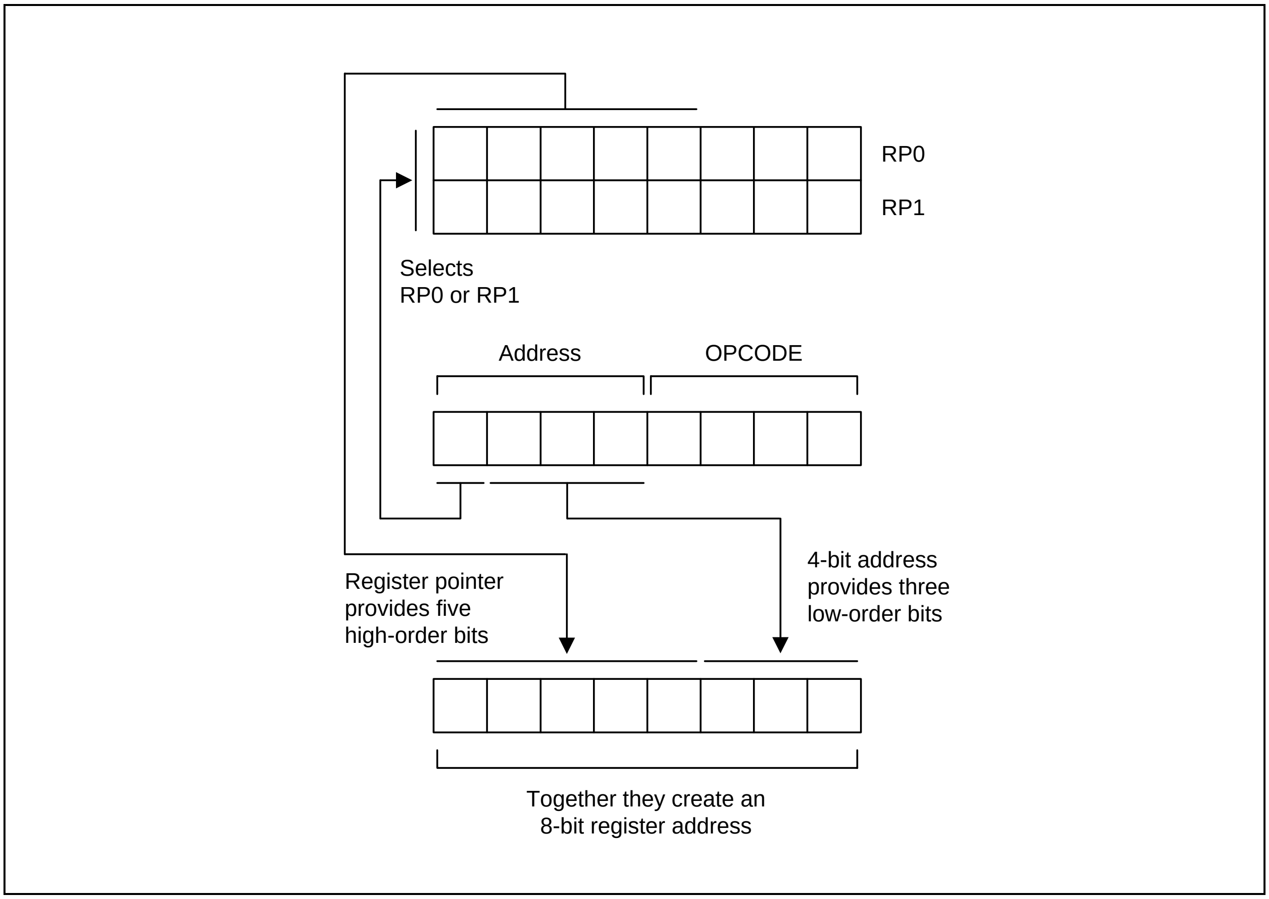


Figure 2-11. 4-Bit Working Register Addressing

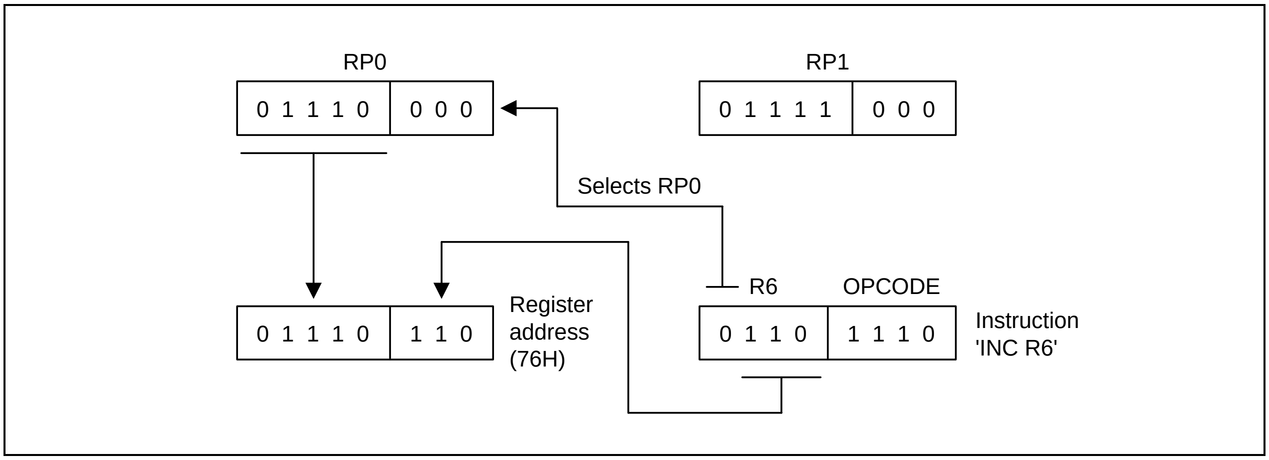


Figure 2-12. 4-Bit Working Register Addressing Example

8-BIT WORKING REGISTER ADDRESSING

You can also use 8-bit working register addressing to access registers in a selected working register area. To initiate 8-bit working register addressing, the upper four bits of the instruction address must contain the value "1100B." This 4-bit value (1100B) indicates that the remaining four bits have the same effect as 4-bit working register addressing.

As shown in Figure 2-13, the lower nibble of the 8-bit address is concatenated in much the same way as for 4-bit addressing: Bit 3 selects either RP0 or RP1, which then supplies the five high-order bits of the final address; the three low-order bits of the complete address are provided by the original instruction.

Figure 2-14 shows an example of 8-bit working register addressing. The four high-order bits of the instruction address (1100B) specify 8-bit working register addressing. Bit 4 ("1") selects RP1 and the five high-order bits in RP1 (10101B) become the five high-order bits of the register address. The three low-order bits of the register address (011) are provided by the three low-order bits of the 8-bit instruction address. The five address bits from RP1 and the three address bits from the instruction are concatenated to form the complete register address, 0ABH (10101011B).

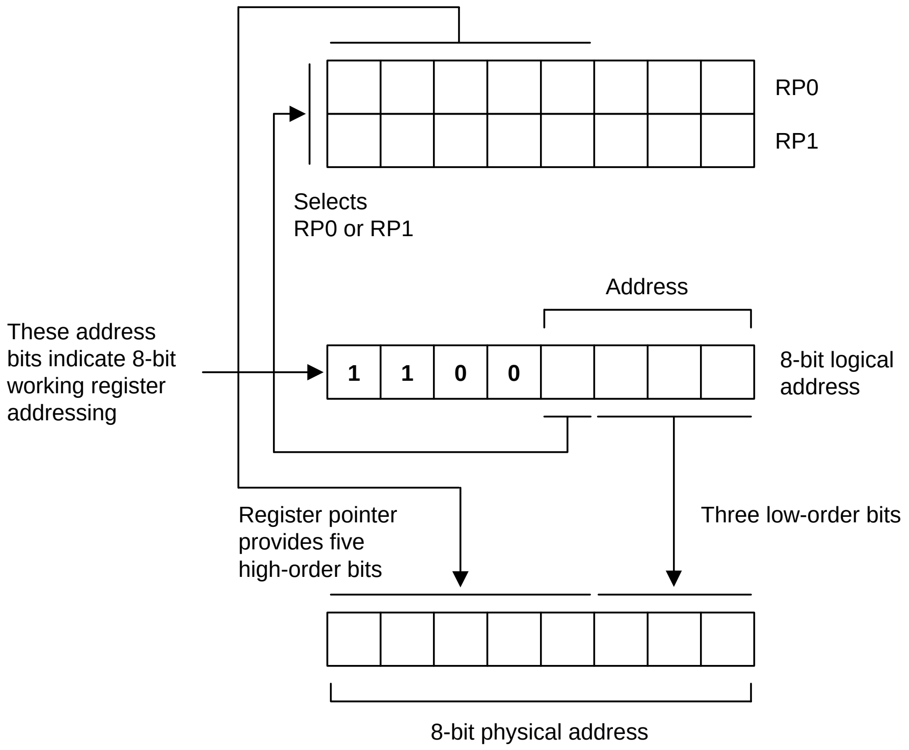


Figure 2-13. 8-Bit Working Register Addressing

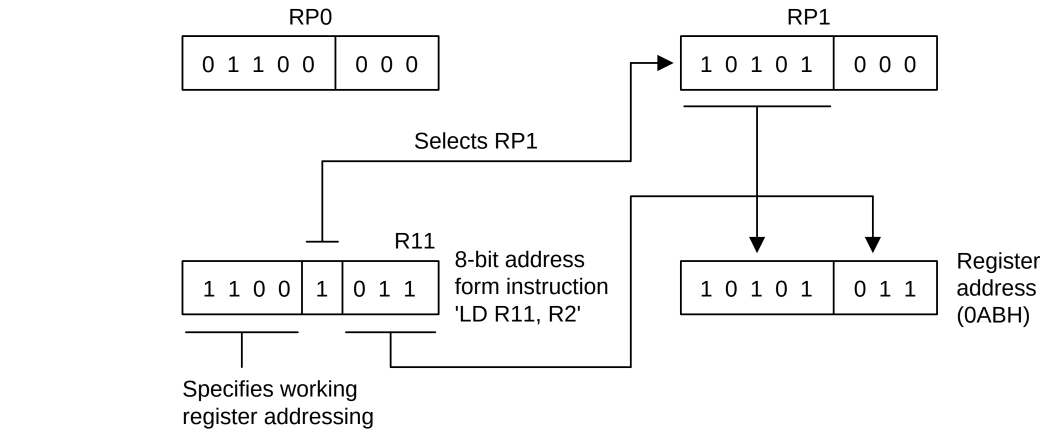


Figure 2-14. 8-Bit Working Register Addressing Example

SYSTEM AND USER STACK

The S3C8-series microcontrollers use the system stack for data storage, subroutine calls and returns. The PUSH and POP instructions are used to control system stack operations. The S3C80M4/F80M4 architecture supports stack operations in the internal register file.

Stack Operations

Return addresses for procedure calls, interrupts, and data are stored on the stack. The contents of the PC are saved to stack by a CALL instruction and restored by the RET instruction. When an interrupt occurs, the contents of the PC and the FLAGS register are pushed to the stack. The IRET instruction then pops these values back to their original locations. The stack address value is always decreased by one before a push operation and increased by one *after* a pop operation. The stack pointer (SP) always points to the stack frame stored on the top of the stack, as shown in Figure 2-15.

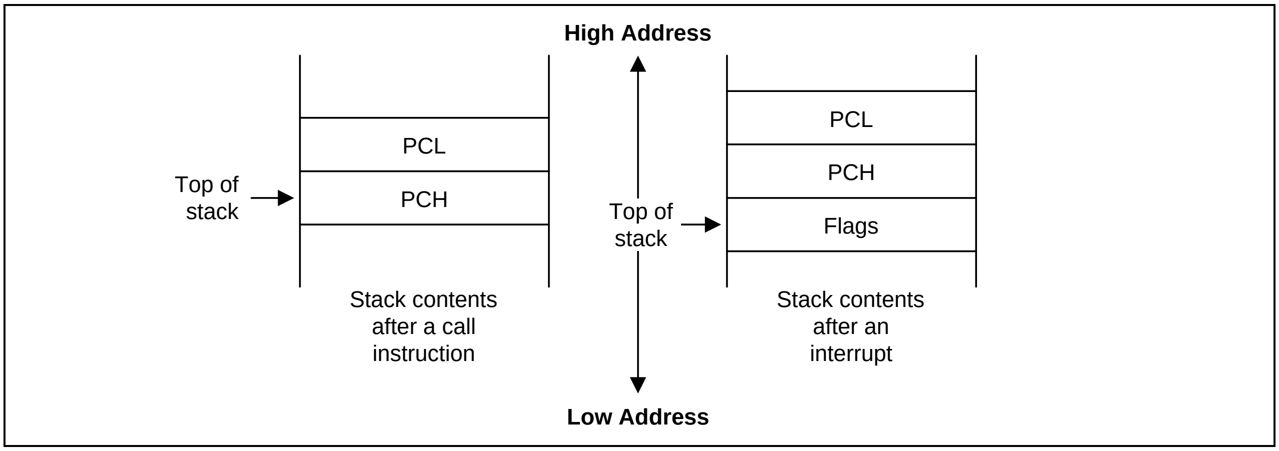

High Address
Top of stack
PCL
PCH
Stack contents after a call instruction
Top of stack
PCL
PCH
Flags
Stack contents after an interrupt
Low Address

Figure 2-15. Stack Operations

User-Defined Stacks

You can freely define stacks in the internal register file as data storage locations. The instructions PUSHUI, PUSHUD, POPUI, and POPUD support user-defined stack operations.

Stack Pointers (SPL, SPH)

Register locations D8H and D9H contain the 16-bit stack pointer (SP) that is used for system stack operations. The most significant byte of the SP address, SP15–SP8, is stored in the SPH register (D8H), and the least significant byte, SP7–SP0, is stored in the SPL register (D9H). After a reset, the SP value is undetermined.

Because only internal memory space is implemented in the S3C84G5, the SPL must be initialized to an 8-bit value in the range 00H–FFH. The SPH register is not needed and can be used as a general-purpose register, if necessary.

When the SPL register contains the only stack pointer value (that is, when it points to a system stack in the register file), you can use the SPH register as a general-purpose data register. However, if an overflow or underflow condition occurs as a result of increasing or decreasing the stack address value in the SPL register during normal stack operations, the value in the SPL register will overflow (or underflow) to the SPH register, overwriting any other data that is currently stored there. To avoid overwriting data in the SPH register, you can initialize the SPL value to "FFH" instead of "00H".

☞ PROGRAMMING TIP — Standard Stack Operations Using PUSH and POP

The following example shows you how to perform stack operations in the internal register file using PUSH and POP instructions:

```
LD          SPL,#0FFH               ;  SPL  ←  FFH
                                    ;  (Normally, the SPL is set to 0FFH by the initialization
                                    ;  routine)
•
•
•
PUSH        PP                      ;  Stack address 0FEH  ←  PP
PUSH        RP0                     ;  Stack address 0FDH  ←  RP0
PUSH        RP1                     ;  Stack address 0FCH  ←  RP1
PUSH        R3                      ;  Stack address 0FBH  ←  R3
•
•
•
POP         R3                      ;  R3  ←  Stack address 0FBH
POP         RP1                     ;  RP1  ←  Stack address 0FCH
POP         RP0                     ;  RP0  ←  Stack address 0FDH
POP         PP                      ;  PP  ←  Stack address 0FEH
```

NOTES

3 ADDRESSING MODES

OVERVIEW

Instructions that are stored in program memory are fetched for execution using the program counter. Instructions indicate the operation to be performed and the data to be operated on. Addressing mode is the method used to determine the location of the data operand. The operands specified in SAM88RC instructions may be condition codes, immediate data, or a location in the register file, program memory, or data memory.

The S3C8-series instruction set supports seven explicit addressing modes. Not all of these addressing modes are available for each instruction. The seven addressing modes and their symbols are:

— Register (R)

— Indirect Register (IR)

— Indexed (X)

— Direct Address (DA)

— Indirect Address (IA)

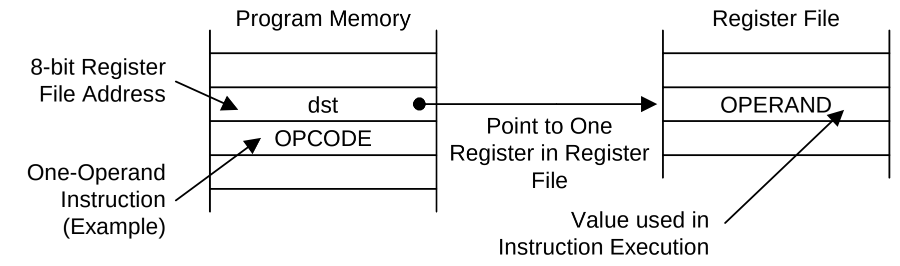
— Relative Address (RA)

— Immediate (IM)

REGISTER ADDRESSING MODE (R)

In Register addressing mode (R), the operand value is the content of a specified register or register pair (see Figure 3-1).

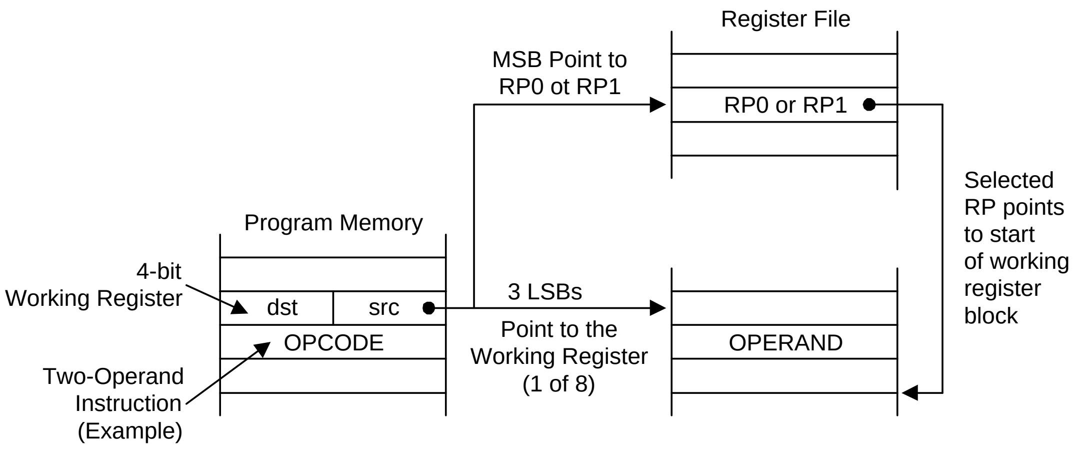
Working register addressing differs from Register addressing in that it uses a register pointer to specify an 8-byte working register space in the register file and an 8-bit register within that space (see Figure 3-2).


Program Memory
Register File
8-bit Register File Address
dst
OPCODE
OPERAND
Point to One Register in Register File
One-Operand Instruction (Example)
Value used in Instruction Execution

Sample Instruction:

```
DEC        CNTR                ;   Where CNTR is the label of an 8-bit register address
```

Figure 3-1. Register Addressing


Register File
MSB Point to RP0 ot RP1
RP0 or RP1
Selected RP points to start of working register block
Program Memory
4-bit Working Register
dst
src
3 LSBs
OPCODE
OPERAND
Point to the Working Register (1 of 8)
Two-Operand Instruction (Example)

Sample Instruction:

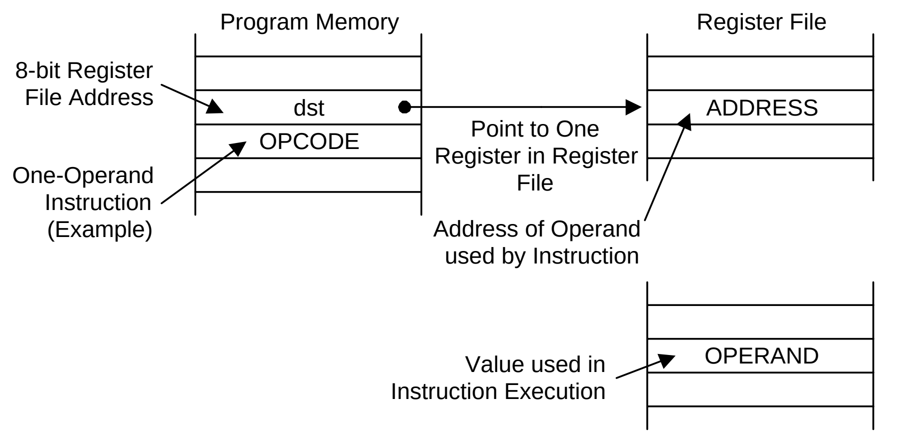
```
ADD        R1, R2              ;   Where R1 and R2 are registers in the currently
                                   selected working register area.
```

Figure 3-2. Working Register Addressing

INDIRECT REGISTER ADDRESSING MODE (IR)

In Indirect Register (IR) addressing mode, the content of the specified register or register pair is the address of the operand. Depending on the instruction used, the actual address may point to a register in the register file, to program memory (ROM), or to an external memory space (see Figures 3-3 through 3-6).

You can use any 8-bit register to indirectly address another register. Any 16-bit register pair can be used to indirectly address another memory location. Please note, however, that you cannot access locations C0H–FFH in set 1 using the Indirect Register addressing mode.


Program Memory
Register File
8-bit Register File Address
dst
OPCODE
One-Operand Instruction (Example)
Point to One Register in Register File
ADDRESS
Address of Operand used by Instruction
OPERAND
Value used in Instruction Execution

Sample Instruction:

```
RL          @SHIFT              ;    Where SHIFT is the label of an 8-bit register address
```

Figure 3-3. Indirect Register Addressing to Register File

INDIRECT REGISTER ADDRESSING MODE (Continued)

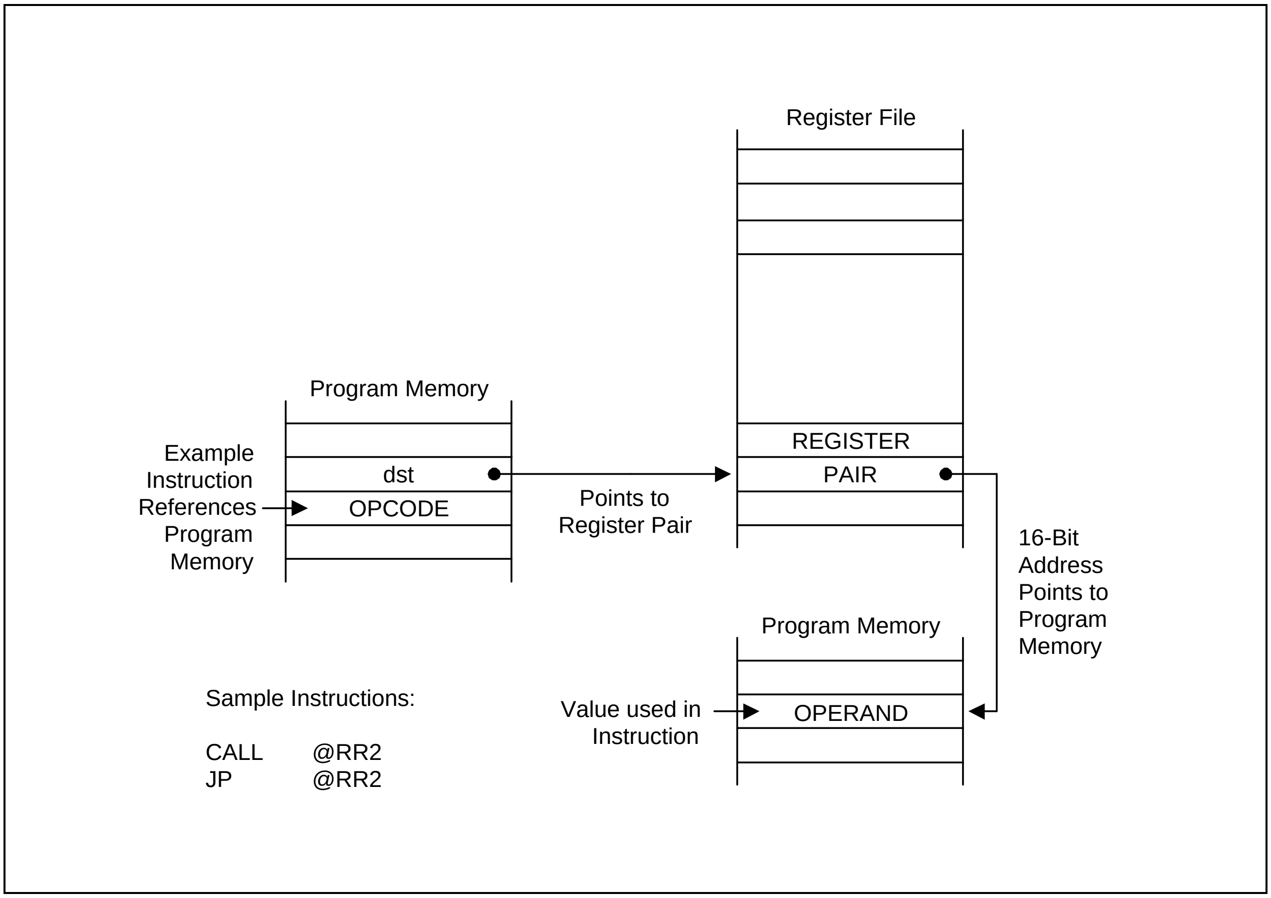


Figure 3-4. Indirect Register Addressing to Program Memory

INDIRECT REGISTER ADDRESSING MODE (Continued)

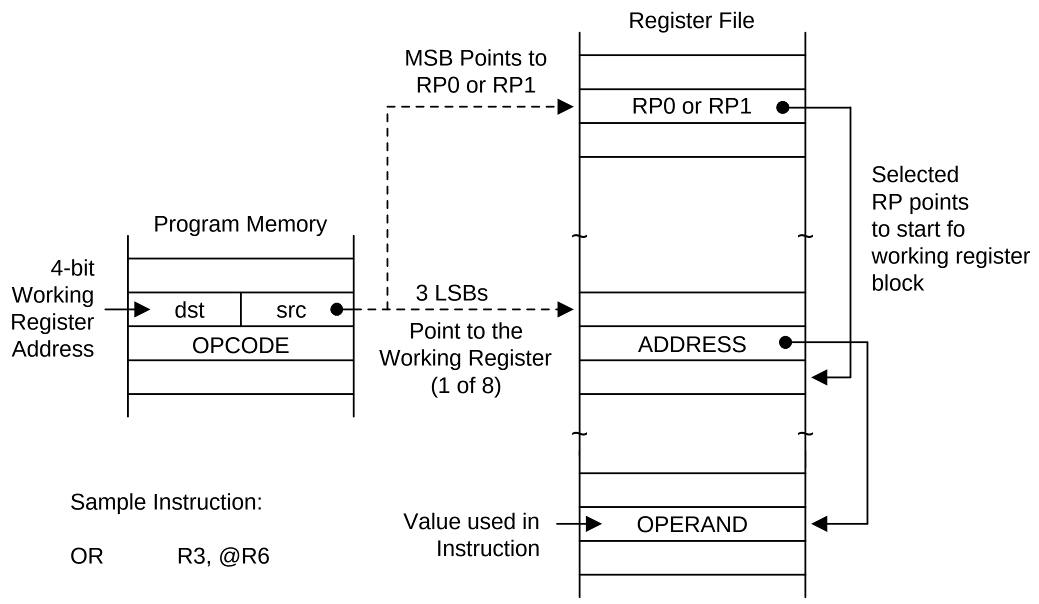
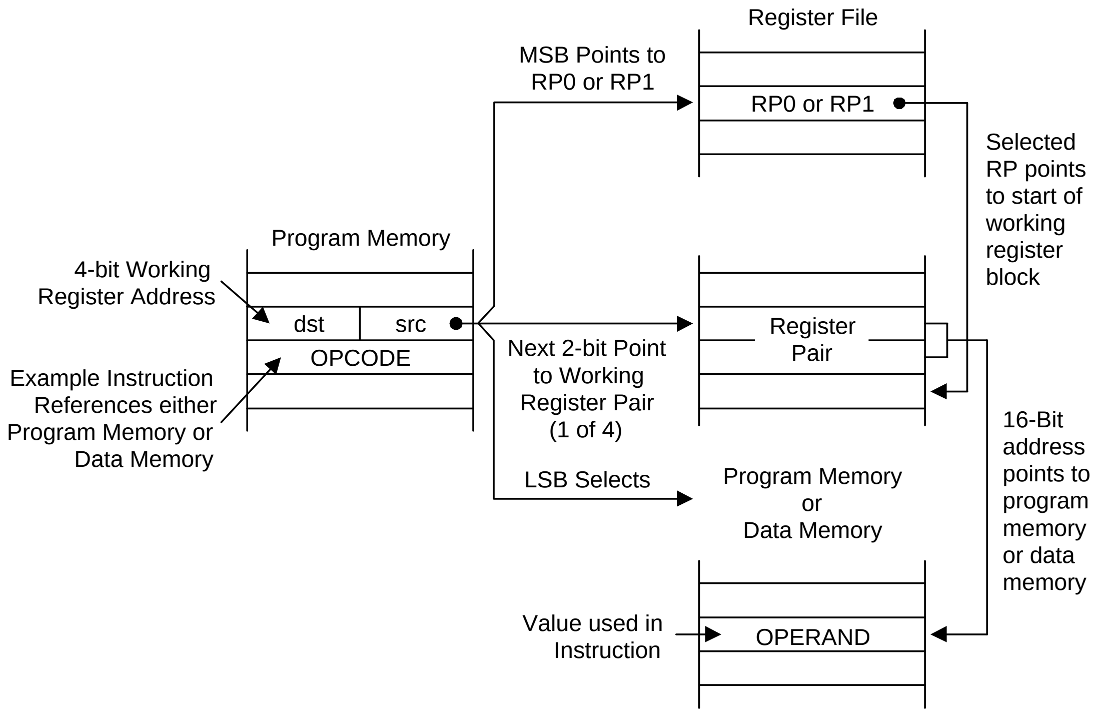


Figure 3-5. Indirect Working Register Addressing to Register File

INDIRECT REGISTER ADDRESSING MODE (Concluded)


Register File
MSB Points to RP0 or RP1
RP0 or RP1
Selected RP points to start of working register block
Program Memory
4-bit Working Register Address
dst
src
OPCODE
Example Instruction References either Program Memory or Data Memory
Next 2-bit Point to Working Register Pair (1 of 4)
Register Pair
16-Bit address points to program memory or data memory
LSB Selects
Program Memory or Data Memory
Value used in Instruction
OPERAND

Sample Instructions:

```
LCD     R5,@RR6      ;  Program memory access
LDE     R3,@RR14     ;  External data memory access
LDE     @RR4, R8     ;  External data memory access
```

Figure 3-6. Indirect Working Register Addressing to Program or Data Memory

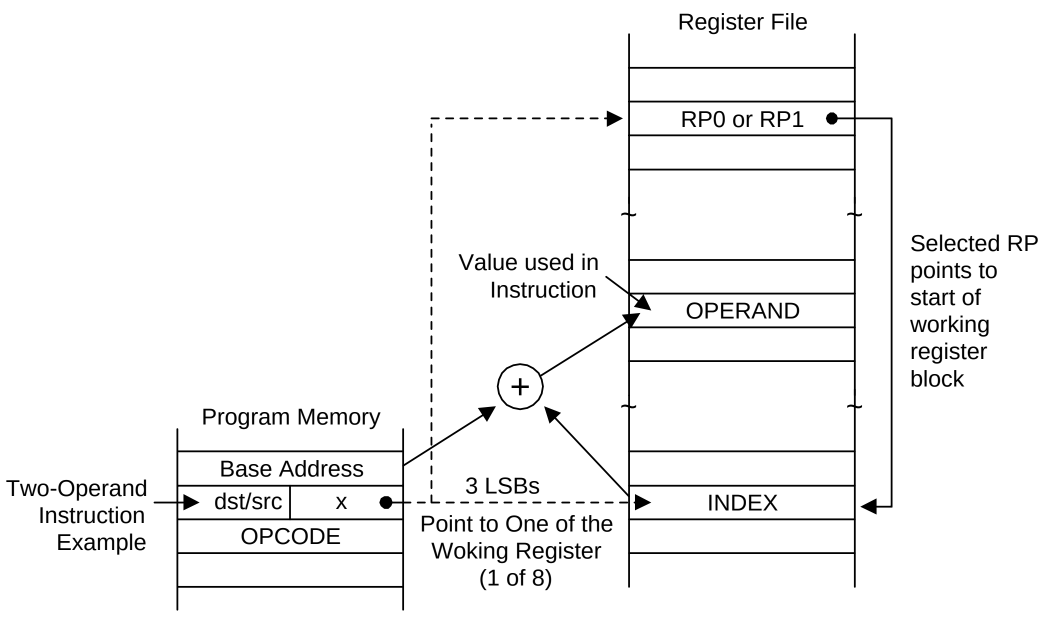
INDEXED ADDRESSING MODE (X)

Indexed (X) addressing mode adds an offset value to a base address during instruction execution in order to calculate the effective operand address (see Figure 3-7). You can use Indexed addressing mode to access locations in the internal register file or in external memory. Please note, however, that you cannot access locations C0H–FFH in set 1 using Indexed addressing mode.

In short offset Indexed addressing mode, the 8-bit displacement is treated as a signed integer in the range –128 to +127. This applies to external memory accesses only (see Figure 3-8.)

For register file addressing, an 8-bit base address provided by the instruction is added to an 8-bit offset contained in a working register. For external memory accesses, the base address is stored in the working register pair designated in the instruction. The 8-bit or 16-bit offset given in the instruction is then added to that base address (see Figure 3-9).

The only instruction that supports Indexed addressing mode for the internal register file is the Load instruction (LD). The LDC and LDE instructions support Indexed addressing mode for internal program memory and for external data memory, when implemented.

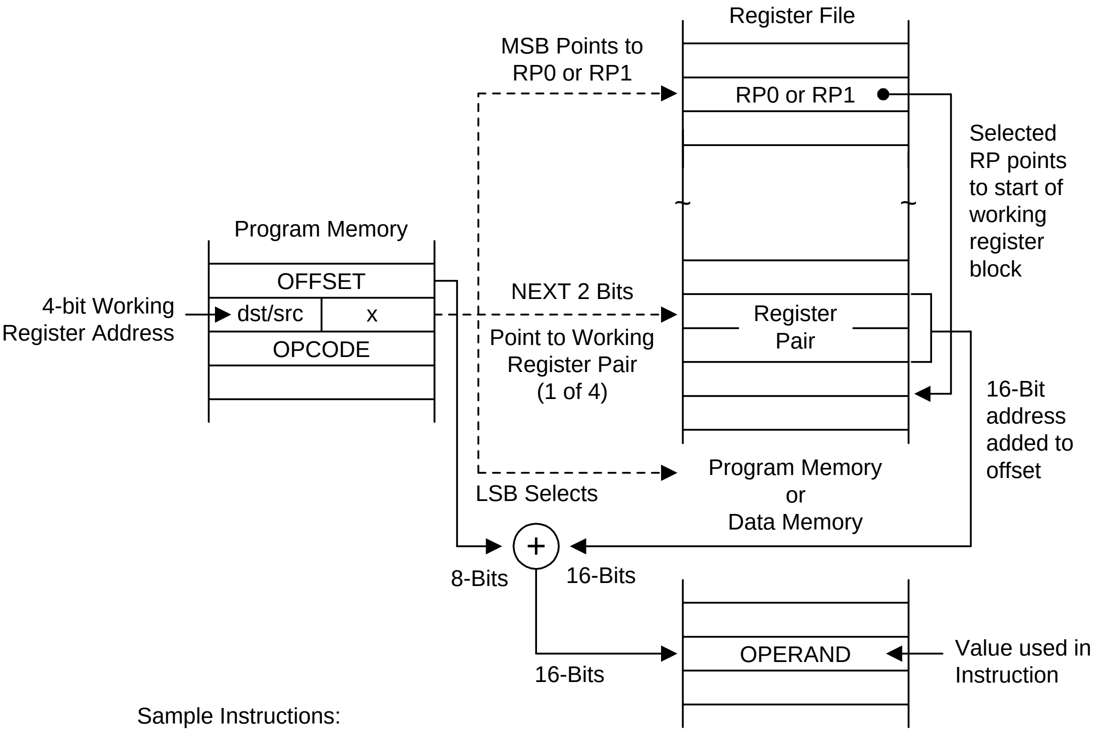

Register File
RP0 or RP1
Value used in Instruction
OPERAND
Selected RP points to start of working register block
+
Program Memory
Base Address
Two-Operand Instruction Example
dst/src
x
3 LSBs
INDEX
OPCODE
Point to One of the Woking Register (1 of 8)

Sample Instruction:

LD R0, #BASE[R1] ; Where BASE is an 8-bit immediate value

Figure 3-7. Indexed Addressing to Register File

INDEXED ADDRESSING MODE (Continued)

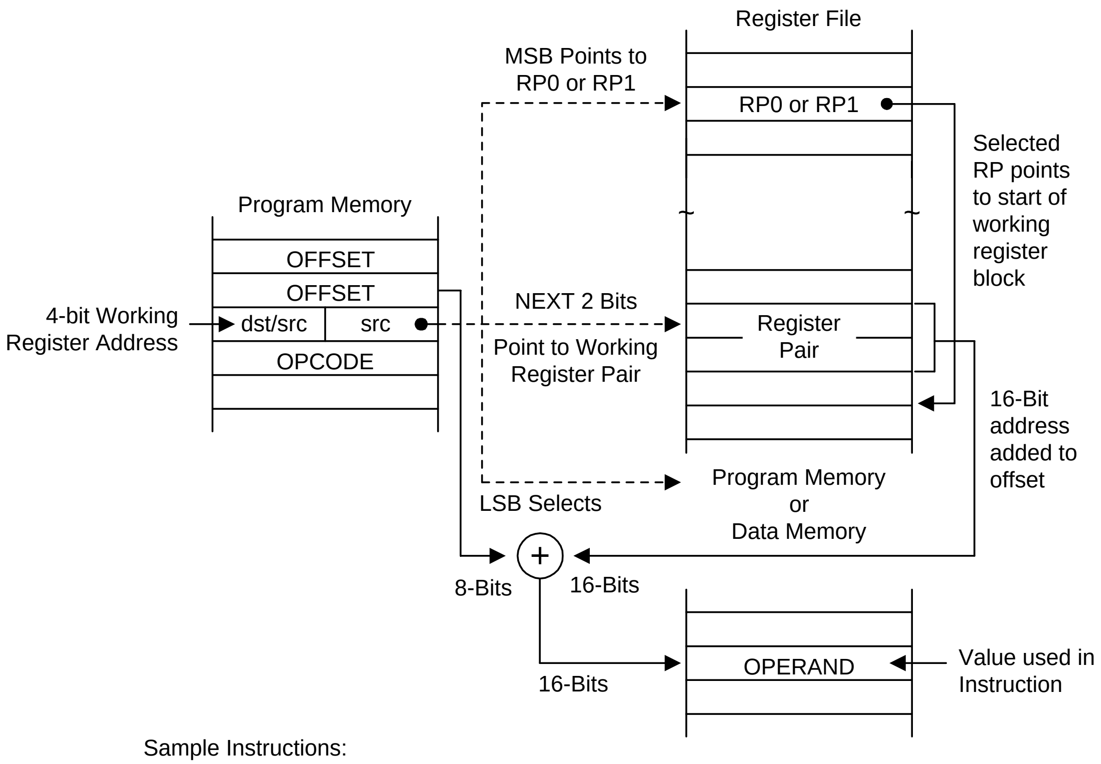


Sample Instructions:

```
LDC     R4, #04H[RR2]     ;  The values in the program address (RR2 + 04H)
                             are loaded into register R4.
LDE     R4,#04H[RR2]      ;  Identical operation to LDC example, except that
                             external program memory is accessed.
```

Figure 3-8. Indexed Addressing to Program or Data Memory with Short Offset

INDEXED ADDRESSING MODE (Concluded)



Sample Instructions:

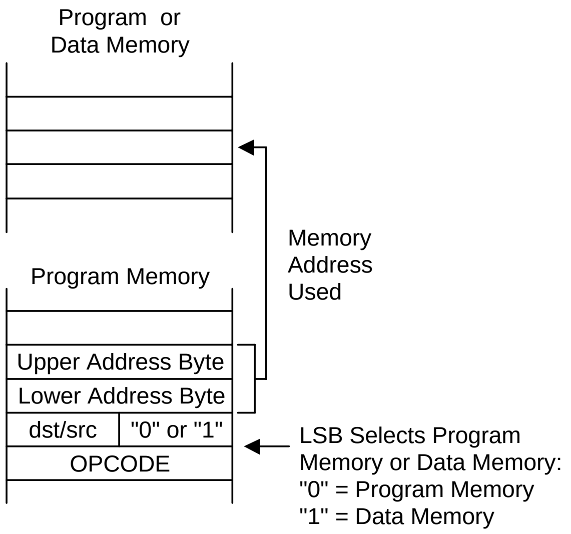
```
LDC     R4, #1000H[RR2]     ;  The values in the program address (RR2 + 1000H)
                               are loaded into register R4.
LDE     R4,#1000H[RR2]      ;  Identical operation to LDC example, except that
                               external program memory is accessed.
```

Figure 3-9. Indexed Addressing to Program or Data Memory

DIRECT ADDRESS MODE (DA)

In Direct Address (DA) mode, the instruction provides the operand's 16-bit memory address. Jump (JP) and Call (CALL) instructions use this addressing mode to specify the 16-bit destination address that is loaded into the PC whenever a JP or CALL instruction is executed.

The LDC and LDE instructions can use Direct Address mode to specify the source or destination address for Load operations to program memory (LDC) or to external data memory (LDE), if implemented.

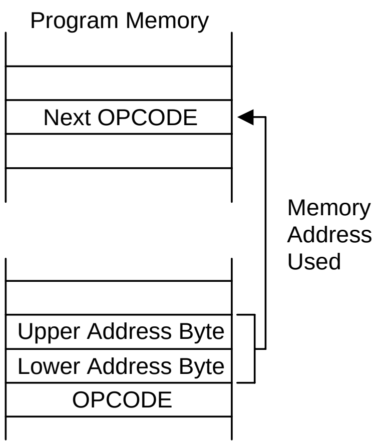


Sample Instructions:

LDC	R5,1234H	;	The values in the program address (1234H) are loaded into register R5.
LDE	R5,1234H	;	Identical operation to LDC example, except that external program memory is accessed.

Figure 3-10. Direct Addressing for Load Instructions

DIRECT ADDRESS MODE (Continued)



Sample Instructions:

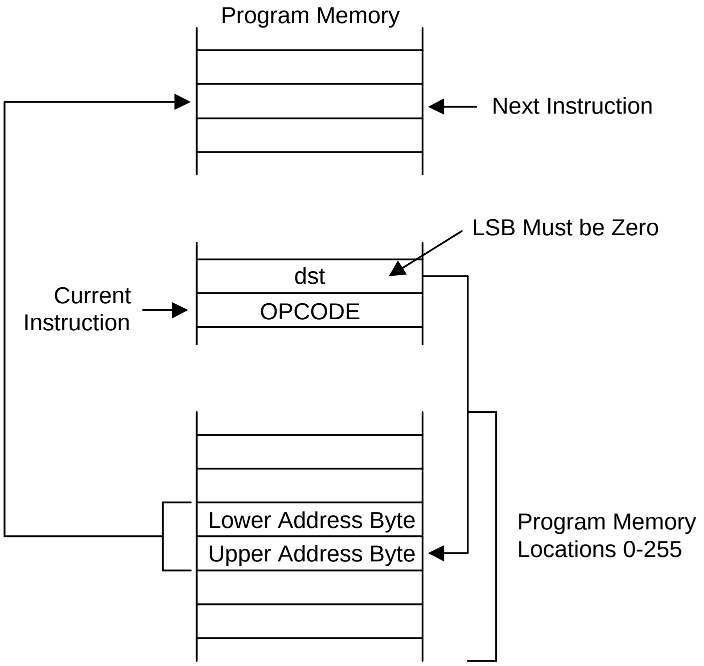
```
JP      C,JOB1        ;   Where JOB1 is a 16-bit immediate address
CALL    DISPLAY       ;   Where DISPLAY is a 16-bit immediate address
```

Figure 3-11. Direct Addressing for Call and Jump Instructions

INDIRECT ADDRESS MODE (IA)

In Indirect Address (IA) mode, the instruction specifies an address located in the lowest 256 bytes of the program memory. The selected pair of memory locations contains the actual address of the next instruction to be executed. Only the CALL instruction can use the Indirect Address mode.

Because the Indirect Address mode assumes that the operand is located in the lowest 256 bytes of program memory, only an 8-bit address is supplied in the instruction; the upper bytes of the destination address are assumed to be all zeros.



Sample Instruction:

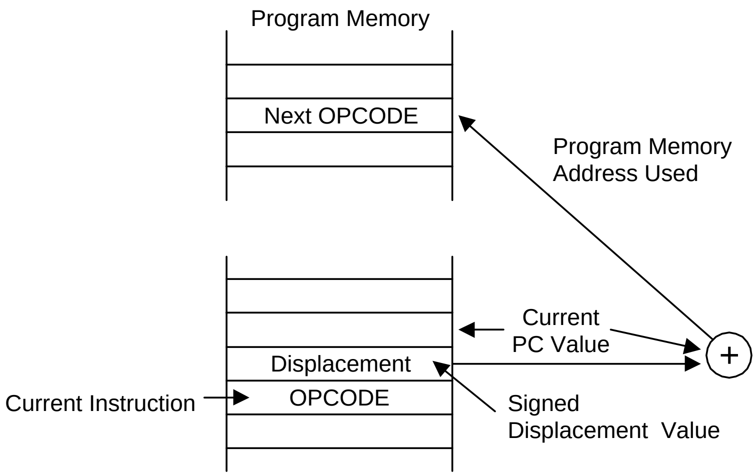
CALL #40H ; The 16-bit value in program memory addresses 40H and 41H is the subroutine start address.

Figure 3-12. Indirect Addressing

RELATIVE ADDRESS MODE (RA)

In Relative Address (RA) mode, a twos-complement signed displacement between – 128 and + 127 is specified in the instruction. The displacement value is then added to the current PC value. The result is the address of the next instruction to be executed. Before this addition occurs, the PC contains the address of the instruction immediately following the current instruction.

Several program control instructions use the Relative Address mode to perform conditional jumps. The instructions that support RA addressing are BTJRF, BTJRT, DJNZ, CPIJE, CPIJNE, and JR.



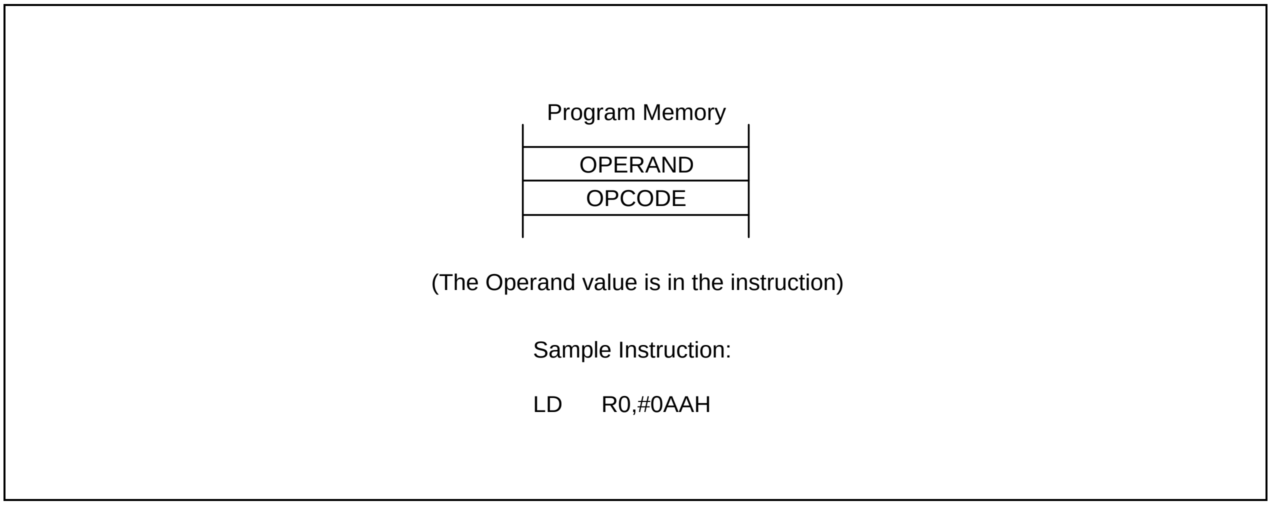
Sample Instructions:

```
JR        ULT,$+OFFSET      ;   Where OFFSET is a value in the range +127 to -128
```

Figure 3-13. Relative Addressing

IMMEDIATE MODE (IM)

In Immediate (IM) addressing mode, the operand value used in the instruction is the value supplied in the operand field itself. The operand may be one byte or one word in length, depending on the instruction used. Immediate addressing mode is useful for loading constant values into registers.



(The Operand value is in the instruction)

Sample Instruction:

```
LD      R0,#0AAH
```

Figure 3-14. Immediate Addressing

4 CONTROL REGISTERS

OVERVIEW

In this chapter, detailed descriptions of the S3C80M4 control registers are presented in an easy-to-read format. You can use this chapter as a quick-reference source when writing application programs. Figure 4-1 illustrates the important features of the standard register description format.

Control register descriptions are arranged in alphabetical order according to register mnemonic. More detailed information about control registers is presented in the context of the specific peripheral hardware descriptions in Part II of this manual.

Data and counter registers are not described in detail in this reference chapter. More information about all of the registers used by a specific peripheral is presented in the corresponding peripheral descriptions in Part II of this manual.

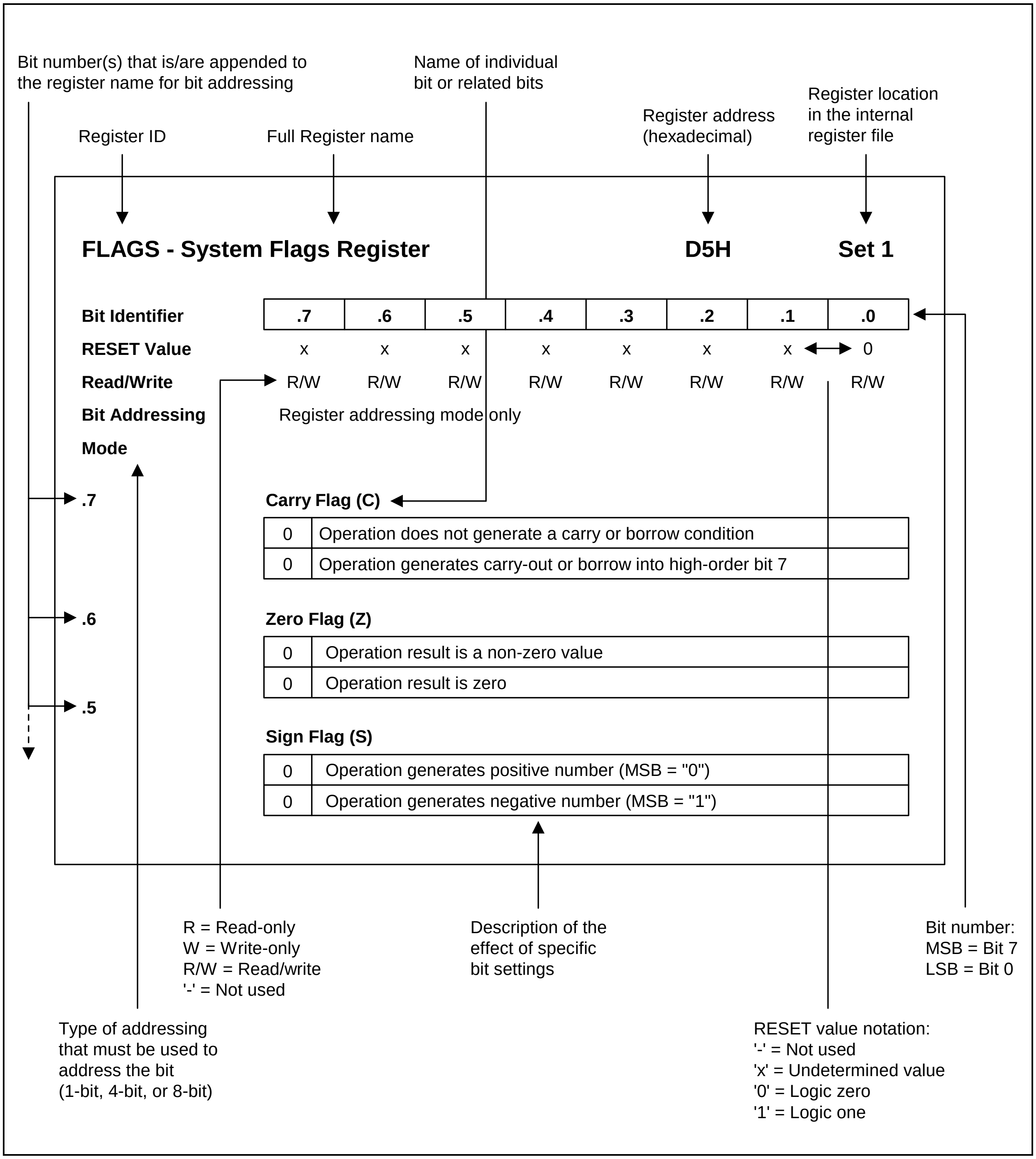
The locations and read/write characteristics of all mapped registers in the S3C80M4 register file are listed in Table 4-1. The hardware reset value for each mapped register is described in Chapter 8, "RESET and Power-Down."

Table 4-1. Set 1 Registers

Register Name	Mnemonic	Decimal	Hex	R/W
Locations D0 – D2H are not mapped.				
Basic Timer Control Register	BTCON	211	D3H	R/W
System Clock Control Register	CLKCON	212	D4H	R/W
System Flags Register	FLAGS	213	D5H	R/W
Register Pointer 0	RP0	214	D6H	R/W
Register Pointer 1	RP1	215	D7H	R/W
Stack Pointer (High Byte)	SPH	216	D8H	R/W
Stack Pointer (Low Byte)	SPL	217	D9H	R/W
Instruction Pointer (High Byte)	IPH	218	DAH	R/W
Instruction Pointer (Low Byte)	IPL	219	DBH	R/W
Interrupt Request Register	IRQ	220	DCH	R
Interrupt Mask Register	IMR	221	DDH	R/W
System Mode Register	SYM	222	DEH	R/W
Register Page Pointer	PP	223	DFH	R/W

Table 4-2. Set 1, Bank 0 Registers

Register Name	Mnemonic	Decimal	Hex	R/W
Port 0 Data Register	P0	224	E0H	R/W
Port 1 Data Register	P1	225	E1H	R/W
Location E2H is not mapped.				
Clock Output Control Register	CLOCON	227	E3H	R/W
Timer 0 Counter Register	T0CNT	228	E4H	R
Timer 0 Data Register	T0DATA	229	E5H	R/W
Timer 0 Control Register	T0CON	230	E6H	R/W
PWM Data Register	PWMDATA	231	E7H	R/W
PWM Control Register	PWMCON	232	E8H	R/W
Locations E9 – EEH are not mapped.				
Port 1 Control Register(High Byte)	P1CONH	240	EFH	R/W
Port 1 Control Register(Low Byte)	P1CONL	241	F0H	R/W
Port 1 Pull-up Resistor Enable Register	P1PUR	242	F1H	R/W
Port 0 Control Register(High Byte)	P0CONH	243	F2H	R/W
Port 0 Control Register(Low Byte)	P0CONL	244	F3H	R/W
Port 0 Interrupt Control Register	P0INT	245	F4H	R/W
Port 0 Interrupt Pending Register	P0PND	246	F5H	R/W
Locations F6 – FAH are not mapped.				
STOP Control Register	STPCON	251	FBH	R/W
Location FCH is not mapped.				
Basic Timer Counter	BTCNT	253	FDH	R
Location FEH is not mapped.				
Interrupt Priority Register	IPR	255	FFH	R/W


Bit number(s) that is/are appended to the register name for bit addressing
Name of individual bit or related bits
Register ID
Full Register name
Register address (hexadecimal)
Register location in the internal register file
R = Read-only
W = Write-only
R/W = Read/write
'-' = Not used
Description of the effect of specific bit settings
Bit number:
MSB = Bit 7
LSB = Bit 0
Type of addressing that must be used to address the bit (1-bit, 4-bit, or 8-bit)
RESET value notation:
'-' = Not used
'x' = Undetermined value
'0' = Logic zero
'1' = Logic one

FLAGS - System Flags Register **D5H** **Set 1**

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
RESET Value	x	x	x	x	x	x	x	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit Addressing Mode	Register addressing mode only							

.7 **Carry Flag (C)**

0	Operation does not generate a carry or borrow condition
0	Operation generates carry-out or borrow into high-order bit 7

.6 **Zero Flag (Z)**

0	Operation result is a non-zero value
0	Operation result is zero

.5 **Sign Flag (S)**

0	Operation generates positive number (MSB = "0")
0	Operation generates negative number (MSB = "1")

Figure 4-1. Register Description Format

BTCON — Basic Timer Control Register D3H Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.4 **Watchdog Timer Function Disable Code (for System Reset)**

1	0	1	0	Disable watchdog timer function
Others				Enable watchdog timer function

.3–.2 **Basic Timer Input Clock Selection Bits**

0	0	fxx/4096
0	1	fxx/1024
1	0	fxx/128
1	1	fxx/16

.1 **Basic Timer Counter Clear Bit [1]**

0	No effect
1	Clear the basic timer counter value

.0 **Clock Frequency Divider Clear Bit for Basic Timer and Timer/Counters [2]**

0	No effect
1	Clear both clock frequency dividers

NOTES:

1. When you write a “1” to BTCON.1, the basic timer counter value is cleared to "00H". Immediately following the write operation, the BTCON.1 value is automatically cleared to “0”.
2. When you write a "1" to BTCON.0, the corresponding frequency divider is cleared to "00H". Immediately following the write operation, the BTCON.0 value is automatically cleared to "0".

CLKCON — System Clock Control Register D4H Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	–	–	0	0	–	–	–
Read/Write	R/W	–	–	R/W	R/W	–	–	–

Addressing Mode Register addressing mode only

.7 **Oscillator IRQ Wake-up Function Bit**

0	Enable IRQ for main wake-up in power down mode
1	Disable IRQ for main wake-up in power down mode

.6–.5 Not used for the S3C80M4

.4–.3 **CPU Clock (System Clock) Selection Bits** (note)

0	0	fxx/16
0	1	fxx/8
1	0	fxx/2
1	1	fxx

.2–.0 Not used for the S3C80M4

NOTE: After a reset, the slowest clock (divided by 16) is selected as the system clock. To select faster clock speeds, load the appropriate values to CLKCON.3 and CLKCON.4.

CLOCON — Clock Output Control Register E3H Set 1, Bank0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	–	–	–	–	–	–	0	0
Read/Write	–	–	–	–	–	–	R/W	R/W

Addressing Mode Register addressing mode only

.7–.2 Not used for the S3C80M4

.1–.0 **Clock Output Frequency Selection Bits**

0	0	fxx/64
0	1	fxx/16
1	0	fxx/8
1	1	fxx/4

FLAGS — System Flags Register D5H Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	x	x	x	x	x	x	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Addressing Mode	Register addressing mode only							

.7 **Carry Flag (C)**

0	Operation does not generate a carry or borrow condition
1	Operation generates a carry-out or borrow into high-order bit 7

.6 **Zero Flag (Z)**

0	Operation result is a non-zero value
1	Operation result is zero

.5 **Sign Flag (S)**

0	Operation generates a positive number (MSB = "0")
1	Operation generates a negative number (MSB = "1")

.4 **Overflow Flag (V)**

0	Operation result is ≤ +127 or ≥ –128
1	Operation result is > +127 or < –128

.3 **Decimal Adjust Flag (D)**

0	Add operation completed
1	Subtraction operation completed

.2 **Half-Carry Flag (H)**

0	No carry-out of bit 3 or no borrow into bit 3 by addition or subtraction
1	Addition generated carry-out of bit 3 or subtraction generated borrow into bit 3

.1 **Fast Interrupt Status Flag (FIS)**

0	Interrupt return (IRET) in progress (when read)
1	Fast interrupt service routine in progress (when read)

.0 **Bank Address Selection Flag (BA)**

0	Bank 0 is selected
1	Bank 1 is selected

IMR — Interrupt Mask Register DDH Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	x	x	x	x	x	x	x	x
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7 **Interrupt Level 7 (IRQ7) Enable Bit; External Interrupts P0.3**

0	Disable (mask)
1	Enable (unmask)

.6 **Interrupt Level 6 (IRQ6) Enable Bit; External Interrupts P0.2**

0	Disable (mask)
1	Enable (unmask)

.5 **Interrupt Level 5 (IRQ5) Enable Bit; External Interrupts P0.1**

0	Disable (mask)
1	Enable (unmask)

.4 **Interrupt Level 4 (IRQ4) Enable Bit; External Interrupts P0.0**

0	Disable (mask)
1	Enable (unmask)

.3 Reserved

.2 **Interrupt Level 2 (IRQ2) Enable Bit; PWM**

0	Disable (mask)
1	Enable (unmask)

.1 Reserved

.0 **Interrupt Level 0 (IRQ0) Enable Bit; Timer 0 Match**

0	Disable (mask)
1	Enable (unmask)

NOTE: When an interrupt level is masked, any interrupt requests that may be issued are not recognized by the CPU.

IPH — Instruction Pointer (High Byte) DAH Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	x	x	x	x	x	x	x	x
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.0	**Instruction Pointer Address (High Byte)**
	The high-byte instruction pointer value is the upper eight bits of the 16-bit instruction pointer address (IP15–IP8). The lower byte of the IP address is located in the IPL register (DBH).

IPL — Instruction Pointer (Low Byte) DBH Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	x	x	x	x	x	x	x	x
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.0	**Instruction Pointer Address (Low Byte)**
	The low-byte instruction pointer value is the lower eight bits of the 16-bit instruction pointer address (IP7–IP0). The upper byte of the IP address is located in the IPH register (DAH).

IPR — Interrupt Priority Register FFH Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	x	x	x	x	x	x	x	x
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7, .4, and .1 **Priority Control Bits for Interrupt Groups A, B, and C**

0	0	0	Group priority undefined
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	Group priority undefined

.6 **Interrupt Subgroup C Priority Control Bit**

0	IRQ6 > IRQ7
1	IRQ7 > IRQ6

.5 **Interrupt Group C Priority Control Bit**

0	IRQ5 > (IRQ6, IRQ7)
1	(IRQ6, IRQ7) > IRQ5

.3 **Interrupt Subgroup B Priority Control Bit**

0	IRQ3 > IRQ4
1	IRQ4 > IRQ3

.2 **Interrupt Group B Priority Control Bit**

0	IRQ2 > (IRQ3, IRQ4)
1	(IRQ3, IRQ4) > IRQ2

.0 **Interrupt Group A Priority Control Bit**

0	IRQ0 > IRQ1
1	IRQ1 > IRQ0

NOTE: Interrupt group A - IRQ0, IRQ1
Interrupt group B -IRQ2, IRQ3, IRQ4
Interrupt group C -IRQ5, IRQ6, IRQ7

IRQ — Interrupt Request Register DCH Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R	R	R	R	R	R	R	R
Addressing Mode	Register addressing mode only							

.7 **Level 7 (IRQ7) Request Pending Bit; External Interrupts P0.3**

0	Not pending
1	Pending

.6 **Level 6 (IRQ6) Request Pending Bit; External Interrupts P0.2**

0	Not pending
1	Pending

.5 **Level 5 (IRQ5) Request Pending Bit; ; External Interrupts P0.1**

0	Not pending
1	Pending

.4 **Level 4 (IRQ4) Request Pending Bit; ; External Interrupts P0.0**

0	Not pending
1	Pending

.3 Reserved

.2 **Level 2 (IRQ2) Request Pending Bit; PWM**

0	Not pending
1	Pending

.1 Reserved

.0 **Level 0 (IRQ0) Request Pending Bit; Timer 0 Match**

0	Not pending
1	Pending

P0CONH — Port 0 Control Register (High Byte) F2H Set 1,Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	1	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.6 **P0.7**

0	0	Schmitt trigger input mode
0	1	Schmitt trigger input mode with pull-up resistor
1	0	Not available
1	1	Output mode, push-pull

.5–.4 **P0.6/PWM**

0	0	Schmitt trigger input mode
0	1	Schmitt trigger input mode with pull-up resistor
1	0	Alternative function (PWM)
1	1	Output mode, push-pull

.3–.2 **P0.5**

0	0	Schmitt trigger input mode
0	1	Schmitt trigger input mode with pull-up resistor
1	0	Not available
1	1	Output mode, push-pull

.1–.0 **P0.4**

0	0	Schmitt trigger input mode
0	1	Schmitt trigger input mode with pull-up resistor
1	0	Not available
1	1	Output mode, push-pull

P0CONL — Port 0 Control Register (Low Byte) F3H Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.6 **P0.3/INT3**

0	0	Schmitt trigger input mode
0	1	Schmitt trigger input mode with pull-up resistor
1	0	Not available
1	1	Output mode, push-pull

.5–.4 **P0.2/INT2**

0	0	Schmitt trigger input mode
0	1	Schmitt trigger input mode with pull-up resistor
1	0	Not available
1	1	Output mode, push-pull

.3–.2 **P0.1/INT1**

0	0	Schmitt trigger input mode
0	1	Schmitt trigger input mode with pull-up resistor
1	0	Not available
1	1	Output mode, push-pull

.1–.0 **P0.0/INT0**

0	0	Schmitt trigger input mode
0	1	Schmitt trigger input mode with pull-up resistor
1	0	Not available
1	1	Output mode, push-pull

P0INT — Port 0 Interrupt Control Register F4H Set 1, Bank 0

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Addressing Mode Register addressing mode only

.7–.6 **P0.3/External interrupt (INT3) Enable Bits**

0	0	Disable interrupt
0	1	Enable interrupt by falling edge
1	0	Enable interrupt by rising edge
1	1	Enable interrupt by both falling and rising edge

.5–.4 **P0.2/External interrupt (INT2) Enable Bits**

0	0	Disable interrupt
0	1	Enable interrupt by falling edge
1	0	Enable interrupt by rising edge
1	1	Enable interrupt by both falling and rising edge

.3–.2 **P0.1/External interrupt (INT1) Enable Bits**

0	0	Disable interrupt
0	1	Enable interrupt by falling edge
1	0	Enable interrupt by rising edge
1	1	Enable interrupt by both falling and rising edge

.1–.0 **P0.0/External interrupt (INT0) Enable Bits**

0	0	Disable interrupt
0	1	Enable interrupt by falling edge
1	0	Enable interrupt by rising edge
1	1	Enable interrupt by both falling and rising edge

P0PND — Port 0 Interrupt Pending Register F5H Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.4 Not used for the S3C80M4

.3 **P0.3/External Interrupt (INT3) Pending Bit**

0	Interrupt request is not pending (When read), Clear pending bit when write 0
1	P0.3/INT3 interrupt request is pending (when read)

.2 **P0.2/External Interrupt (INT2) Pending Bit**

0	Interrupt request is not pending (When read), Clear pending bit when write 0
1	P0.2/INT2 interrupt request is pending (when read)

.1 **P0.1/External Interrupt (INT1) Pending Bit**

0	Interrupt request is not pending (When read), Clear pending bit when write 0
1	P0.1/INT1 interrupt request is pending (when read)

.0 **P0.0/External Interrupt (INT0) Pending Bit**

0	Interrupt request is not pending (When read), Clear pending bit when write 0
1	P0.0/INT0 interrupt request is pending (when read)

P1CONH — Port 1 Control Register (High Byte) EFH Set 1, Bank 0

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
RESET Value	–	–	0	0	0	0	0	0
Read/Write	–	–	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.6

Not used for the S3C80M4

.5–.4 **P1.6/CLKOUT**

0	0	Input mode
0	1	Output mode, N-channel open-drain
1	0	Alternative function (CLKOUT)
1	1	Output mode, push-pull

.3–.2 **P1.5**

0	0	Input mode
0	1	Output mode, N-channel open-drain
1	0	Not available
1	1	Output mode, push-pull

.1–.0 **P1.4**

0	0	input mode
0	1	Output mode, N-channel open-drain
1	0	Not available
1	1	Output mode, push-pull

P1CONL — Port 1 Control Register (Low Byte) F0H Set 1, Bank 0

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.6 **P1.3**

0	0	Schmitt trigger input mode
0	1	Output mode, N-channel open-drain
1	0	Not available
1	1	Output mode, push-pull

.5–.4 **P1.2**

0	0	Schmitt trigger input mode
0	1	Output mode, N-channel open-drain
1	0	Not available
1	1	Output mode, push-pull

.3–.2 **P1.1/T0CLK**

0	0	Schmitt trigger input mode (T0CLK)
0	1	Output mode, N-channel open-drain
1	0	Not available
1	1	Output mode, push-pull

.1–.0 **P1.0/T0OUT**

0	0	Schmitt trigger input mode
0	1	Output mode, N-channel open-drain
1	0	Alternative function (T0OUT)
1	1	Output mode, push-pull

P1PUR — Port 1 Pull-up Resistor Enable Register F1H Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	–	0	0	0	0	0	0	0
Read/Write	–	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7 Not used for the S3C80M4

.6 **P1.6 Pull-up Resistor Enable Bit**

0	Pull-up disable
1	Pull-up enable

.5 **P1.5 Pull-up Resistor Enable Bit**

0	Pull-up disable
1	Pull-up enable

.4 **P1.4 Pull-up Resistor Enable Bit**

0	Pull-up disable
1	Pull-up enable

.3 **P1.3 Pull-up Resistor Enable Bit**

0	Pull-up disable
1	Pull-up enable

.2 **P1.2 Pull-up Resistor Enable Bit**

0	Pull-up disable
1	Pull-up enable

.1 **P1.1 Pull-up Resistor Enable Bit**

0	Pull-up disable
1	Pull-up enable

.0 **P1.0 Pull-up Resistor Enable Bit**

0	Pull-up disable
1	Pull-up enable

NOTE: A pull-up resistor of port 1 is automatically disabled only when the corresponding pin is selected as push-pull output or alternative function.

PP — Register Page Pointer DFH Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.4 **Destination Register Page Selection Bits**

0	0	0	0	Destination: page 0
Others				Not used for the S3C80M4

.3– .0 **Source Register Page Selection Bits**

0	0	0	0	Source: page 0
Others				Not used for the S3C80M4

NOTE: In the S3C80M4 microcontroller, the internal register file is configured as one pages (pages 0).
The page 0 is used for general purpose register file.

PWMCON — Pulse Width Modulation Control Register E8H Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.6 **PWM Input Clock Selection Bits**

0	0	fosc/64
0	1	fosc/8
1	0	fosc/2
1	1	fosc/1

.5 Not used, But you must keep "1"

.4 **PWMDATA Reload Interval Selection Bit**

0	Reload from 8-bit up counter overflow
1	Reload from 6-bit up counter overflow

.3 **PWM Counter Clear Bit**

0	No effect
1	Clear the PWM counter (when write)

.2 **PWM Counter Enable Bit**

0	Counter STOP
1	Counter RUN (Resume countering)

.1 **PWM Overflow Interrupt Enable Bit**

0	Disable interrupt
1	Enable interrupt

.0 **PWM Overflow Interrupt Pending Bit**

0	Interrupt is not pending (when read), Clear pending (when write)
1	Interrupt is pending (when read), No effect (when write)

NOTE: The PWMCON.3 is not automatically cleared to "0". You must pay attention when clear pending bit.

RP0 — Register Pointer 0 D6H Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	1	1	0	0	0	–	–	–
Read/Write	R/W	R/W	R/W	R/W	R/W	–	–	–
Addressing Mode	Register addressing only							

.7–.3 **Register Pointer 0 Address Value**

Register pointer 0 can independently point to one of the 256-byte working register areas in the register file. Using the register pointers RP0 and RP1, you can select two 8-byte register slices at one time as active working register space. After a reset, RP0 points to address C0H in register set 1, selecting the 8-byte working register slice C0H–C7H.

.2–.0 Not used for the S3C80M4

RP1 — Register Pointer 1 D7H Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	1	1	0	0	1	–	–	–
Read/Write	R/W	R/W	R/W	R/W	R/W	–	–	–
Addressing Mode	Register addressing only							

.7– .3 **Register Pointer 1 Address Value**

Register pointer 1 can independently point to one of the 256-byte working register areas in the register file. Using the register pointers RP0 and RP1, you can select two 8-byte register slices at one time as active working register space. After a reset, RP1 points to address C8H in register set 1, selecting the 8-byte working register slice C8H–CFH.

.2– .0 Not used for the S3C80M4

SPH — Stack Pointer (High Byte) D8H Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	x	x	x	x	x	x	x	x
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.0 **Stack Pointer Address (High Byte)**

The high-byte stack pointer value is the upper eight bits of the 16-bit stack pointer address (SP15–SP8). The lower byte of the stack pointer value is located in register SPL (D9H). The SP value is undefined following a reset.

SPL — Stack Pointer (Low Byte) D9H Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	x	x	x	x	x	x	x	x
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.0 **Stack Pointer Address (Low Byte)**

The low-byte stack pointer value is the lower eight bits of the 16-bit stack pointer address (SP7–SP0). The upper byte of the stack pointer value is located in register SPH (D8H). The SP value is undefined following a reset.

STPCON — Stop Control Register FBH Set 1, Bank 0

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.0 **STOP Control Bits**

1 0 1 0 0 1 0 1	Enable stop instruction
Other values	Disable stop instruction

NOTE: Before execute the STOP instruction, You must set this STPCON register as “10100101b”. Otherwise the STOP instruction will not execute as well as reset will be generated.

SYM — System Mode Register DEH Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	–	–	x	x	x	0	0
Read/Write	R/W	–	–	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7 Not used, But you must keep "0"

.6–.5 Not used for the S3C80M4

.4–.2 **Fast Interrupt Level Selection Bits** [(1)]

0	0	0	IRQ0
0	0	1	IRQ1
0	1	0	IRQ2
0	1	1	IRQ3
1	0	0	IRQ4
1	0	1	IRQ5
1	1	0	IRQ6
1	1	1	IRQ7

.1 **Fast Interrupt Enable Bit** [(2)]

0	Disable fast interrupt processing
1	Enable fast interrupt processing

.0 **Global Interrupt Enable Bit** [(3)]

0	Disable all interrupt processing
1	Enable all interrupt processing

NOTES:

1. You can select only one interrupt level at a time for fast interrupt processing.
2. Setting SYM.1 to "1" enables fast interrupt processing for the interrupt level currently selected by SYM.2–SYM.4.
3. Following a reset, you must enable global interrupt processing by executing an EI instruction (not by writing a "1" to SYM.0).

T0CON — Timer 0 Control Register E6H Set 1, Bank 0

Bit Identifier	**.7**	**.6**	**.5**	**.4**	**.3**	**.2**	**.1**	**.0**
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.5 **Timer 0 Input Clock Selection Bits**

0	0	0	fxx/1024
0	0	1	fxx/256
0	1	0	fxx/64
0	1	1	fxx/8
1	0	0	fxx/1
1	0	1	External clock (T0CLK) falling edge
1	1	0	External clock (T0CLK) rising edge
1	1	1	Counter stop

.4 Not used for the S3C80M4

.3 **Timer 0 Counter Clear Bit**

0	No effect
1	Clear the timer 0 counter (when write)

.2 **Timer 0 Counter Enable Bit**

0	Disable counting operation
1	Enable counting operation

.1 **Timer 0 Match Interrupt Enable Bit**

0	Disable interrupt
1	Enable interrupt

.0 **Timer 0 Interrupt Pending Bit**

0	Interrupt request is not pending (when read), Pending bit clear when write 0
1	Interrupt request is pending (when read)

NOTE: The T0CON.3 value is automatically cleared to "0" after being cleared counter.

NOTES

5 INTERRUPT STRUCTURE

OVERVIEW

The S3C8-series interrupt structure has three basic components: levels, vectors, and sources. The SAM8 CPU recognizes up to eight interrupt levels and supports up to 128 interrupt vectors. When a specific interrupt level has more than one vector address, the vector priorities are established in hardware. A vector address can be assigned to one or more sources.

Levels

Interrupt levels are the main unit for interrupt priority assignment and recognition. All peripherals and I/O blocks can issue interrupt requests. In other words, peripheral and I/O operations are interrupt-driven. There are eight possible interrupt levels: IRQ0–IRQ7, also called level 0–level 7. Each interrupt level directly corresponds to an interrupt request number (IRQn). The total number of interrupt levels used in the interrupt structure varies from device to device. The S3C80M4 interrupt structure recognizes eight interrupt levels.

The interrupt level numbers 0 through 7 do not necessarily indicate the relative priority of the levels. They are just identifiers for the interrupt levels that are recognized by the CPU. The relative priority of different interrupt levels is determined by settings in the interrupt priority register, IPR. Interrupt group and subgroup logic controlled by IPR settings lets you define more complex priority relationships between different levels.

Vectors

Each interrupt level can have one or more interrupt vectors, or it may have no vector address assigned at all. The maximum number of vectors that can be supported for a given level is 128 (The actual number of vectors used for S3C8-series devices is always much smaller). If an interrupt level has more than one vector address, the vector priorities are set in hardware. S3C80M4 uses eight vectors.

Sources

A source is any peripheral that generates an interrupt. A source can be an external pin or a counter overflow. Each vector can have several interrupt sources. In the S3C80M4 interrupt structure, there are eight possible interrupt sources.

When a service routine starts, the respective pending bit should be either cleared automatically by hardware or cleared "manually" by program software. The characteristics of the source's pending mechanism determine which method would be used to clear its respective pending bit.

INTERRUPT TYPES

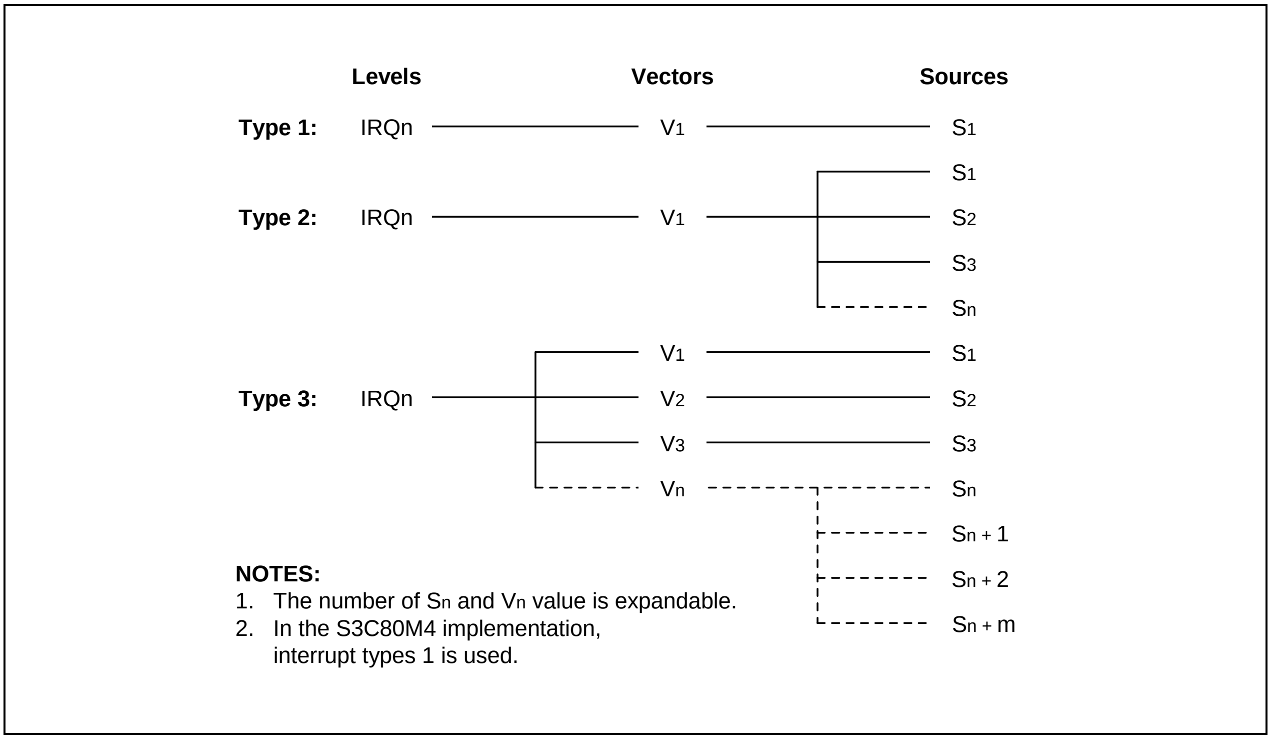
The three components of the S3C8 interrupt structure described before — levels, vectors, and sources — are combined to determine the interrupt structure of an individual device and to make full use of its available interrupt logic. There are three possible combinations of interrupt structure components, called interrupt types 1, 2, and 3. The types differ in the number of vectors and interrupt sources assigned to each level (see Figure 5-1):

Type 1: One level (IRQn) + one vector (V_1) + one source (S_1)

Type 2: One level (IRQn) + one vector (V_1) + multiple sources ($S_1 – S_n$)

Type 3: One level (IRQn) + multiple vectors ($V_1 – V_n$) + multiple sources ($S_1 – S_n$, $S_{n+1} – S_{n+m}$)

In the S3C80M4 microcontroller, two interrupt types are implemented.



NOTES:

1. The number of S_n and V_n value is expandable.
2. In the S3C80M4 implementation, interrupt types 1 is used.

Figure 5-1. S3C8-Series Interrupt Types

S3C80M4 INTERRUPT STRUCTURE

The S3C80M4/F80M4 microcontroller supports nineteen interrupt sources. All nineteen of the interrupt sources have a corresponding interrupt vector address. Eight interrupt levels are recognized by the CPU in this device-specific interrupt structure, as shown in Figure 5-2.

When multiple interrupt levels are active, the interrupt priority register (IPR) determines the order in which contending interrupts are to be serviced. If multiple interrupts occur within the same interrupt level, the interrupt with the lowest vector address is usually processed first (The relative priorities of multiple interrupts within a single level are fixed in hardware).

When the CPU grants an interrupt request, interrupt processing starts. All other interrupts are disabled and the program counter value and status flags are pushed to stack. The starting address of the service routine is fetched from the appropriate vector address (plus the next 8-bit value to concatenate the full 16-bit address) and the service routine is executed.

Levels	Vectors	Sources	Reset/Clear
RESET	100H	Basic Timer Overflow	H/W
IRQ0	EEH	Timer 0 match	S/W
IRQ1	ECH	Reserved	-
IRQ2	EAH	PWM interrupt	S/W
IRQ3	E8H	Reserved	-
IRQ4	E6H	P0.0 External interrupt	S/W
IRQ5	E4H	P0.1 External interrupt	S/W
IRQ6	E2H	P0.2 External interrupt	S/W
IRQ7	E0H	P0.3 External interrupt	S/W

Figure 5-2. S3C80M4/F80M4 Interrupt Structure

INTERRUPT VECTOR ADDRESSES

All interrupt vector addresses for the S3C80M4/F80M4 interrupt structure are stored in the vector address area of the internal 4-Kbyte ROM, 0H–FFFH (see Figure 5-3).

You can allocate unused locations in the vector address area as normal program memory. If you do so, please be careful not to overwrite any of the stored vector addresses (Table 5-1 lists all vector addresses).

The program reset address in the ROM is 0100H.

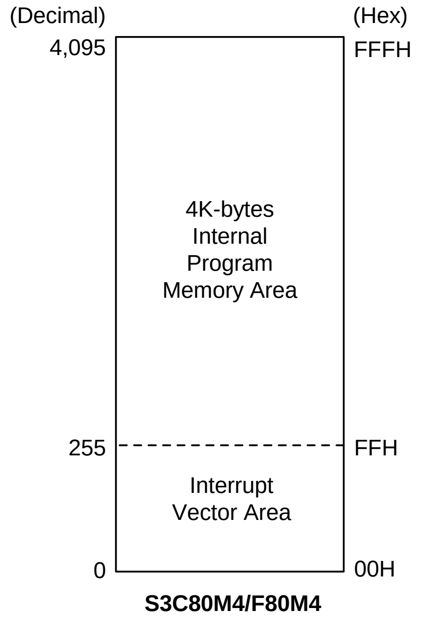


Figure 5-3. ROM Vector Address Area

Table 5-1. Interrupt Vectors

Vector Address		Interrupt Source	Request	Reset/Clear	
Decimal Value	Hex Value		Interrupt Level	H/W	S/W
256	100H	Basic timer overflow	Reset	√	
238	EEH	Timer 0 match	IRQ0		√
236	ECH	Reserved	IRQ1	–	–
234	EAH	PWM interrupt	IRQ2		√
232	E8H	Reserved	IRQ3	–	–
230	E6H	P0.0 external interrupt	IRQ4		√
228	E4H	P0.1 external interrupt	IRQ5		√
226	E2H	P0.2 external interrupt	IRQ6		√
224	E0H	P0.3 external interrupt	IRQ7		√

ENABLE/DISABLE INTERRUPT INSTRUCTIONS (EI, DI)

Executing the Enable Interrupts (EI) instruction globally enables the interrupt structure. All interrupts are then serviced as they occur according to the established priorities.

NOTE

> The system initialization routine executed after a reset must always contain an EI instruction to globally enable the interrupt structure.

During the normal operation, you can execute the DI (Disable Interrupt) instruction at any time to globally disable interrupt processing. The EI and DI instructions change the value of bit 0 in the SYM register.

SYSTEM-LEVEL INTERRUPT CONTROL REGISTERS

In addition to the control registers for specific interrupt sources, four system-level registers control interrupt processing:

- The interrupt mask register, IMR, enables (un-masks) or disables (masks) interrupt levels.
- The interrupt priority register, IPR, controls the relative priorities of interrupt levels.
- The interrupt request register, IRQ, contains interrupt pending flags for each interrupt level (as opposed to each interrupt source).
- The system mode register, SYM, enables or disables global interrupt processing (SYM settings also enable fast interrupts and control the activity of external interface, if implemented).

Table 5-2. Interrupt Control Register Overview

Control Register	ID	R/W	Function Description
Interrupt mask register	IMR	R/W	Bit settings in the IMR register enable or disable interrupt processing for each of the eight interrupt levels: IRQ0–IRQ7.
Interrupt priority register	IPR	R/W	Controls the relative processing priorities of the interrupt levels. The seven levels of S3C80M4/F80M4 are organized into three groups: A, B, and C. Group A is IRQ0 and IRQ1, group B is IRQ2, IRQ3 and IRQ4, and group C is IRQ5, IRQ6, and IRQ7.
Interrupt request register	IRQ	R	This register contains a request pending bit for each interrupt level.
System mode register	SYM	R/W	This register enables/disables fast interrupt processing, dynamic global interrupt processing, and external interface control (An external memory interface is implemented in the S3C80M4/F80M4 microcontroller).

NOTE: Before IMR register is changed to any value, all interrupts must be disable. Using DI instruction is recommended.

INTERRUPT PROCESSING CONTROL POINTS

Interrupt processing can therefore be controlled in two ways: globally or by specific interrupt level and source. The system-level control points in the interrupt structure are:

— Global interrupt enable and disable (by EI and DI instructions or by direct manipulation of SYM.0)

— Interrupt level enable/disable settings (IMR register)

— Interrupt level priority settings (IPR register)

— Interrupt source enable/disable settings in the corresponding peripheral control registers

NOTE

When writing an application program that handles interrupt processing, be sure to include the necessary register file address (register pointer) information.

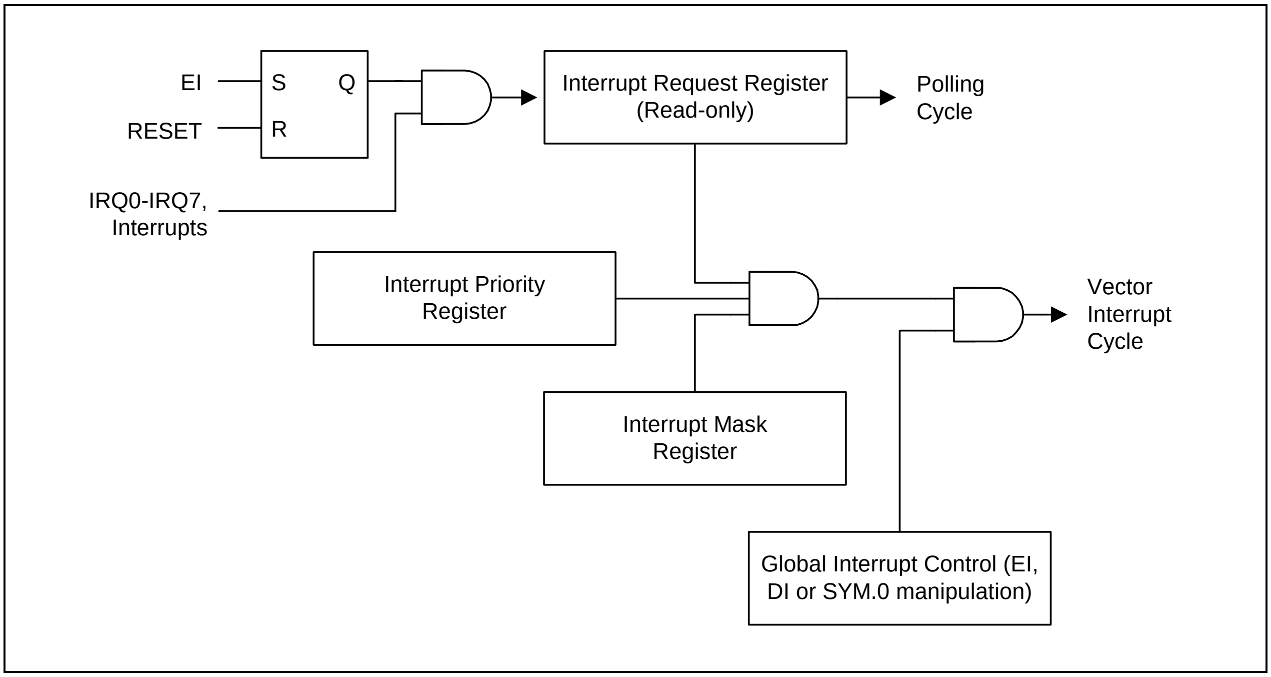


Figure 5-4. Interrupt Function Diagram

PERIPHERAL INTERRUPT CONTROL REGISTERS

For each interrupt source there is one or more corresponding peripheral control registers that let you control the interrupt generated by the related peripheral (see Table 5-3).

Table 5-3. Interrupt Source Control and Data Registers

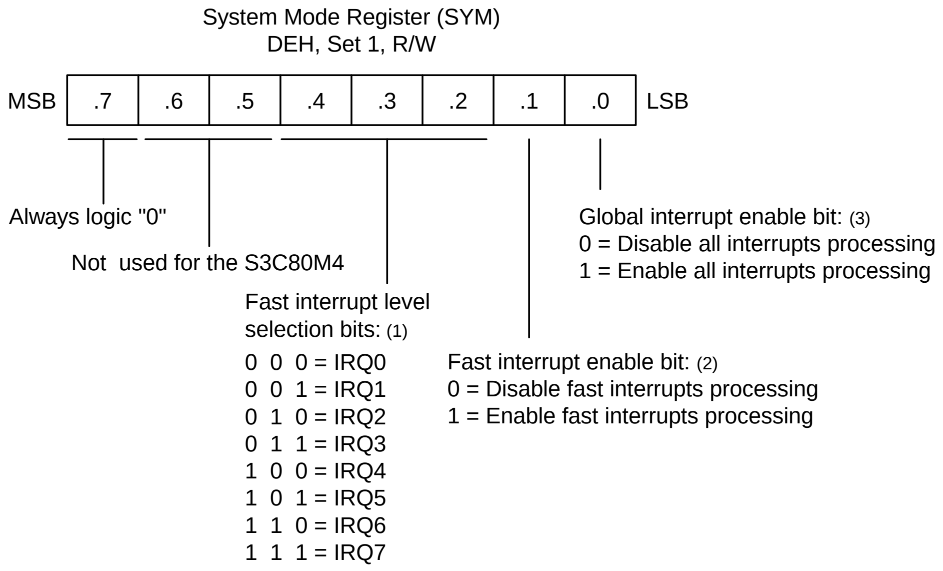
Interrupt Source	Interrupt Level	Register(s)	Location(s) in Set 1
Timer 0 match	IRQ0	T0CON T0DATA T0CNT	E6H, bank 0 E5H, bank 0 E4H, bank 0
Reserved	IRQ1	–	–
PWM interrupt	IRQ2	PWMCON PWMDATA	E8H, bank 0 E7H, bank 0
Reserved	IRQ3	–	–
P0.0 external interrupt	IRQ4	P0CONL P0INT P0PND	F3H, bank 0 F4H, bank 0 F5H, bank 0
P0.1 external interrupt	IRQ5	P0CONL P0INT P0PND	F3H, bank 0 F4H, bank 0 F5H, bank 0
P0.2 external interrupt	IRQ6	P0CONL P0INT P0PND	F3H, bank 0 F4H, bank 0 F5H, bank 0
P0.3 external interrupt	IRQ7	P0CONL P0INT P0PND	F3H, bank 0 F4H, bank 0 F5H, bank 0

SYSTEM MODE REGISTER (SYM)

The system mode register, SYM (set 1, DEH), is used to globally enable and disable interrupt processing and to control fast interrupt processing (see Figure 5-5).

A reset clears SYM.1, and SYM.0 to "0". The 3-bit value for fast interrupt level selection, SYM.4–SYM.2, is undetermined.

The instructions EI and DI enable and disable global interrupt processing, respectively, by modifying the bit 0 value of the SYM register. In order to enable interrupt processing an Enable Interrupt (EI) instruction must be included in the initialization routine, which follows a reset operation. Although you can manipulate SYM.0 directly to enable and disable interrupts during the normal operation, it is recommended to use the EI and DI instructions for this purpose.


System Mode Register (SYM)
DEH, Set 1, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Always logic "0"
Not used for the S3C80M4
Fast interrupt level selection bits: (1)
0 0 0 = IRQ0
0 0 1 = IRQ1
0 1 0 = IRQ2
0 1 1 = IRQ3
1 0 0 = IRQ4
1 0 1 = IRQ5
1 1 0 = IRQ6
1 1 1 = IRQ7
Fast interrupt enable bit: (2)
0 = Disable fast interrupts processing
1 = Enable fast interrupts processing
Global interrupt enable bit: (3)
0 = Disable all interrupts processing
1 = Enable all interrupts processing

NOTES:

1. You can select only one interrupt level at a time for fast interrupt processing.
2. Setting SYM.1 to "1" enables fast interrupt processing for the interrupt processing for the interrupt level currently selected by SYM.2-SYM.4.
3. Following a reset, you must enable global interrupt processing by executing EI instruction (not by writing a "1" to SYM.0)

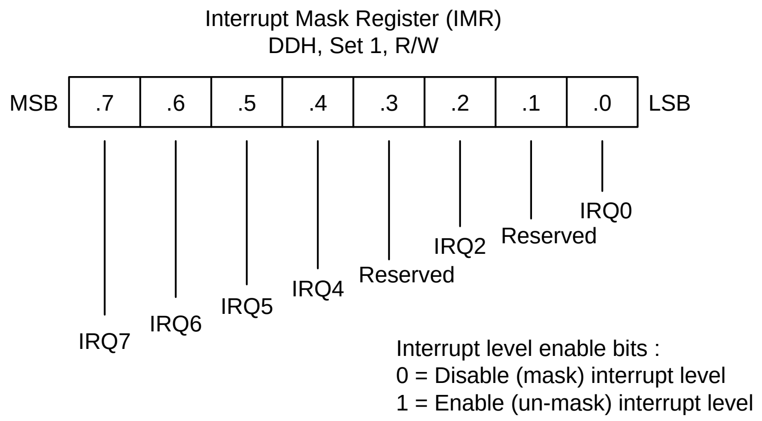
Figure 5-5. System Mode Register (SYM)

INTERRUPT MASK REGISTER (IMR)

The interrupt mask register, IMR (set 1, DDH) is used to enable or disable interrupt processing for individual interrupt levels. After a reset, all IMR bit values are undetermined and must therefore be written to their required settings by the initialization routine.

Each IMR bit corresponds to a specific interrupt level: bit 0 to IRQ0, bit 2 to IRQ2, and so on. When the IMR bit of an interrupt level is cleared to "0", interrupt processing for that level is disabled (masked). When you set a level's IMR bit to "1", interrupt processing for the level is enabled (not masked).

The IMR register is mapped to register location DDH in set 1. Bit values can be read and written by instructions using the Register addressing mode.


Interrupt Mask Register (IMR)
DDH, Set 1, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
IRQ7
IRQ6
IRQ5
IRQ4
Reserved
IRQ2
Reserved
IRQ0
Interrupt level enable bits :
0 = Disable (mask) interrupt level
1 = Enable (un-mask) interrupt level

NOTE: When an interrupt level is masked, any interrupt requests that may be issued are not recognized by the CPU.

Figure 5-6. Interrupt Mask Register (IMR)

INTERRUPT PRIORITY REGISTER (IPR)

The interrupt priority register, IPR (set 1, bank 0, FFH), is used to set the relative priorities of the interrupt levels in the microcontroller's interrupt structure. After a reset, all IPR bit values are undetermined and must therefore be written to their required settings by the initialization routine.

When more than one interrupt sources are active, the source with the highest priority level is serviced first. If two sources belong to the same interrupt level, the source with the lower vector address usually has the priority (This priority is fixed in hardware).

To support programming of the relative interrupt level priorities, they are organized into groups and subgroups by the interrupt logic. Please note that these groups (and subgroups) are used only by IPR logic for the IPR register priority definitions (see Figure 5-7):

Group A	IRQ0, IRQ1
Group B	IRQ2, IRQ3, IRQ4
Group C	IRQ5, IRQ6, IRQ7

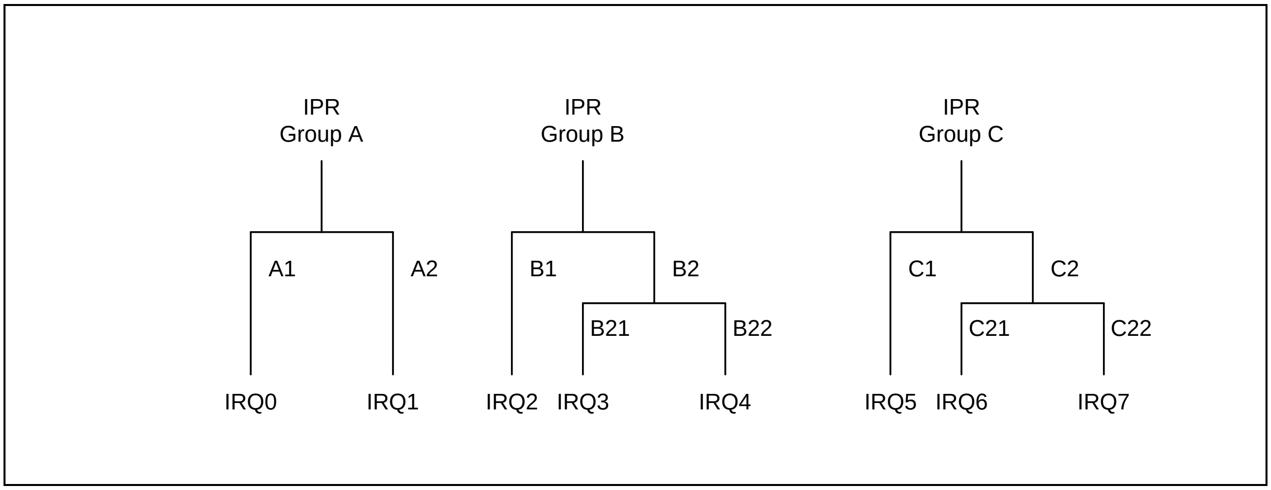


Figure 5-7. Interrupt Request Priority Groups

As you can see in Figure 5-8, IPR.7, IPR.4, and IPR.1 control the relative priority of interrupt groups A, B, and C. For example, the setting "001B" for these bits would select the group relationship B > C > A. The setting "101B" would select the relationship C > B > A.

The functions of the other IPR bit settings are as follows:

— IPR.5 controls the relative priorities of group C interrupts.

— Interrupt group C includes a subgroup that has an additional priority relationship among the interrupt levels 5, 6, and 7. IPR.6 defines the subgroup C relationship. IPR.5 controls the interrupt group C.

— IPR.0 controls the relative priority setting of IRQ0 and IRQ1 interrupts.

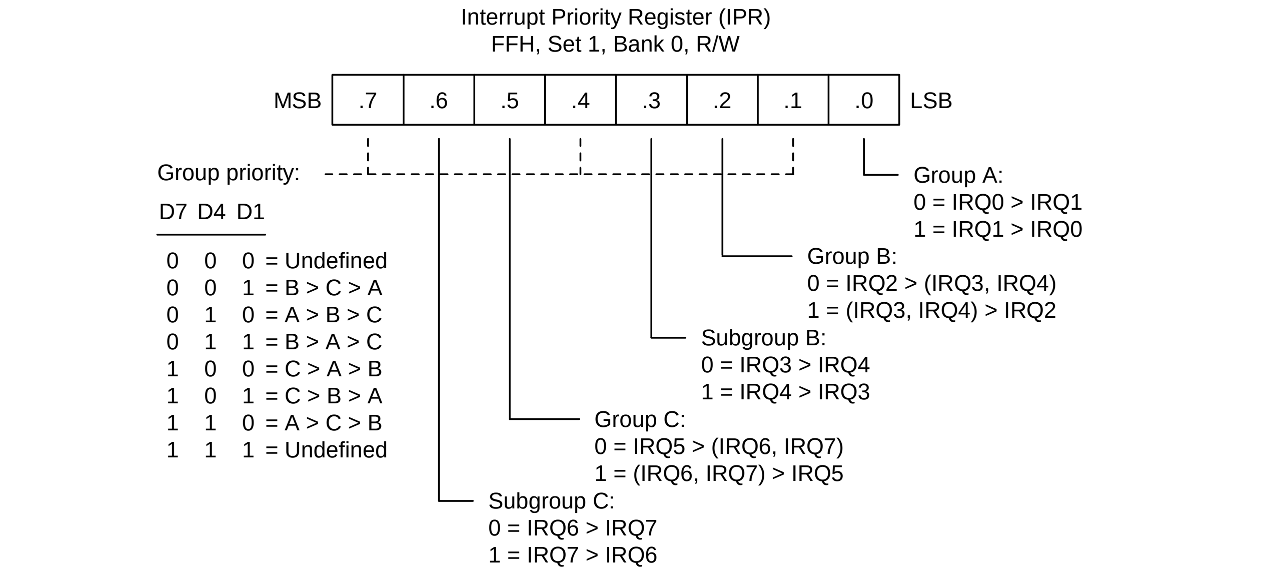


Figure 5-8. Interrupt Priority Register (IPR)

INTERRUPT REQUEST REGISTER (IRQ)

You can poll bit values in the interrupt request register, IRQ (set 1, DCH), to monitor interrupt request status for all levels in the microcontroller’s interrupt structure. Each bit corresponds to the interrupt level of the same number: bit 0 to IRQ0, bit 2 to IRQ2, and so on. A "0" indicates that no interrupt request is currently being issued for that level. A "1" indicates that an interrupt request has been generated for that level.

IRQ bit values are read-only addressable using Register addressing mode. You can read (test) the contents of the IRQ register at any time using bit or byte addressing to determine the current interrupt request status of specific interrupt levels. After a reset, all IRQ status bits are cleared to “0”.

You can poll IRQ register values even if a DI instruction has been executed (that is, if global interrupt processing is disabled). If an interrupt occurs while the interrupt structure is disabled, the CPU will not service it. You can, however, still detect the interrupt request by polling the IRQ register. In this way, you can determine which events occurred while the interrupt structure was globally disabled.

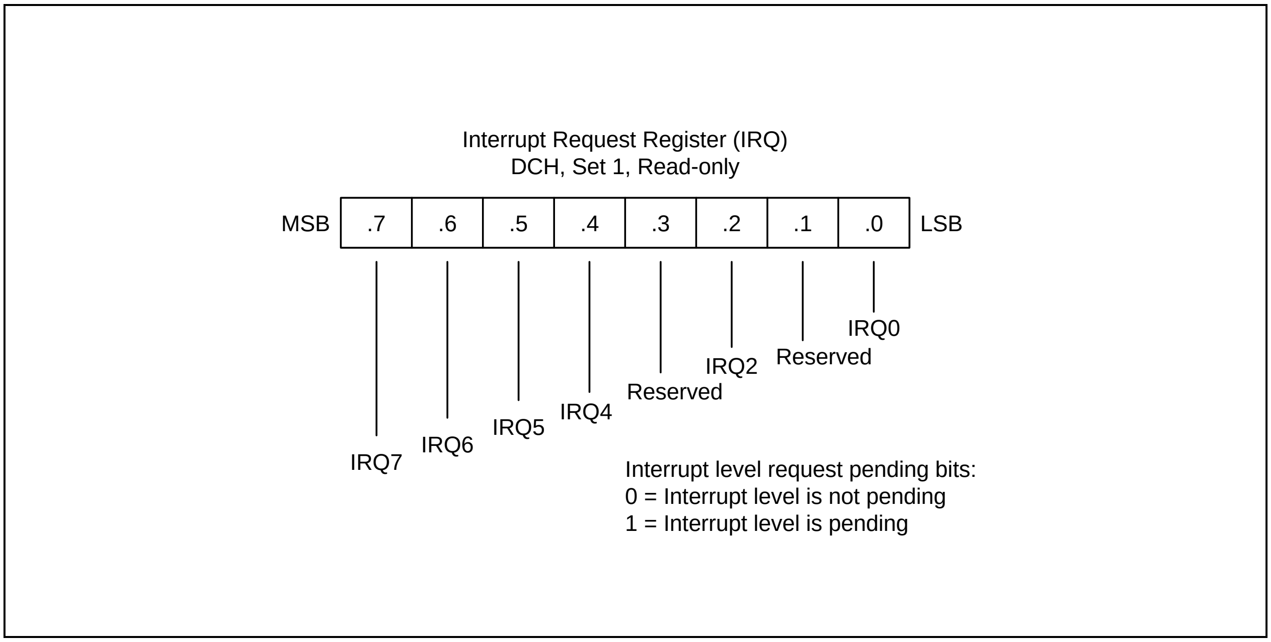

Interrupt Request Register (IRQ)
DCH, Set 1, Read-only
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
IRQ0
Reserved
IRQ2
Reserved
IRQ4
IRQ5
IRQ6
IRQ7
Interrupt level request pending bits:
0 = Interrupt level is not pending
1 = Interrupt level is pending

Figure 5-9. Interrupt Request Register (IRQ)

INTERRUPT PENDING FUNCTION TYPES

Overview

There are two types of interrupt pending bits: one type that is automatically cleared by hardware after the interrupt service routine is acknowledged and executed; the other that must be cleared in the interrupt service routine.

Pending Bits Cleared Automatically by Hardware

For interrupt pending bits that are cleared automatically by hardware, interrupt logic sets the corresponding pending bit to "1" when a request occurs. It then issues an IRQ pulse to inform the CPU that an interrupt is waiting to be serviced. The CPU acknowledges the interrupt source by sending an IACK, executes the service routine, and clears the pending bit to "0". This type of pending bit is not mapped and cannot, therefore, be read or written by application software.

In the S3C80M4 interrupt structure, the timer 0 overflow interrupt (IRQ0) belongs to this category of interrupts in which pending condition is cleared automatically by hardware.

Pending Bits Cleared by the Service Routine

The second type of pending bit is the one that should be cleared by program software. The service routine must clear the appropriate pending bit before a return-from-interrupt subroutine (IRET) occurs. To do this, a "0" must be written to the corresponding pending bit location in the source’s mode or control register.

INTERRUPT SOURCE POLLING SEQUENCE

The interrupt request polling and servicing sequence is as follows:

1. A source generates an interrupt request by setting the interrupt request bit to "1".
2. The CPU polling procedure identifies a pending condition for that source.
3. The CPU checks the source's interrupt level.
4. The CPU generates an interrupt acknowledge signal.
5. Interrupt logic determines the interrupt's vector address.
6. The service routine starts and the source's pending bit is cleared to "0" (by hardware or by software).
7. The CPU continues polling for interrupt requests.

INTERRUPT SERVICE ROUTINES

Before an interrupt request is serviced, the following conditions must be met:

- Interrupt processing must be globally enabled (EI, SYM.0 = "1")
- The interrupt level must be enabled (IMR register)
- The interrupt level must have the highest priority if more than one levels are currently requesting service
- The interrupt must be enabled at the interrupt's source (peripheral control register)

When all the above conditions are met, the interrupt request is acknowledged at the end of the instruction cycle. The CPU then initiates an interrupt machine cycle that completes the following processing sequence:

1. Reset (clear to "0") the interrupt enable bit in the SYM register (SYM.0) to disable all subsequent interrupts.
2. Save the program counter (PC) and status flags to the system stack.
3. Branch to the interrupt vector to fetch the address of the service routine.
4. Pass control to the interrupt service routine.

When the interrupt service routine is completed, the CPU issues an Interrupt Return (IRET). The IRET restores the PC and status flags, setting SYM.0 to "1". It allows the CPU to process the next interrupt request.

GENERATING INTERRUPT VECTOR ADDRESSES

The interrupt vector area in the ROM (00H–FFH) contains the addresses of interrupt service routines that correspond to each level in the interrupt structure. Vectored interrupt processing follows this sequence:

1. Push the program counter's low-byte value to the stack.
2. Push the program counter's high-byte value to the stack.
3. Push the FLAG register values to the stack.
4. Fetch the service routine's high-byte address from the vector location.
5. Fetch the service routine's low-byte address from the vector location.
6. Branch to the service routine specified by the concatenated 16-bit vector address.

NOTE

A 16-bit vector address always begins at an even-numbered ROM address within the range of 00H–FFH.

NESTING OF VECTORED INTERRUPTS

It is possible to nest a higher-priority interrupt request while a lower-priority request is being serviced. To do this, you must follow these steps:

1. Push the current 8-bit interrupt mask register (IMR) value to the stack (PUSH IMR).
2. Load the IMR register with a new mask value that enables only the higher priority interrupt.
3. Execute an EI instruction to enable interrupt processing (a higher priority interrupt will be processed if it occurs).
4. When the lower-priority interrupt service routine ends, restore the IMR to its original value by returning the previous mask value from the stack (POP IMR).
5. Execute an IRET.

Depending on the application, you may be able to simplify the procedure above to some extent.

INSTRUCTION POINTER (IP)

The instruction pointer (IP) is adopted by all the S3C8-series microcontrollers to control the optional high-speed interrupt processing feature called *fast interrupts*. The IP consists of register pair DAH and DBH. The names of IP registers are IPH (high byte, IP15–IP8) and IPL (low byte, IP7–IP0).

FAST INTERRUPT PROCESSING

The feature called *fast interrupt processing* allows an interrupt within a given level to be completed in approximately 6 clock cycles rather than the usual 16 clock cycles. To select a specific interrupt level for fast interrupt processing, you write the appropriate 3-bit value to SYM.4–SYM.2. Then, to enable fast interrupt processing for the selected level, you set SYM.1 to “1”.

FAST INTERRUPT PROCESSING (Continued)

Two other system registers support fast interrupt processing:

— The instruction pointer (IP) contains the starting address of the service routine (and is later used to swap the program counter values), and

— When a fast interrupt occurs, the contents of the FLAGS register is stored in an unmapped, dedicated register called FLAGS' (“FLAGS prime”).

NOTE

For the S3C80M4/F80M4 microcontroller, the service routine for any one of the eight interrupt levels: IRQ0–IRQ7, can be selected for fast interrupt processing.

Procedure for Initiating Fast Interrupts

To initiate fast interrupt processing, follow these steps:

1. Load the start address of the service routine into the instruction pointer (IP).
2. Load the interrupt level number (IRQn) into the fast interrupt selection field (SYM.4–SYM.2)
3. Write a "1" to the fast interrupt enable bit in the SYM register.

Fast Interrupt Service Routine

When an interrupt occurs in the level selected for fast interrupt processing, the following events occur:

1. The contents of the instruction pointer and the PC are swapped.
2. The FLAG register values are written to the FLAGS' (“FLAGS prime”) register.
3. The fast interrupt status bit in the FLAGS register is set.
4. The interrupt is serviced.
5. Assuming that the fast interrupt status bit is set, when the fast interrupt service routine ends, the instruction pointer and PC values are swapped back.
6. The content of FLAGS' (“FLAGS prime”) is copied automatically back to the FLAGS register.
7. The fast interrupt status bit in FLAGS is cleared automatically.

Relationship to Interrupt Pending Bit Types

As described previously, there are two types of interrupt pending bits: One type that is automatically cleared by hardware after the interrupt service routine is acknowledged and executed; the other that must be cleared by the application program's interrupt service routine. You can select fast interrupt processing for interrupts with either type of pending condition clear function — by hardware or by software.

Programming Guidelines

Remember that the only way to enable/disable a fast interrupt is to set/clear the fast interrupt enable bit in the SYM register, SYM.1. Executing an EI or DI instruction globally enables or disables all interrupt processing, including fast interrupts. If you use fast interrupts, remember to load the IP with a new start address when the fast interrupt service routine ends.

NOTES

6 INSTRUCTION SET

OVERVIEW

The SAM8 instruction set is specifically designed to support the large register files that are typical of most SAM8 microcontrollers. There are 78 instructions. The powerful data manipulation capabilities and features of the instruction set include:

- A full complement of 8-bit arithmetic and logic operations, including multiply and divide
- No special I/O instructions (I/O control/data registers are mapped directly into the register file)
- Decimal adjustment included in binary-coded decimal (BCD) operations
- 16-bit (word) data can be incremented and decremented
- Flexible instructions for bit addressing, rotate, and shift operations

DATA TYPES

The SAM8 CPU performs operations on bits, bytes, BCD digits, and two-byte words. Bits in the register file can be set, cleared, complemented, and tested. Bits within a byte are numbered from 7 to 0, where bit 0 is the least significant (right-most) bit.

REGISTER ADDRESSING

To access an individual register, an 8-bit address in the range 0-255 or the 4-bit address of a working register is specified. Paired registers can be used to construct 16-bit data or 16-bit program memory or data memory addresses. For detailed information about register addressing, please refer to Section 2, "Address Spaces."

ADDRESSING MODES

There are seven explicit addressing modes: Register (R), Indirect Register (IR), Indexed (X), Direct (DA), Relative (RA), Immediate (IM), and Indirect (IA). For detailed descriptions of these addressing modes, please refer to Section 3, "Addressing Modes."

Table 6-1. Instruction Group Summary

Mnemonic	Operands	Instruction
Load Instructions		
CLR	dst	Clear
LD	dst,src	Load
LDB	dst,src	Load bit
LDE	dst,src	Load external data memory
LDC	dst,src	Load program memory
LDED	dst,src	Load external data memory and decrement
LDCD	dst,src	Load program memory and decrement
LDEI	dst,src	Load external data memory and increment
LDCI	dst,src	Load program memory and increment
LDEPD	dst,src	Load external data memory with pre-decrement
LDCPD	dst,src	Load program memory with pre-decrement
LDEPI	dst,src	Load external data memory with pre-increment
LDCPI	dst,src	Load program memory with pre-increment
LDW	dst,src	Load word
POP	dst	Pop from stack
POPUD	dst,src	Pop user stack (decrementing)
POPUI	dst,src	Pop user stack (incrementing)
PUSH	src	Push to stack
PUSHUD	dst,src	Push user stack (decrementing)
PUSHUI	dst,src	Push user stack (incrementing)

Table 6-1. Instruction Group Summary (Continued)

Mnemonic	Operands	Instruction
Arithmetic Instructions		
ADC	dst,src	Add with carry
ADD	dst,src	Add
CP	dst,src	Compare
DA	dst	Decimal adjust
DEC	dst	Decrement
DECW	dst	Decrement word
DIV	dst,src	Divide
INC	dst	Increment
INCW	dst	Increment word
MULT	dst,src	Multiply
SBC	dst,src	Subtract with carry
SUB	dst,src	Subtract
Logic Instructions		
AND	dst,src	Logical AND
COM	dst	Complement
OR	dst,src	Logical OR
XOR	dst,src	Logical exclusive OR

Table 6-1. Instruction Group Summary (Continued)

Mnemonic	Operands	Instruction
Program Control Instructions		
BTJRF	dst,src	Bit test and jump relative on false
BTJRT	dst,src	Bit test and jump relative on true
CALL	dst	Call procedure
CPIJE	dst,src	Compare, increment and jump on equal
CPIJNE	dst,src	Compare, increment and jump on non-equal
DJNZ	r,dst	Decrement register and jump on non-zero
ENTER		Enter
EXIT		Exit
IRET		Interrupt return
JP	cc,dst	Jump on condition code
JP	dst	Jump unconditional
JR	cc,dst	Jump relative on condition code
NEXT		Next
RET		Return
WFI		Wait for interrupt
Bit Manipulation Instructions		
BAND	dst,src	Bit AND
BCP	dst,src	Bit compare
BITC	dst	Bit complement
BITR	dst	Bit reset
BITS	dst	Bit set
BOR	dst,src	Bit OR
BXOR	dst,src	Bit XOR
TCM	dst,src	Test complement under mask
TM	dst,src	Test under mask

Table 6-1. Instruction Group Summary (Concluded)

Mnemonic	Operands	Instruction
Rotate and Shift Instructions		
RL	dst	Rotate left
RLC	dst	Rotate left through carry
RR	dst	Rotate right
RRC	dst	Rotate right through carry
SRA	dst	Shift right arithmetic
SWAP	dst	Swap nibbles
CPU Control Instructions		
CCF		Complement carry flag
DI		Disable interrupts
EI		Enable interrupts
IDLE		Enter Idle mode
NOP		No operation
RCF		Reset carry flag
SB0		Set bank 0
SB1		Set bank 1
SCF		Set carry flag
SRP	src	Set register pointers
SRP0	src	Set register pointer 0
SRP1	src	Set register pointer 1
STOP		Enter Stop mode

FLAGS REGISTER (FLAGS)

The flags register FLAGS contains eight bits that describe the current status of CPU operations. Four of these bits, FLAGS.7–FLAGS.4, can be tested and used with conditional jump instructions; two others FLAGS.3 and FLAGS.2 are used for BCD arithmetic.

The FLAGS register also contains a bit to indicate the status of fast interrupt processing (FLAGS.1) and a bank address status bit (FLAGS.0) to indicate whether bank 0 or bank 1 is currently being addressed. FLAGS register can be set or reset by instructions as long as its outcome does not affect the flags, such as, Load instruction.

Logical and Arithmetic instructions such as, AND, OR, XOR, ADD, and SUB can affect the Flags register. For example, the AND instruction updates the Zero, Sign and Overflow flags based on the outcome of the AND instruction. If the AND instruction uses the Flags register as the destination, then simultaneously, two write will occur to the Flags register producing an unpredictable result.

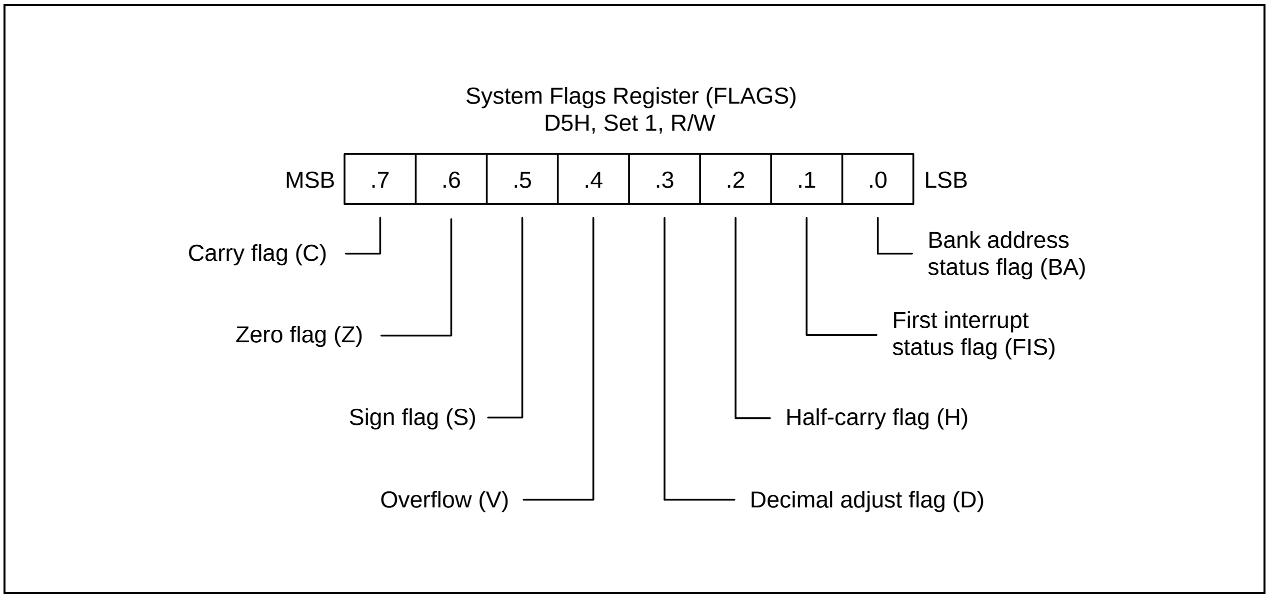


Figure 6-1. System Flags Register (FLAGS)

FLAG DESCRIPTIONS

C Carry Flag (FLAGS.7)

The C flag is set to "1" if the result from an arithmetic operation generates a carry-out from or a borrow to the bit 7 position (MSB). After rotate and shift operations, it contains the last value shifted out of the specified register. Program instructions can set, clear, or complement the carry flag.

Z Zero Flag (FLAGS.6)

For arithmetic and logic operations, the Z flag is set to "1" if the result of the operation is zero. For operations that test register bits, and for shift and rotate operations, the Z flag is set to "1" if the result is logic zero.

S Sign Flag (FLAGS.5)

Following arithmetic, logic, rotate, or shift operations, the sign bit identifies the state of the MSB of the result. A logic zero indicates a positive number and a logic one indicates a negative number.

V Overflow Flag (FLAGS.4)

The V flag is set to "1" when the result of a two's-complement operation is greater than + 127 or less than – 128. It is also cleared to "0" following logic operations.

D Decimal Adjust Flag (FLAGS.3)

The DA bit is used to specify what type of instruction was executed last during BCD operations, so that a subsequent decimal adjust operation can execute correctly. The DA bit is not usually accessed by programmers, and cannot be used as a test condition.

H Half-Carry Flag (FLAGS.2)

The H bit is set to "1" whenever an addition generates a carry-out of bit 3, or when a subtraction borrows out of bit 4. It is used by the Decimal Adjust (DA) instruction to convert the binary result of a previous addition or subtraction into the correct decimal (BCD) result. The H flag is seldom accessed directly by a program.

FIS Fast Interrupt Status Flag (FLAGS.1)

The FIS bit is set during a fast interrupt cycle and reset during the IRET following interrupt servicing. When set, it inhibits all interrupts and causes the fast interrupt return to be executed when the IRET instruction is executed.

BA Bank Address Flag (FLAGS.0)

The BA flag indicates which register bank in the set 1 area of the internal register file is currently selected, bank 0 or bank 1. The BA flag is cleared to "0" (select bank 0) when you execute the SB0 instruction and is set to "1" (select bank 1) when you execute the SB1 instruction.

INSTRUCTION SET NOTATION

Table 6-2. Flag Notation Conventions

Flag	Description
C	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow flag
D	Decimal-adjust flag
H	Half-carry flag
0	Cleared to logic zero
1	Set to logic one
*	Set or cleared according to operation
–	Value is unaffected
x	Value is undefined

Table 6-3. Instruction Set Symbols

Symbol	Description
dst	Destination operand
src	Source operand
@	Indirect register address prefix
PC	Program counter
IP	Instruction pointer
FLAGS	Flags register (D5H)
RP	Register pointer
#	Immediate operand or register address prefix
H	Hexadecimal number suffix
D	Decimal number suffix
B	Binary number suffix
opc	Opcode

Table 6-4. Instruction Notation Conventions

Notation	Description	Actual Operand Range
cc	Condition code	See list of condition codes in Table 6-6.
r	Working register only	Rn (n = 0–15)
rb	Bit (b) of working register	Rn.b (n = 0–15, b = 0–7)
r0	Bit 0 (LSB) of working register	Rn (n = 0–15)
rr	Working register pair	RRp (p = 0, 2, 4, ..., 14)
R	Register or working register	reg or Rn (reg = 0–255, n = 0–15)
Rb	Bit 'b' of register or working register	reg.b (reg = 0–255, b = 0–7)
RR	Register pair or working register pair	reg or RRp (reg = 0–254, even number only, where p = 0, 2, ..., 14)
IA	Indirect addressing mode	addr (addr = 0–254, even number only)
Ir	Indirect working register only	@Rn (n = 0–15)
IR	Indirect register or indirect working register	@Rn or @reg (reg = 0–255, n = 0–15)
Irr	Indirect working register pair only	@RRp (p = 0, 2, ..., 14)
IRR	Indirect register pair or indirect working register pair	@RRp or @reg (reg = 0–254, even only, where p = 0, 2, ..., 14)
X	Indexed addressing mode	#reg [Rn] (reg = 0–255, n = 0–15)
XS	Indexed (short offset) addressing mode	#addr [RRp] (addr = range –128 to +127, where p = 0, 2, ..., 14)
xl	Indexed (long offset) addressing mode	#addr [RRp] (addr = range 0–65535, where p = 0, 2, ..., 14)
da	Direct addressing mode	addr (addr = range 0–65535)
ra	Relative addressing mode	addr (addr = number in the range +127 to –128 that is an offset relative to the address of the next instruction)
im	Immediate addressing mode	#data (data = 0–255)
iml	Immediate (long) addressing mode	#data (data = range 0–65535)

Table 6-5. Opcode Quick Reference

OPCODE MAP									
LOWER NIBBLE (HEX)									
	–	0	1	2	3	4	5	6	7
U	0	DEC R1	DEC IR1	ADD r1,r2	ADD r1,Ir2	ADD R2,R1	ADD IR2,R1	ADD R1,IM	BOR r0–Rb
P	1	RLC R1	RLC IR1	ADC r1,r2	ADC r1,Ir2	ADC R2,R1	ADC IR2,R1	ADC R1,IM	BCP r1.b, R2
P	2	INC R1	INC IR1	SUB r1,r2	SUB r1,Ir2	SUB R2,R1	SUB IR2,R1	SUB R1,IM	BXOR r0–Rb
E	3	JP IRR1	SRP/0/1 IM	SBC r1,r2	SBC r1,Ir2	SBC R2,R1	SBC IR2,R1	SBC R1,IM	BTJR r2.b, RA
R	4	DA R1	DA IR1	OR r1,r2	OR r1,Ir2	OR R2,R1	OR IR2,R1	OR R1,IM	LDB r0–Rb
	5	POP R1	POP IR1	AND r1,r2	AND r1,Ir2	AND R2,R1	AND IR2,R1	AND R1,IM	BITC r1.b
N	6	COM R1	COM IR1	TCM r1,r2	TCM r1,Ir2	TCM R2,R1	TCM IR2,R1	TCM R1,IM	BAND r0–Rb
I	7	PUSH R2	PUSH IR2	TM r1,r2	TM r1,Ir2	TM R2,R1	TM IR2,R1	TM R1,IM	BIT r1.b
B	8	DECW RR1	DECW IR1	PUSHUD IR1,R2	PUSHUI IR1,R2	MULT R2,RR1	MULT IR2,RR1	MULT IM,RR1	LD r1, x, r2
B	9	RL R1	RL IR1	POPUD IR2,R1	POPUI IR2,R1	DIV R2,RR1	DIV IR2,RR1	DIV IM,RR1	LD r2, x, r1
L	A	INCW RR1	INCW IR1	CP r1,r2	CP r1,Ir2	CP R2,R1	CP IR2,R1	CP R1,IM	LDC r1, Irr2, xL
E	B	CLR R1	CLR IR1	XOR r1,r2	XOR r1,Ir2	XOR R2,R1	XOR IR2,R1	XOR R1,IM	LDC r2, Irr2, xL
	C	RRC R1	RRC IR1	CPIJE Ir,r2,RA	LDC r1,Irr2	LDW RR2,RR1	LDW IR2,RR1	LDW RR1,IML	LD r1, Ir2
H	D	SRA R1	SRA IR1	CPIJNE Irr,r2,RA	LDC r2,Irr1	CALL IA1		LD IR1,IM	LD Ir1, r2
E	E	RR R1	RR IR1	LDCD r1,Irr2	LDCI r1,Irr2	LD R2,R1	LD R2,IR1	LD R1,IM	LDC r1, Irr2, xs
X	F	SWAP R1	SWAP IR1	LDCPD r2,Irr1	LDCPI r2,Irr1	CALL IRR1	LD IR2,R1	CALL DA1	LDC r2, Irr1, xs

Table 6-5. Opcode Quick Reference (Continued)

OPCODE MAP									
LOWER NIBBLE (HEX)									
	–	8	9	A	B	C	D	E	F
U	0	LD r1,R2	LD r2,R1	DJNZ r1,RA	JR cc,RA	LD r1,IM	JP cc,DA	INC r1	NEXT
P	1	↓	↓	↓	↓	↓	↓	↓	ENTER
P	2								EXIT
E	3								WFI
R	4								SB0
	5								SB1
N	6								IDLE
I	7	↓	↓	↓	↓	↓	↓	↓	STOP
B	8								DI
B	9								EI
L	A								RET
E	B								IRET
	C								RCF
H	D	↓	↓	↓	↓	↓	↓	↓	SCF
E	E								CCF
X	F	LD r1,R2	LD r2,R1	DJNZ r1,RA	JR cc,RA	LD r1,IM	JP cc,DA	INC r1	NOP

CONDITION CODES

The opcode of a conditional jump always contains a 4-bit field called the condition code (cc). This specifies under which conditions it is to execute the jump. For example, a conditional jump with the condition code for "equal" after a compare operation only jumps if the two operands are equal. Condition codes are listed in Table 6-6.

The carry (C), zero (Z), sign (S), and overflow (V) flags are used to control the operation of conditional jump instructions.

Table 6-6. Condition Codes

Binary	Mnemonic	Description	Flags Set
0000	F	Always false	–
1000	T	Always true	–
0111 (note)	C	Carry	C = 1
1111 (note)	NC	No carry	C = 0
0110 (note)	Z	Zero	Z = 1
1110 (note)	NZ	Not zero	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No overflow	V = 0
0110 (note)	EQ	Equal	Z = 1
1110 (note)	NE	Not equal	Z = 0
1001	GE	Greater than or equal	(S XOR V) = 0
0001	LT	Less than	(S XOR V) = 1
1010	GT	Greater than	(Z OR (S XOR V)) = 0
0010	LE	Less than or equal	(Z OR (S XOR V)) = 1
1111 (note)	UGE	Unsigned greater than or equal	C = 0
0111 (note)	ULT	Unsigned less than	C = 1
1011	UGT	Unsigned greater than	(C = 0 AND Z = 0) = 1
0011	ULE	Unsigned less than or equal	(C OR Z) = 1

NOTES:

1. It indicates condition codes that are related to two different mnemonics but which test the same flag. For example, Z and EQ are both true if the zero flag (Z) is set, but after an ADD instruction, Z would probably be used; after a CP instruction, however, EQ would probably be used.
2. For operations involving unsigned numbers, the special condition codes UGE, ULT, UGT, and ULE must be used.

INSTRUCTION DESCRIPTIONS

This section contains detailed information and programming examples for each instruction in the SAM8 instruction set. Information is arranged in a consistent format for improved readability and for fast referencing. The following information is included in each instruction description:

- Instruction name (mnemonic)
- Full instruction name
- Source/destination format of the instruction operand
- Shorthand notation of the instruction's operation
- Textual description of the instruction's effect
- Specific flag settings affected by the instruction
- Detailed description of the instruction's format, execution time, and addressing mode(s)
- Programming example(s) explaining how to use the instruction

ADC — Add with carry

ADC dst,src

Operation: dst ← dst + src + c

The source operand, along with the setting of the carry flag, is added to the destination operand and the sum is stored in the destination. The contents of the source are unaffected. Two's-complement addition is performed. In multiple precision arithmetic, this instruction permits the carry from the addition of low-order operands to be carried into the addition of high-order operands.

Flags:

C: Set if there is a carry from the most significant bit of the result; cleared otherwise.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
V: Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise.
D: Always cleared to "0".
H: Set if there is a carry from the most significant bit of the low-order four bits of the result; cleared otherwise.

Format:

	Bytes	**Cycles**	**Opcode**	**Addr Mode**	
			(Hex)	**dst**	**src**
opc │ dst | src	2	4	12	r	r
		6	13	r	Ir
opc │ src │ dst	3	6	14	R	R
		6	15	R	IR
opc │ dst │ src	3	6	16	R	IM

Examples: Given: R1 = 10H, R2 = 03H, C flag = "1", register 01H = 20H, register 02H = 03H, and register 03H = 0AH:

ADC	R1,R2	→	R1 = 14H, R2 = 03H
ADC	R1,@R2	→	R1 = 1BH, R2 = 03H
ADC	01H,02H	→	Register 01H = 24H, register 02H = 03H
ADC	01H,@02H	→	Register 01H = 2BH, register 02H = 03H
ADC	01H,#11H	→	Register 01H = 32H

In the first example, destination register R1 contains the value 10H, the carry flag is set to "1", and the source working register R2 contains the value 03H. The statement "ADC R1,R2" adds 03H and the carry flag value ("1") to the destination value 10H, leaving 14H in register R1.

ADD — Add

ADD dst,src

Operation: dst ← dst + src

The source operand is added to the destination operand and the sum is stored in the destination. The contents of the source are unaffected. Two's-complement addition is performed.

Flags:

C: Set if there is a carry from the most significant bit of the result; cleared otherwise.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise.
D: Always cleared to "0".
H: Set if a carry from the low-order nibble occurred.

Format:

	Bytes	**Cycles**	**Opcode**	**Addr Mode**	
			(Hex)	**dst**	**src**
opc | dst | src	2	4	02	r	r
		6	03	r	Ir
opc | src | dst	3	6	04	R	R
		6	05	R	IR
opc | dst | src	3	6	06	R	IM

Examples: Given: R1 = 12H, R2 = 03H, register 01H = 21H, register 02H = 03H, register 03H = 0AH:

```
ADD   R1,R2      →   R1 = 15H, R2 = 03H
ADD   R1,@R2     →   R1 = 1CH, R2 = 03H
ADD   01H,02H    →   Register 01H = 24H, register 02H = 03H
ADD   01H,@02H   →   Register 01H = 2BH, register 02H = 03H
ADD   01H,#25H   →   Register 01H = 46H
```

In the first example, destination working register R1 contains 12H and the source working register R2 contains 03H. The statement "ADD R1,R2" adds 03H to 12H, leaving the value 15H in register R1.

AND — Logical AND

AND dst,src

Operation: dst ← dst AND src

The source operand is logically ANDed with the destination operand. The result is stored in the destination. The AND operation results in a "1" bit being stored whenever the corresponding bits in the two operands are both logic ones; otherwise a "0" bit value is stored. The contents of the source are unaffected.

Flags:

C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Always cleared to "0".
D: Unaffected.
H: Unaffected.

Format:

	Bytes	Cycles	Opcode	Addr Mode	
			(Hex)	dst	src
opc | dst | src	2	4	52	r	r
		6	53	r	Ir
opc | src | dst	3	6	54	R	R
		6	55	R	IR
opc | dst | src	3	6	56	R	IM

Examples: Given: R1 = 12H, R2 = 03H, register 01H = 21H, register 02H = 03H, register 03H = 0AH:

AND R1,R2 → R1 = 02H, R2 = 03H
AND R1,@R2 → R1 = 02H, R2 = 03H
AND 01H,02H → Register 01H = 01H, register 02H = 03H
AND 01H,@02H → Register 01H = 00H, register 02H = 03H
AND 01H,#25H → Register 01H = 21H

In the first example, destination working register R1 contains the value 12H and the source working register R2 contains 03H. The statement "AND R1,R2" logically ANDs the source operand 03H with the destination operand value 12H, leaving the value 02H in register R1.

BAND — Bit AND

BAND dst,src.b

BAND dst.b,src

Operation: dst(0) ← dst(0) AND src(b)

or

dst(b) ← dst(b) AND src(0)

The specified bit of the source (or the destination) is logically ANDed with the zero bit (LSB) of the destination (or source). The resultant bit is stored in the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags:

C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Cleared to "0".
V: Undefined.
D: Unaffected.
H: Unaffected.

Format:

			Bytes	**Cycles**	**Opcode**	**Addr Mode**	
					(Hex)	**dst**	**src**
opc	dst | b | 0	src	3	6	67	r0	Rb
opc	src | b | 1	dst	3	6	67	Rb	r0

NOTE: In the second byte of the 3-byte instruction formats, the destination (or source) address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Examples: Given: R1 = 07H and register 01H = 05H:

BAND	R1,01H.1	→	R1 = 06H, register 01H = 05H
BAND	01H.1,R1	→	Register 01H = 05H, R1 = 07H

In the first example, source register 01H contains the value 05H (00000101B) and destination working register R1 contains 07H (00000111B). The statement "BAND R1,01H.1" ANDs the bit 1 value of the source register ("0") with the bit 0 value of register R1 (destination), leaving the value 06H (00000110B) in register R1.

BCP — Bit Compare

BCP dst,src.b

Operation: dst(0) – src(b)

The specified bit of the source is compared to (subtracted from) bit zero (LSB) of the destination. The zero flag is set if the bits are the same; otherwise it is cleared. The contents of both operands are unaffected by the comparison.

Flags:
C: Unaffected.
Z: Set if the two bits are the same; cleared otherwise.
S: Cleared to "0".
V: Undefined.
D: Unaffected.
H: Unaffected.

Format:

opc	dst | b | 0	src

Bytes	Cycles	Opcode	Addr Mode	
		(Hex)	dst	src
3	6	17	r0	Rb

NOTE: In the second byte of the instruction format, the destination address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H and register 01H = 01H:

BCP R1,01H.1 → R1 = 07H, register 01H = 01H

If destination working register R1 contains the value 07H (00000111B) and the source register 01H contains the value 01H (00000001B), the statement "BCP R1,01H.1" compares bit one of the source register (01H) and bit zero of the destination register (R1). Because the bit values are not identical, the zero flag bit (Z) is cleared in the FLAGS register (0D5H).

BITC — Bit Complement

BITC dst.b

Operation: dst(b) ← NOT dst(b)

This instruction complements the specified bit within the destination without affecting any other bits in the destination.

Flags:

C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Cleared to "0".
V: Undefined.
D: Unaffected.
H: Unaffected.

Format:

		Bytes	**Cycles**	**Opcode**	**Addr Mode**
				(Hex)	**dst**
opc	dst | b | 0	2	4	57	rb

NOTE: In the second byte of the instruction format, the destination address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H

BITC R1.1 → R1 = 05H

If working register R1 contains the value 07H (00000111B), the statement "BITC R1.1" complements bit one of the destination and leaves the value 05H (00000101B) in register R1. Because the result of the complement is not "0", the zero flag (Z) in the FLAGS register (0D5H) is cleared.

BITR — Bit Reset

BITR dst.b

Operation: dst(b) ← 0

The BITR instruction clears the specified bit within the destination without affecting any other bits in the destination.

Flags: No flags are affected.

Format:

opc	dst | b | 0	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
		2	4	77	rb

NOTE: In the second byte of the instruction format, the destination address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H:

BITR R1.1 → R1 = 05H

If the value of working register R1 is 07H (00000111B), the statement "BITR R1.1" clears bit one of the destination register R1, leaving the value 05H (00000101B).

BITS — Bit Set

BITS dst.b

Operation: dst(b) ← 1

The BITS instruction sets the specified bit within the destination without affecting any other bits in the destination.

Flags: No flags are affected.

Format:

		Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**
opc	dst | b | 1	2	4	77	rb

NOTE: In the second byte of the instruction format, the destination address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H:

BITS R1.3 → R1 = 0FH

If working register R1 contains the value 07H (00000111B), the statement "BITS R1.3" sets bit three of the destination register R1 to "1", leaving the value 0FH (00001111B).

BOR — Bit OR

BOR dst,src.b

BOR dst.b,src

Operation: dst(0) ← dst(0) OR src(b)

or

dst(b) ← dst(b) OR src(0)

The specified bit of the source (or the destination) is logically ORed with bit zero (LSB) of the destination (or the source). The resulting bit value is stored in the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags:

C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Cleared to "0".
V: Undefined.
D: Unaffected.
H: Unaffected.

Format:

			Bytes	Cycles	Opcode	Addr Mode	
					(Hex)	dst	src
opc	dst | b | 0	src	3	6	07	r0	Rb
opc	src | b | 1	dst	3	6	07	Rb	r0

NOTE: In the second byte of the 3-byte instruction formats, the destination (or source) address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit.

Examples: Given: R1 = 07H and register 01H = 03H:

BOR	R1, 01H.1	→	R1 = 07H, register 01H = 03H
BOR	01H.2, R1	→	Register 01H = 07H, R1 = 07H

In the first example, destination working register R1 contains the value 07H (00000111B) and source register 01H the value 03H (00000011B). The statement "BOR R1,01H.1" logically ORs bit one of register 01H (source) with bit zero of R1 (destination). This leaves the same value (07H) in working register R1.

In the second example, destination register 01H contains the value 03H (00000011B) and the source working register R1 the value 07H (00000111B). The statement "BOR 01H.2,R1" logically ORs bit two of register 01H (destination) with bit zero of R1 (source). This leaves the value 07H in register 01H.

BTJRF — Bit Test, Jump Relative on False

BTJRF dst,src.b

Operation: If src(b) is a "0", then PC ← PC + dst

The specified bit within the source operand is tested. If it is a "0", the relative address is added to the program counter and control passes to the statement whose address is now in the PC; otherwise, the instruction following the BTJRF instruction is executed.

Flags: No flags are affected.

Format:

			Bytes	**Cycles**	**Opcode**	**Addr Mode**	
	(Note 1)				**(Hex)**	**dst**	**src**
opc	src | b | 0	dst	3	10	37	RA	rb

NOTE: In the second byte of the instruction format, the source address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H:

BTJRF SKIP,R1.3 → PC jumps to SKIP location

If working register R1 contains the value 07H (00000111B), the statement "BTJRF SKIP,R1.3" tests bit 3. Because it is "0", the relative address is added to the PC and the PC jumps to the memory location pointed to by the SKIP. (Remember that the memory location must be within the allowed range of + 127 to – 128.)

BTJRT — Bit Test, Jump Relative on True

BTJRT dst,src.b

Operation: If src(b) is a "1", then PC ← PC + dst

The specified bit within the source operand is tested. If it is a "1", the relative address is added to the program counter and control passes to the statement whose address is now in the PC; otherwise, the instruction following the BTJRT instruction is executed.

Flags: No flags are affected.

Format:

				Bytes	**Cycles**	**Opcode**	**Addr Mode**	
	(Note 1)					**(Hex)**	**dst**	**src**
opc	src | b | 1	dst		3	10	37	RA	rb

NOTE: In the second byte of the instruction format, the source address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H:

BTJRT SKIP,R1.1

If working register R1 contains the value 07H (00000111B), the statement "BTJRT SKIP,R1.1" tests bit one in the source register (R1). Because it is a "1", the relative address is added to the PC and the PC jumps to the memory location pointed to by the SKIP. (Remember that the memory location must be within the allowed range of + 127 to – 128.)

BXOR — Bit XOR

BXOR dst,src.b

BXOR dst.b,src

Operation: dst(0) ← dst(0) XOR src(b)

or

dst(b) ← dst(b) XOR src(0)

The specified bit of the source (or the destination) is logically exclusive-ORed with bit zero (LSB) of the destination (or source). The result bit is stored in the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags:

C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Cleared to "0".
V: Undefined.
D: Unaffected.
H: Unaffected.

Format:

			Bytes	**Cycles**	**Opcode**	**Addr Mode**	
					(Hex)	**dst**	**src**
opc	dst | b | 0	src	3	6	27	r0	Rb
opc	src | b | 1	dst	3	6	27	Rb	r0

NOTE: In the second byte of the 3-byte instruction formats, the destination (or source) address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Examples: Given: R1 = 07H (00000111B) and register 01H = 03H (00000011B):

BXOR R1,01H.1 → R1 = 06H, register 01H = 03H

BXOR 01H.2,R1 → Register 01H = 07H, R1 = 07H

In the first example, destination working register R1 has the value 07H (00000111B) and source register 01H has the value 03H (00000011B). The statement "BXOR R1,01H.1" exclusive-ORs bit one of register 01H (source) with bit zero of R1 (destination). The result bit value is stored in bit zero of R1, changing its value from 07H to 06H. The value of source register 01H is unaffected.

CALL — Call Procedure

CALL dst

Operation:

SP	←	SP – 1
@SP	←	PCL
SP	←	SP –1
@SP	←	PCH
PC	←	dst

The current contents of the program counter are pushed onto the top of the stack. The program counter value used is the address of the first instruction following the CALL instruction. The specified destination address is then loaded into the program counter and points to the first instruction of a procedure. At the end of the procedure the return instruction (RET) can be used to return to the original program flow. RET pops the top of the stack back into the program counter.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst	3	14	F6	DA
opc | dst	2	12	F4	IRR
opc | dst	2	14	D4	IA

Examples: Given: R0 = 35H, R1 = 21H, PC = 1A47H, and SP = 0002H:

CALL 3521H → SP = 0000H

(Memory locations 0000H = 1AH, 0001H = 4AH, where 4AH is the address that follows the instruction.)

CALL @RR0 → SP = 0000H (0000H = 1AH, 0001H = 49H)

CALL #40H → SP = 0000H (0000H = 1AH, 0001H = 49H)

In the first example, if the program counter value is 1A47H and the stack pointer contains the value 0002H, the statement "CALL 3521H" pushes the current PC value onto the top of the stack. The stack pointer now points to memory location 0000H. The PC is then loaded with the value 3521H, the address of the first instruction in the program sequence to be executed.

If the contents of the program counter and stack pointer are the same as in the first example, the statement "CALL @RR0" produces the same result except that the 49H is stored in stack location 0001H (because the two-byte instruction format was used). The PC is then loaded with the value 3521H, the address of the first instruction in the program sequence to be executed. Assuming that the contents of the program counter and stack pointer are the same as in the first example, if program address 0040H contains 35H and program address 0041H contains 21H, the statement "CALL #40H" produces the same result as in the second example.

CCF — Complement Carry Flag

CCF

Operation: C ← NOT C

The carry flag (C) is complemented. If C = "1", the value of the carry flag is changed to logic zero; if C = "0", the value of the carry flag is changed to logic one.

Flags: **C:** Complemented.

No other flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	EF

Example: Given: The carry flag = "0":

CCF

If the carry flag = "0", the CCF instruction complements it in the FLAGS register (0D5H), changing its value from logic zero to logic one.

CLR — Clear

CLR dst

Operation: dst ← "0"

The destination location is cleared to "0".

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
opc | dst	2	4	B0	R
		4	B1	IR

Examples: Given: Register 00H = 4FH, register 01H = 02H, and register 02H = 5EH:

CLR 00H → Register 00H = 00H

CLR @01H → Register 01H = 02H, register 02H = 00H

In Register (R) addressing mode, the statement "CLR 00H" clears the destination register 00H value to 00H. In the second example, the statement "CLR @01H" uses Indirect Register (IR) addressing mode to clear the 02H register value to 00H.

COM — Complement

COM dst

Operation: dst ← NOT dst

The contents of the destination location are complemented (one's complement); all "1s" are changed to "0s", and vice-versa.

Flags:

C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Always reset to "0".
D: Unaffected.
H: Unaffected.

Format:

opc	dst

Bytes	Cycles	Opcode (Hex)	Addr Mode dst
2	4	60	R
	4	61	IR

Examples: Given: R1 = 07H and register 07H = 0F1H:

COM	R1	→	R1 = 0F8H
COM	@R1	→	R1 = 07H, register 07H = 0EH

In the first example, destination working register R1 contains the value 07H (00000111B). The statement "COM R1" complements all the bits in R1: all logic ones are changed to logic zeros, and vice-versa, leaving the value 0F8H (11111000B).

In the second example, Indirect Register (IR) addressing mode is used to complement the value of destination register 07H (11110001B), leaving the new value 0EH (00001110B).

CP — Compare

CP dst,src

Operation: dst – src

The source operand is compared to (subtracted from) the destination operand, and the appropriate flags are set accordingly. The contents of both operands are unaffected by the comparison.

Flags:

C: Set if a "borrow" occurred (src > dst); cleared otherwise.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
V: Set if arithmetic overflow occurred; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

	Bytes	Cycles	Opcode	Addr Mode	
			(Hex)	dst	src
opc | dst | src	2	4	A2	r	r
		6	A3	r	lr
opc | src | dst	3	6	A4	R	R
		6	A5	R	IR
opc | dst | src	3	6	A6	R	IM

Examples:

1. Given: R1 = 02H and R2 = 03H:

```
          CP      R1,R2  →      Set the C and S flags
```

Destination working register R1 contains the value 02H and source register R2 contains the value 03H. The statement "CP R1,R2" subtracts the R2 value (source/subtrahend) from the R1 value (destination/minuend). Because a "borrow" occurs and the difference is negative, C and S are "1".

2. Given: R1 = 05H and R2 = 0AH:

```
          CP      R1,R2
          JP      UGE,SKIP
          INC     R1
SKIP      LD      R3,R1
```

In this example, destination working register R1 contains the value 05H which is less than the contents of the source working register R2 (0AH). The statement "CP R1,R2" generates C = "1" and the JP instruction does not jump to the SKIP location. After the statement "LD R3,R1" executes, the value 06H remains in working register R3.

CPIJE — Compare, Increment, and Jump on Equal

CPIJE dst,src,RA

Operation: If dst – src = "0", PC ← PC + RA

Ir ← Ir + 1

The source operand is compared to (subtracted from) the destination operand. If the result is "0", the relative address is added to the program counter and control passes to the statement whose address is now in the program counter. Otherwise, the instruction immediately following the CPIJE instruction is executed. In either case, the source pointer is incremented by one before the next instruction is executed.

Flags: No flags are affected.

Format:

				Bytes	Cycles	Opcode	Addr Mode	
						(Hex)	dst	src
opc	src	dst	RA	3	12	C2	r	Ir

NOTE: Execution time is 18 cycles if the jump is taken or 16 cycles if it is not taken.

Example: Given: R1 = 02H, R2 = 03H, and register 03H = 02H:

CPIJE R1,@R2,SKIP → R2 = 04H, PC jumps to SKIP location

In this example, working register R1 contains the value 02H, working register R2 the value 03H, and register 03 contains 02H. The statement "CPIJE R1,@R2,SKIP" compares the @R2 value 02H (00000010B) to 02H (00000010B). Because the result of the comparison is *equal*, the relative address is added to the PC and the PC then jumps to the memory location pointed to by SKIP. The source register (R2) is incremented by one, leaving a value of 04H. (Remember that the memory location must be within the allowed range of + 127 to – 128.)

CPIJNE — Compare, Increment, and Jump on Non-Equal

CPIJNE dst,src,RA

Operation: If dst – src "0", PC ← PC + RA

Ir ← Ir + 1

The source operand is compared to (subtracted from) the destination operand. If the result is not "0", the relative address is added to the program counter and control passes to the statement whose address is now in the program counter; otherwise the instruction following the CPIJNE instruction is executed. In either case the source pointer is incremented by one before the next instruction.

Flags: No flags are affected.

Format:

				Bytes	Cycles	Opcode	Addr Mode	
						(Hex)	**dst**	**src**
opc	src	dst	RA	3	12	D2	r	Ir

NOTE: Execution time is 18 cycles if the jump is taken or 16 cycles if it is not taken.

Example: Given: R1 = 02H, R2 = 03H, and register 03H = 04H:

CPIJNER1,@R2,SKIP → R2 = 04H, PC jumps to SKIP location

Working register R1 contains the value 02H, working register R2 (the source pointer) the value 03H, and general register 03 the value 04H. The statement "CPIJNE R1,@R2,SKIP" subtracts 04H (00000100B) from 02H (00000010B). Because the result of the comparison is *non-equal*, the relative address is added to the PC and the PC then jumps to the memory location pointed to by SKIP. The source pointer register (R2) is also incremented by one, leaving a value of 04H. (Remember that the memory location must be within the allowed range of + 127 to – 128.)

DA — Decimal Adjust

DA dst

Operation: dst ← DA dst

The destination operand is adjusted to form two 4-bit BCD digits following an addition or subtraction operation. For addition (ADD, ADC) or subtraction (SUB, SBC), the following table indicates the operation performed. (The operation is undefined if the destination operand was not the result of a valid addition or subtraction of BCD digits):

Instruction	Carry Before DA	Bits 4–7 Value (Hex)	H Flag Before DA	Bits 0–3 Value (Hex)	Number Added to Byte	Carry After DA
	0	0–9	0	0–9	00	0
	0	0–8	0	A–F	06	0
	0	0–9	1	0–3	06	0
ADD	0	A–F	0	0–9	60	1
ADC	0	9–F	0	A–F	66	1
	0	A–F	1	0–3	66	1
	1	0–2	0	0–9	60	1
	1	0–2	0	A–F	66	1
	1	0–3	1	0–3	66	1
	0	0–9	0	0–9	00 = – 00	0
SUB	0	0–8	1	6–F	FA = – 06	0
SBC	1	7–F	0	0–9	A0 = – 60	1
	1	6–F	1	6–F	9A = – 66	1

Flags:

C: Set if there was a carry from the most significant bit; cleared otherwise (see table).
Z: Set if result is "0"; cleared otherwise.
S: Set if result bit 7 is set; cleared otherwise.
V: Undefined.
D: Unaffected.
H: Unaffected.

Format:

opc	dst

Bytes	Cycles	Opcode (Hex)	Addr Mode dst
2	4	40	R
	4	41	IR

DA — Decimal Adjust

DA (Continued)

Example: Given: Working register R0 contains the value 15 (BCD), working register R1 contains 27 (BCD), and address 27H contains 46 (BCD):

```
ADD     R1,R0     ;     C ← "0", H ← "0", Bits 4–7 = 3, bits 0–3 = C, R1 ← 3CH
DA      R1        ;     R1 ← 3CH + 06
```

If addition is performed using the BCD values 15 and 27, the result should be 42. The sum is incorrect, however, when the binary representations are added in the destination location using standard binary arithmetic:

```
     0 0 0 1   0 1 0 1         15
   + 0 0 1 0   0 1 1 1         27
     0 0 1 1   1 1 0 0    =    3CH
```

The DA instruction adjusts this result so that the correct BCD representation is obtained:

```
     0 0 1 1   1 1 0 0
   + 0 0 0 0   0 1 1 0
     0 1 0 0   0 0 1 0    =    42
```

Assuming the same values given above, the statements

```
SUB    27H,R0 ;       C ← "0", H ← "0", Bits 4–7 = 3, bits 0–3 = 1
DA     @R1    ;       @R1 ← 31–0
```

leave the value 31 (BCD) in address 27H (@R1).

DEC — Decrement

DEC dst

Operation: dst ← dst – 1

The contents of the destination operand are decremented by one.

Flags:

C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Set if result is negative; cleared otherwise.
V: Set if arithmetic overflow occurred; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

		Bytes	**Cycles**	**Opcode**	**Addr Mode**
				(Hex)	**dst**
opc	dst	2	4	00	R
			4	01	IR

Examples: Given: R1 = 03H and register 03H = 10H:

DEC R1 → R1 = 02H
DEC @R1 → Register 03H = 0FH

In the first example, if working register R1 contains the value 03H, the statement "DEC R1" decrements the hexadecimal value by one, leaving the value 02H. In the second example, the statement "DEC @R1" decrements the value 10H contained in the destination register 03H by one, leaving the value 0FH.

DECW — Decrement Word

DECW dst

Operation: dst ← dst – 1

The contents of the destination location (which must be an even address) and the operand following that location are treated as a single 16-bit value that is decremented by one.

Flags:

C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
V: Set if arithmetic overflow occurred; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

		Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode** **dst**
opc	dst	2	8	80	RR
			8	81	IR

Examples: Given: R0 = 12H, R1 = 34H, R2 = 30H, register 30H = 0FH, and register 31H = 21H:

DECW RR0 → R0 = 12H, R1 = 33H
DECW @R2 → Register 30H = 0FH, register 31H = 20H

In the first example, destination register R0 contains the value 12H and register R1 the value 34H. The statement "DECW RR0" addresses R0 and the following operand R1 as a 16-bit word and decrements the value of R1 by one, leaving the value 33H.

NOTE: A system malfunction may occur if you use a Zero flag (FLAGS.6) result together with a DECW instruction. To avoid this problem, we recommend that you use DECW as shown in the following example:

```
LOOP: DECW  RR0
      LD    R2,R1
      OR    R2,R0
      JR    NZ,LOOP
```

DI — Disable Interrupts

DI

Operation: SYM (0) ← 0

Bit zero of the system mode control register, SYM.0, is cleared to "0", globally disabling all interrupt processing. Interrupt requests will continue to set their respective interrupt pending bits, but the CPU will not service them while interrupt processing is disabled.

Flags: No flags are affected.

Format:

	Bytes	**Cycles**	**Opcode (Hex)**
opc	1	4	8F

Example: Given: SYM = 01H:

DI

If the value of the SYM register is 01H, the statement "DI" leaves the new value 00H in the register and clears SYM.0 to "0", disabling interrupt processing.

Before changing IMR, interrupt pending and interrupt source control register, be sure DI state.

DIV — Divide (Unsigned)

DIV dst,src

Operation: dst ÷ src

dst (UPPER) ← REMAINDER

dst (LOWER) ← QUOTIENT

The destination operand (16 bits) is divided by the source operand (8 bits). The quotient (8 bits) is stored in the lower half of the destination. The remainder (8 bits) is stored in the upper half of the destination. When the quotient is ≥ 2^8, the numbers stored in the upper and lower halves of the destination for quotient and remainder are incorrect. Both operands are treated as unsigned integers.

Flags:

C: Set if the V flag is set and quotient is between 2^8 and 2^9 –1; cleared otherwise.
Z: Set if divisor or quotient = "0"; cleared otherwise.
S: Set if MSB of quotient = "1"; cleared otherwise.
V: Set if quotient is ≥ 2^8 or if divisor = "0"; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

opc	src	dst

Bytes	Cycles	Opcode (Hex)	Addr Mode dst	 src
3	26/10	94	RR	R
	26/10	95	RR	IR
	26/10	96	RR	IM

NOTE: Execution takes 10 cycles if the divide-by-zero is attempted; otherwise it takes 26 cycles.

Examples: Given: R0 = 10H, R1 = 03H, R2 = 40H, register 40H = 80H:

```
DIV    RR0,R2      →     R0 = 03H, R1 = 40H
DIV    RR0,@R2     →     R0 = 03H, R1 = 20H
DIV    RR0,#20H    →     R0 = 03H, R1 = 80H
```

In the first example, destination working register pair RR0 contains the values 10H (R0) and 03H (R1), and register R2 contains the value 40H. The statement "DIV RR0,R2" divides the 16-bit RR0 value by the 8-bit value of the R2 (source) register. After the DIV instruction, R0 contains the value 03H and R1 contains 40H. The 8-bit remainder is stored in the upper half of the destination register RR0 (R0) and the quotient in the lower half (R1).

DJNZ — Decrement and Jump if Non-Zero

DJNZ r,dst

Operation: r ← r – 1

If r ≠ 0, PC ← PC + dst

The working register being used as a counter is decremented. If the contents of the register are not logic zero after decrementing, the relative address is added to the program counter and control passes to the statement whose address is now in the PC. The range of the relative address is +127 to –128, and the original value of the PC is taken to be the address of the instruction byte following the DJNZ statement.

NOTE: In case of using DJNZ instruction, the working register being used as a counter should be set at the one of location 0C0H to 0CFH with SRP, SRP0, or SRP1 instruction.

Flags: No flags are affected.

Format:

		Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**
r | opc	dst	2	8 (jump taken)	rA	RA
			8 (no jump)	r = 0 to F	

Example: Given: R1 = 02H and LOOP is the label of a relative address:

```
SRP     #0C0H
DJNZ    R1,LOOP
```

DJNZ is typically used to control a "loop" of instructions. In many cases, a label is used as the destination operand instead of a numeric relative address value. In the example, working register R1 contains the value 02H, and LOOP is the label for a relative address.

The statement "DJNZ R1, LOOP" decrements register R1 by one, leaving the value 01H. Because the contents of R1 after the decrement are non-zero, the jump is taken to the relative address specified by the LOOP label.

EI — Enable Interrupts

EI

Operation: SYM (0) ← 1

An EI instruction sets bit zero of the system mode register, SYM.0 to "1". This allows interrupts to be serviced as they occur (assuming they have highest priority). If an interrupt's pending bit was set while interrupt processing was disabled (by executing a DI instruction), it will be serviced when you execute the EI instruction.

Flags: No flags are affected.

Format:

	Bytes	**Cycles**	**Opcode (Hex)**
opc	1	4	9F

Example: Given: SYM = 00H:

EI

If the SYM register contains the value 00H, that is, if interrupts are currently disabled, the statement "EI" sets the SYM register to 01H, enabling all interrupts. (SYM.0 is the enable bit for global interrupt processing.)

ENTER — Enter

ENTER

Operation:

SP	←	SP – 2
@SP	←	IP
IP	←	PC
PC	←	@IP
IP	←	IP + 2

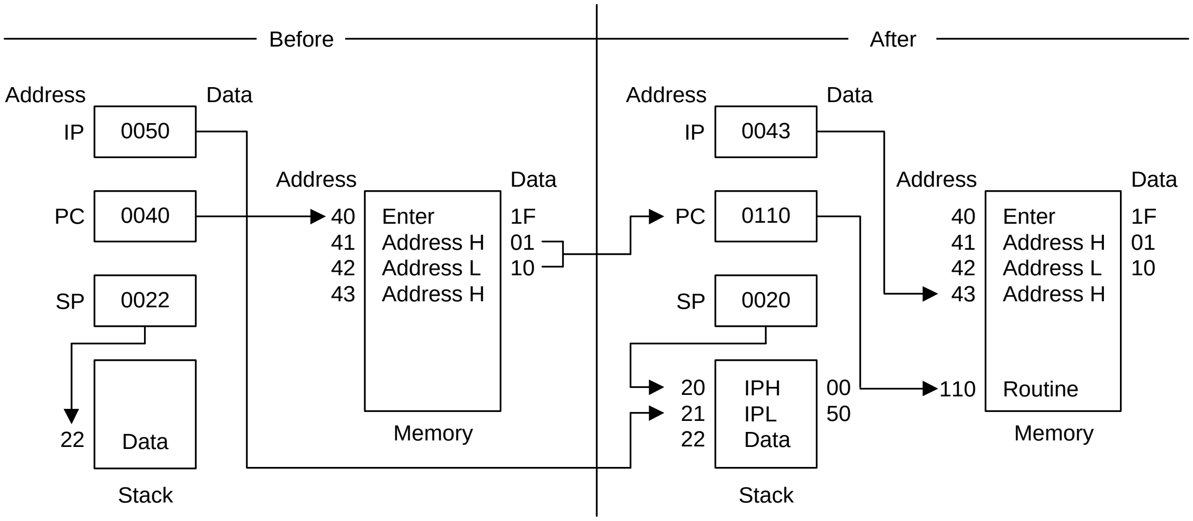
This instruction is useful when implementing threaded-code languages. The contents of the instruction pointer are pushed to the stack. The program counter (PC) value is then written to the instruction pointer. The program memory word that is pointed to by the instruction pointer is loaded into the PC, and the instruction pointer is incremented by two.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	14	1F

Example: The diagram below shows one example of how to use an ENTER statement.



EXIT — Exit

EXIT

Operation:

IP ← @SP
SP ← SP + 2
PC ← @IP
IP ← IP + 2

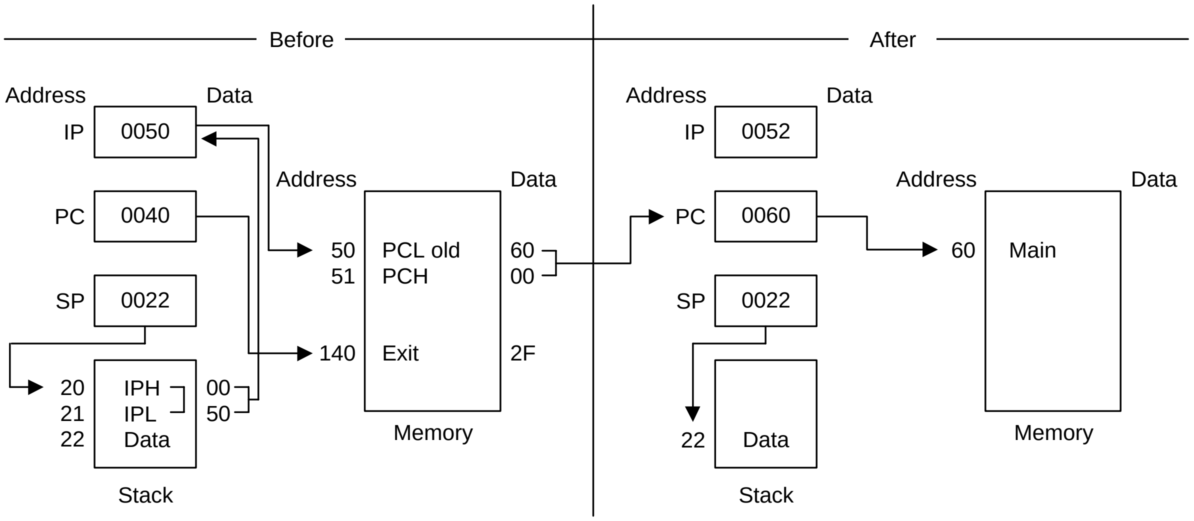
This instruction is useful when implementing threaded-code languages. The stack value is popped and loaded into the instruction pointer. The program memory word that is pointed to by the instruction pointer is then loaded into the program counter, and the instruction pointer is incremented by two.

Flags: No flags are affected.

Format:

	Bytes	**Cycles**	**Opcode (Hex)**
opc	1	14 (internal stack) 16 (internal stack)	2F

Example: The diagram below shows one example of how to use an EXIT statement.



IDLE — Idle Operation

IDLE

Operation:

The IDLE instruction stops the CPU clock while allowing system clock oscillation to continue. Idle mode can be released by an interrupt request (IRQ) or an external reset operation.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	 src
opc	1	4	6F	–	–

Example: The instruction

IDLE

stops the CPU clock but not the system clock.

INC — Increment

INC dst

Operation: dst ← dst + 1

The contents of the destination operand are incremented by one.

Flags:

C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
V: Set if arithmetic overflow occurred; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

	Bytes	**Cycles**	**Opcode**	**Addr Mode**
			(Hex)	**dst**
dst | opc	1	4	rE	r
			r = 0 to F	
opc | dst	2	4	20	R
		4	21	IR

Examples: Given: R0 = 1BH, register 00H = 0CH, and register 1BH = 0FH:

INC	R0	→	R0 = 1CH
INC	00H	→	Register 00H = 0DH
INC	@R0	→	R0 = 1BH, register 01H = 10H

In the first example, if destination working register R0 contains the value 1BH, the statement "INC R0" leaves the value 1CH in that same register.

The next example shows the effect an INC instruction has on register 00H, assuming that it contains the value 0CH.

In the third example, INC is used in Indirect Register (IR) addressing mode to increment the value of register 1BH from 0FH to 10H.

INCW — Increment Word

INCW dst

Operation: dst ← dst + 1

The contents of the destination (which must be an even address) and the byte following that location are treated as a single 16-bit value that is incremented by one.

Flags:

C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
V: Set if arithmetic overflow occurred; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

		Bytes	Cycles	Opcode	Addr Mode
				(Hex)	dst
opc	dst	2	8	A0	RR
			8	A1	IR

Examples: Given: R0 = 1AH, R1 = 02H, register 02H = 0FH, and register 03H = 0FFH:

INCW RR0 → R0 = 1AH, R1 = 03H

INCW @R1 → Register 02H = 10H, register 03H = 00H

In the first example, the working register pair RR0 contains the value 1AH in register R0 and 02H in register R1. The statement "INCW RR0" increments the 16-bit destination by one, leaving the value 03H in register R1. In the second example, the statement "INCW @R1" uses Indirect Register (IR) addressing mode to increment the contents of general register 03H from 0FFH to 00H and register 02H from 0FH to 10H.

NOTE: A system malfunction may occur if you use a Zero (Z) flag (FLAGS.6) result together with an INCW instruction. To avoid this problem, we recommend that you use INCW as shown in the following example:

```
LOOP:   INCW    RR0
        LD      R2,R1
        OR      R2,R0
        JR      NZ,LOOP
```

IRET — Interrupt Return

IRET	IRET (Normal)	IRET (Fast)
Operation:	FLAGS ← @SP	PC ↔ IP
	SP ← SP + 1	FLAGS ← FLAGS'
	PC ← @SP	FIS ← 0
	SP ← SP + 2	
	SYM(0) ← 1	

This instruction is used at the end of an interrupt service routine. It restores the flag register and the program counter. It also re-enables global interrupts. A "normal IRET" is executed only if the fast interrupt status bit (FIS, bit one of the FLAGS register, 0D5H) is cleared (= "0"). If a fast interrupt occurred, IRET clears the FIS bit that was set at the beginning of the service routine.

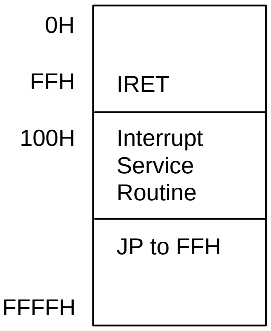
Flags: All flags are restored to their original settings (that is, the settings before the interrupt occurred).

Format:

IRET (Normal)	**Bytes**	**Cycles**	**Opcode (Hex)**
opc	1	10 (internal stack)	BF
		12 (internal stack)	

IRET (Fast)	**Bytes**	**Cycles**	**Opcode (Hex)**
opc	1	6	BF

Example: In the figure below, the instruction pointer is initially loaded with 100H in the main program before interrupts are enabled. When an interrupt occurs, the program counter and instruction pointer are swapped. This causes the PC to jump to address 100H and the IP to keep the return address. The last instruction in the service routine normally is a jump to IRET at address FFH. This causes the instruction pointer to be loaded with 100H "again" and the program counter to jump back to the main program. Now, the next interrupt can occur and the IP is still correct at 100H.



NOTE: In the fast interrupt example above, if the last instruction is not a jump to IRET, you must pay attention to the order of the last two instructions. The IRET cannot be immediately proceded by a clearing of the interrupt status (as with a reset of the IPR register).

JP — Jump

JP cc,dst (Conditional)

JP dst (Unconditional)

Operation: If cc is true, PC ← dst

The conditional JUMP instruction transfers program control to the destination address if the condition specified by the condition code (cc) is true; otherwise, the instruction following the JP instruction is executed. The unconditional JP simply replaces the contents of the PC with the contents of the specified register pair. Control then passes to the statement addressed by the PC.

Flags: No flags are affected.

Format: (1)

		Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**
(2) cc | opc	dst	3	8	ccD	DA
				cc = 0 to F	
opc	dst	2	8	30	IRR

NOTES:

1. The 3-byte format is used for a conditional jump and the 2-byte format for an unconditional jump.
2. In the first byte of the three-byte instruction format (conditional jump), the condition code and the opcode are both four bits.

Examples: Given: The carry flag (C) = "1", register 00 = 01H, and register 01 = 20H:

JP C,LABEL_W → LABEL_W = 1000H, PC = 1000H

JP @00H → PC = 0120H

The first example shows a conditional JP. Assuming that the carry flag is set to "1", the statement "JP C,LABEL_W" replaces the contents of the PC with the value 1000H and transfers control to that location. Had the carry flag not been set, control would then have passed to the statement immediately following the JP instruction.

The second example shows an unconditional JP. The statement "JP @00" replaces the contents of the PC with the contents of the register pair 00H and 01H, leaving the value 0120H.

JR — Jump Relative

JR cc,dst

Operation: If cc is true, PC ← PC + dst

If the condition specified by the condition code (cc) is true, the relative address is added to the program counter and control passes to the statement whose address is now in the program counter; otherwise, the instruction following the JR instruction is executed. (See list of condition codes).

The range of the relative address is +127, –128, and the original value of the program counter is taken to be the address of the first instruction byte following the JR statement.

Flags: No flags are affected.

Format:

(1)

cc	opc	dst

Bytes	Cycles	Opcode (Hex)	Addr Mode **dst**
2	6	ccB	RA
		cc = 0 to F	

NOTE: In the first byte of the two-byte instruction format, the condition code and the opcode are each four bits.

Example: Given: The carry flag = "1" and LABEL_X = 1FF7H:

JR C,LABEL_X → PC = 1FF7H

If the carry flag is set (that is, if the condition code is true), the statement "JR C,LABEL_X" will pass control to the statement whose address is now in the PC. Otherwise, the program instruction following the JR would be executed.

LD — Load

LD dst,src

Operation: dst ← src

The contents of the source are loaded into the destination. The source's contents are unaffected.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode	Addr Mode	
					(Hex)	dst	src
dst | opc	src		2	4	rC	r	IM
				4	r8	r	R
src | opc	dst		2	4	r9	R	r
					r = 0 to F		
opc	dst | src		2	4	C7	r	Ir
				4	D7	Ir	r
opc	src	dst	3	6	E4	R	R
				6	E5	R	IR
opc	dst	src	3	6	E6	R	IM
				6	D6	IR	IM
opc	src	dst	3	6	F5	IR	R
opc	dst | src	x	3	6	87	r	x [r]
opc	src | dst	x	3	6	97	x [r]	r

LD — Load

LD (Continued)

Examples: Given: R0 = 01H, R1 = 0AH, register 00H = 01H, register 01H = 20H, register 02H = 02H, LOOP = 30H, and register 3AH = 0FFH:

LD	R0,#10H	→	R0 = 10H
LD	R0,01H	→	R0 = 20H, register 01H = 20H
LD	01H,R0	→	Register 01H = 01H, R0 = 01H
LD	R1,@R0	→	R1 = 20H, R0 = 01H
LD	@R0,R1	→	R0 = 01H, R1 = 0AH, register 01H = 0AH
LD	00H,01H	→	Register 00H = 20H, register 01H = 20H
LD	02H,@00H	→	Register 02H = 20H, register 00H = 01H
LD	00H,#0AH	→	Register 00H = 0AH
LD	@00H,#10H	→	Register 00H = 01H, register 01H = 10H
LD	@00H,02H	→	Register 00H = 01H, register 01H = 02, register 02H = 02H
LD	R0,#LOOP[R1]	→	R0 = 0FFH, R1 = 0AH
LD	#LOOP[R0],R1	→	Register 31H = 0AH, R0 = 01H, R1 = 0AH

LDB — Load Bit

LDB dst,src.b

LDB dst.b,src

Operation: dst(0) ← src(b)

or

dst(b) ← src(0)

The specified bit of the source is loaded into bit zero (LSB) of the destination, or bit zero of the source is loaded into the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode	Addr Mode	
					(Hex)	dst	src
opc	dst | b | 0	src	3	6	47	r0	Rb
opc	src | b | 1	dst	3	6	47	Rb	r0

NOTE: In the second byte of the instruction formats, the destination (or source) address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Examples: Given: R0 = 06H and general register 00H = 05H:

LDB	R0,00H.2	→	R0 = 07H, register 00H = 05H
LDB	00H.0,R0	→	R0 = 06H, register 00H = 04H

In the first example, destination working register R0 contains the value 06H and the source general register 00H the value 05H. The statement "LD R0,00H.2" loads the bit two value of the 00H register into bit zero of the R0 register, leaving the value 07H in register R0.

In the second example, 00H is the destination register. The statement "LD 00H.0,R0" loads bit zero of register R0 to the specified bit (bit zero) of the destination register, leaving 04H in general register 00H.

LDC/LDE — Load Memory

LDC/LDE dst,src

Operation: dst ← src

This instruction loads a byte from program or data memory into a working register or vice-versa. The source values are unaffected. LDC refers to program memory and LDE to data memory. The assembler makes 'Irr' or 'rr' values an even number for program memory and odd an odd number for data memory.

Flags: No flags are affected.

Format:

					Bytes	**Cycles**	**Opcode**	**Addr Mode**	
							(Hex)	**dst**	**src**
1.	opc	dst | src			2	10	C3	r	Irr
2.	opc	src | dst			2	10	D3	Irr	r
3.	opc	dst | src	XS		3	12	E7	r	XS [rr]
4.	opc	src | dst	XS		3	12	F7	XS [rr]	r
5.	opc	dst | src	XL_L	XL_H	4	14	A7	r	XL [rr]
6.	opc	src | dst	XL_L	XL_H	4	14	B7	XL [rr]	r
7.	opc	dst | 0000	DA_L	DA_H	4	14	A7	r	DA
8.	opc	src | 0000	DA_L	DA_H	4	14	B7	DA	r
9.	opc	dst | 0001	DA_L	DA_H	4	14	A7	r	DA
10.	opc	src | 0001	DA_L	DA_H	4	14	B7	DA	r

NOTES:

1. The source (src) or working register pair [rr] for formats 5 and 6 cannot use register pair 0–1.
2. For formats 3 and 4, the destination address 'XS [rr]' and the source address 'XS [rr]' are each one byte.
3. For formats 5 and 6, the destination address 'XL [rr] and the source address 'XL [rr]' are each two bytes.
4. The DA and r source values for formats 7 and 8 are used to address program memory; the second set of values, used in formats 9 and 10, are used to address data memory.

LDC/LDE — Load Memory

LDC/LDE (Continued)

Examples: Given: R0 = 11H, R1 = 34H, R2 = 01H, R3 = 04H; Program memory locations 0103H = 4FH, 0104H = 1A, 0105H = 6DH, and 1104H = 88H. External data memory locations 0103H = 5FH, 0104H = 2AH, 0105H = 7DH, and 1104H = 98H:

```
LDC         R0,@RR2            ;  R0 ← contents of program memory location 0104H
                               ;  R0 = 1AH, R2 = 01H, R3 = 04H

LDE         R0,@RR2            ;  R0 ← contents of external data memory location 0104H
                               ;  R0 = 2AH, R2 = 01H, R3 = 04H

LDC (note)  @RR2,R0            ;  11H (contents of R0) is loaded into program memory
                               ;  location 0104H (RR2),
                               ;  working registers R0, R2, R3 → no change

LDE         @RR2,R0            ;  11H (contents of R0) is loaded into external data memory
                               ;  location 0104H (RR2),
                               ;  working registers R0, R2, R3 → no change

LDC         R0,#01H[RR2]       ;  R0 ← contents of program memory location 0105H
                               ;  (01H + RR2),
                               ;  R0 = 6DH, R2 = 01H, R3 = 04H

LDE         R0,#01H[RR2]       ;  R0 ← contents of external data memory location 0105H
                               ;  (01H + RR2), R0 = 7DH, R2 = 01H, R3 = 04H

LDC (note)  #01H[RR2],R0       ;  11H (contents of R0) is loaded into program memory location
                               ;  0105H (01H + 0104H)

LDE         #01H[RR2],R0       ;  11H (contents of R0) is loaded into external data memory
                               ;  location 0105H (01H + 0104H)

LDC         R0,#1000H[RR2]     ;  R0 ← contents of program memory location 1104H
                               ;  (1000H + 0104H), R0 = 88H, R2 = 01H, R3 = 04H

LDE         R0,#1000H[RR2]     ;  R0 ← contents of external data memory location 1104H
                               ;  (1000H + 0104H), R0 = 98H, R2 = 01H, R3 = 04H

LDC         R0,1104H           ;  R0 ← contents of program memory location 1104H, R0 = 88H

LDE         R0,1104H           ;  R0 ← contents of external data memory location 1104H,
                               ;  R0 = 98H

LDC (note)  1105H,R0           ;  11H (contents of R0) is loaded into program memory location
                               ;  1105H, (1105H) ← 11H

LDE         1105H,R0           ;  11H (contents of R0) is loaded into external data memory
                               ;  location 1105H, (1105H) ← 11H
```

NOTE: These instructions are not supported by masked ROM type devices.

LDCD/LDED — Load Memory and Decrement

LDCD/LDED dst,src

Operation: dst ← src

rr ← rr – 1

These instructions are used for user stacks or block transfers of data from program or data memory to the register file. The address of the memory location is specified by a working register pair. The contents of the source location are loaded into the destination location. The memory address is then decremented. The contents of the source are unaffected.

LDCD references program memory and LDED references external data memory. The assembler makes 'Irr' an even number for program memory and an odd number for data memory.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode	Addr Mode	
				(Hex)	dst	src
opc	dst | src	2	10	E2	r	Irr

Examples: Given: R6 = 10H, R7 = 33H, R8 = 12H, program memory location 1033H = 0CDH, and external data memory location 1033H = 0DDH:

```
LDCD    R8,@RR6     ;  0CDH (contents of program memory location 1033H) is loaded
                    ;  into R8 and RR6 is decremented by one
                    ;  R8 = 0CDH, R6 = 10H, R7 = 32H (RR6 ← RR6 – 1)

LDED    R8,@RR6     ;  0DDH (contents of data memory location 1033H) is loaded
                    ;  into R8 and RR6 is decremented by one (RR6 ← RR6 – 1)
                    ;  R8 = 0DDH, R6 = 10H, R7 = 32H
```

LDCI/LDEI — Load Memory and Increment

LDCI/LDEI dst,src

Operation: dst ← src

rr ← rr + 1

These instructions are used for user stacks or block transfers of data from program or data memory to the register file. The address of the memory location is specified by a working register pair. The contents of the source location are loaded into the destination location. The memory address is then incremented automatically. The contents of the source are unaffected.

LDCI refers to program memory and LDEI refers to external data memory. The assembler makes 'Irr' even for program memory and odd for data memory.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode	Addr Mode	
				(Hex)	**dst**	**src**
opc	dst | src	2	10	E3	r	Irr

Examples: Given: R6 = 10H, R7 = 33H, R8 = 12H, program memory locations 1033H = 0CDH and 1034H = 0C5H; external data memory locations 1033H = 0DDH and 1034H = 0D5H:

```
LDCI    R8,@RR6     ;  0CDH (contents of program memory location 1033H) is loaded
                    ;  into R8 and RR6 is incremented by one (RR6 ← RR6 + 1)
                    ;  R8 = 0CDH, R6 = 10H, R7 = 34H

LDEI    R8,@RR6     ;  0DDH (contents of data memory location 1033H) is loaded
                    ;  into R8 and RR6 is incremented by one (RR6 ← RR6 + 1)
                    ;  R8 = 0DDH, R6 = 10H, R7 = 34H
```

LDCPD/LDEPD — Load Memory with Pre-Decrement

LDCPD/
LDEPD dst,src

Operation: rr ← rr – 1

dst ← src

These instructions are used for block transfers of data from program or data memory from the register file. The address of the memory location is specified by a working register pair and is first decremented. The contents of the source location are then loaded into the destination location. The contents of the source are unaffected.

LDCPD refers to program memory and LDEPD refers to external data memory. The assembler makes 'Irr' an even number for program memory and an odd number for external data memory.

Flags: No flags are affected.

Format:

		Bytes	**Cycles**	**Opcode**	**Addr Mode**	
				(Hex)	**dst**	**src**
opc	src | dst	2	14	F2	Irr	r

Examples: Given: R0 = 77H, R6 = 30H, and R7 = 00H:

```
LDCPD   @RR6,R0     ;  (RR6 ← RR6 – 1)
                    ;  77H (contents of R0) is loaded into program memory location
                    ;  2FFFH (3000H – 1H)
                    ;  R0 = 77H, R6 = 2FH, R7 = 0FFH

LDEPD   @RR6,R0     ;  (RR6 ← RR6 – 1)
                    ;  77H (contents of R0) is loaded into external data memory
                    ;  location 2FFFH (3000H – 1H)
                    ;  R0 = 77H, R6 = 2FH, R7 = 0FFH
```

LDCPI/LDEPI — Load Memory with Pre-Increment

LDCPI/
LDEPI dst,src

Operation: rr ← rr + 1

dst ← src

These instructions are used for block transfers of data from program or data memory from the register file. The address of the memory location is specified by a working register pair and is first incremented. The contents of the source location are loaded into the destination location. The contents of the source are unaffected.

LDCPI refers to program memory and LDEPI refers to external data memory. The assembler makes 'Irr' an even number for program memory and an odd number for data memory.

Flags: No flags are affected.

Format:

		Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode**	
					dst	**src**
opc	src | dst	2	14	F3	Irr	r

Examples: Given: R0 = 7FH, R6 = 21H, and R7 = 0FFH:

```
LDCPI    @RR6,R0        ;  (RR6 ← RR6 + 1)
                        ;  7FH (contents of R0) is loaded into program memory
                        ;  location 2200H (21FFH + 1H)
                        ;  R0 = 7FH, R6 = 22H, R7 = 00H

LDEPI    @RR6,R0        ;  (RR6 ← RR6 + 1)
                        ;  7FH (contents of R0) is loaded into external data memory
                        ;  location 2200H (21FFH + 1H)
                        ;  R0 = 7FH, R6 = 22H, R7 = 00H
```

LDW — Load Word

LDW dst,src

Operation: dst ← src

The contents of the source (a word) are loaded into the destination. The contents of the source are unaffected.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc | src | dst	3	8	C4	RR	RR
		8	C5	RR	IR
opc | dst | src	4	8	C6	RR	IML

Examples: Given: R4 = 06H, R5 = 1CH, R6 = 05H, R7 = 02H, register 00H = 1AH, register 01H = 02H, register 02H = 03H, and register 03H = 0FH:

LDW	RR6,RR4	→	R6 = 06H, R7 = 1CH, R4 = 06H, R5 = 1CH
LDW	00H,02H	→	Register 00H = 03H, register 01H = 0FH, register 02H = 03H, register 03H = 0FH
LDW	RR2,@R7	→	R2 = 03H, R3 = 0FH,
LDW	04H,@01H	→	Register 04H = 03H, register 05H = 0FH
LDW	RR6,#1234H	→	R6 = 12H, R7 = 34H
LDW	02H,#0FEDH	→	Register 02H = 0FH, register 03H = 0EDH

In the second example, please note that the statement "LDW 00H,02H" loads the contents of the source word 02H, 03H into the destination word 00H, 01H. This leaves the value 03H in general register 00H and the value 0FH in register 01H.

The other examples show how to use the LDW instruction with various addressing modes and formats.

MULT — Multiply (Unsigned)

MULT dst,src

Operation: dst ← dst × src

The 8-bit destination operand (even register of the register pair) is multiplied by the source operand (8 bits) and the product (16 bits) is stored in the register pair specified by the destination address. Both operands are treated as unsigned integers.

Flags:

C: Set if result is $>$ 255; cleared otherwise.
Z: Set if the result is "0"; cleared otherwise.
S: Set if MSB of the result is a "1"; cleared otherwise.
V: Cleared.
D: Unaffected.
H: Unaffected.

Format:

opc	src	dst

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**	**Addr Mode src**
opc | src | dst	3	22	84	RR	R
		22	85	RR	IR
		22	86	RR	IM

Examples: Given: Register 00H = 20H, register 01H = 03H, register 02H = 09H, register 03H = 06H:

MULT 00H, 02H → Register 00H = 01H, register 01H = 20H, register 02H = 09H
MULT 00H, @01H → Register 00H = 00H, register 01H = 0C0H
MULT 00H, #30H → Register 00H = 06H, register 01H = 00H

In the first example, the statement "MULT 00H,02H" multiplies the 8-bit destination operand (in the register 00H of the register pair 00H, 01H) by the source register 02H operand (09H). The 16-bit product, 0120H, is stored in the register pair 00H, 01H.

NEXT — Next

NEXT

Operation: PC ← @ IP

IP ← IP + 2

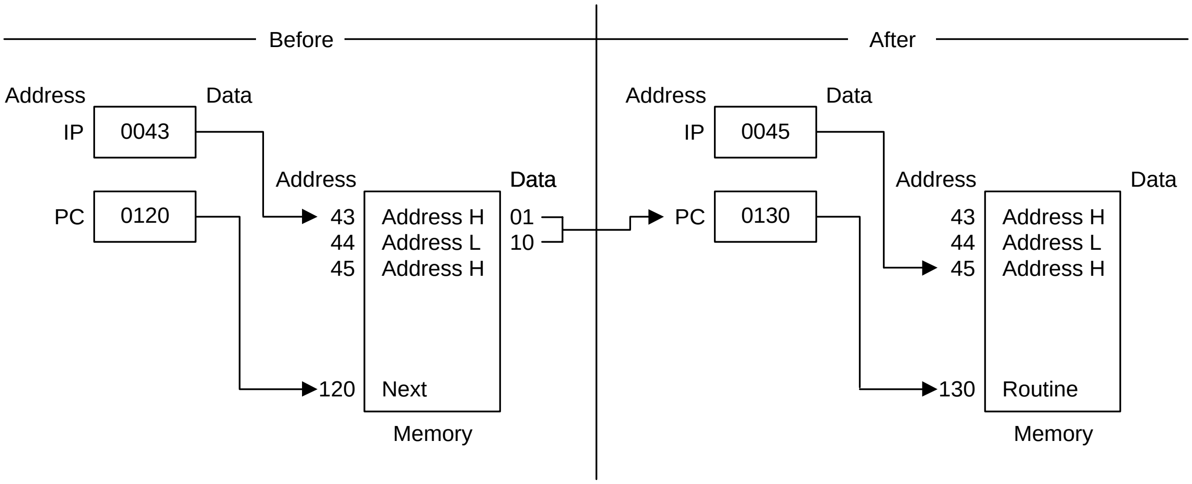
The NEXT instruction is useful when implementing threaded-code languages. The program memory word that is pointed to by the instruction pointer is loaded into the program counter. The instruction pointer is then incremented by two.

Flags: No flags are affected.

Format:

	Bytes	**Cycles**	**Opcode (Hex)**
opc	1	10	0F

Example: The following diagram shows one example of how to use the NEXT instruction.



NOP — No Operation

NOP

Operation: No action is performed when the CPU executes this instruction. Typically, one or more NOPs are executed in sequence in order to effect a timing delay of variable duration.

Flags: No flags are affected.

Format:

	Bytes	**Cycles**	**Opcode (Hex)**
opc	1	4	FF

Example: When the instruction

NOP

is encountered in a program, no operation occurs. Instead, there is a delay in instruction execution time.

OR — Logical OR

OR dst,src

Operation: dst ← dst OR src

The source operand is logically ORed with the destination operand and the result is stored in the destination. The contents of the source are unaffected. The OR operation results in a "1" being stored whenever either of the corresponding bits in the two operands is a "1"; otherwise a "0" is stored.

Flags:

C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Always cleared to "0".
D: Unaffected.
H: Unaffected.

Format:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode**	
				dst	**src**
opc | dst | src	2	4	42	r	r
		6	43	r	Ir
opc | src | dst	3	6	44	R	R
		6	45	R	IR
opc | dst | src	3	6	46	R	IM

Examples: Given: R0 = 15H, R1 = 2AH, R2 = 01H, register 00H = 08H, register 01H = 37H, and register 08H = 8AH:

OR	R0,R1	→	R0 = 3FH, R1 = 2AH
OR	R0,@R2	→	R0 = 37H, R2 = 01H, register 01H = 37H
OR	00H,01H	→	Register 00H = 3FH, register 01H = 37H
OR	01H,@00H	→	Register 00H = 08H, register 01H = 0BFH
OR	00H,#02H	→	Register 00H = 0AH

In the first example, if working register R0 contains the value 15H and register R1 the value 2AH, the statement "OR R0,R1" logical-ORs the R0 and R1 register contents and stores the result (3FH) in destination register R0.

The other examples show the use of the logical OR instruction with the various addressing modes and formats.

POP — Pop From Stack

POP dst

Operation: dst ← @SP

SP ← SP + 1

The contents of the location addressed by the stack pointer are loaded into the destination. The stack pointer is then incremented by one.

Flags: No flags affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
opc	dst	2	8	50	R
			8	51	IR

Examples: Given: Register 00H = 01H, register 01H = 1BH, SPH (0D8H) = 00H, SPL (0D9H) = 0FBH, and stack register 0FBH = 55H:

POP 00H → Register 00H = 55H, SP = 00FCH

POP @00H → Register 00H = 01H, register 01H = 55H, SP = 00FCH

In the first example, general register 00H contains the value 01H. The statement "POP 00H" loads the contents of location 00FBH (55H) into destination register 00H and then increments the stack pointer by one. Register 00H then contains the value 55H and the SP points to location 00FCH.

POPUD — Pop User Stack (Decrementing)

POPUD dst,src

Operation: dst ← src

IR ← IR – 1

This instruction is used for user-defined stacks in the register file. The contents of the register file location addressed by the user stack pointer are loaded into the destination. The user stack pointer is then decremented.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode	Addr Mode	
					(Hex)	dst	src
opc	src	dst	3	8	92	R	IR

Example: Given: Register 00H = 42H (user stack pointer register), register 42H = 6FH, and register 02H = 70H:

POPUD 02H,@00H → Register 00H = 41H, register 02H = 6FH, register 42H = 6FH

If general register 00H contains the value 42H and register 42H the value 6FH, the statement "POPUD 02H,@00H" loads the contents of register 42H into the destination register 02H. The user stack pointer is then decremented by one, leaving the value 41H.

POPUI — Pop User Stack (Incrementing)

POPUI dst,src

Operation: dst ← src

IR ← IR + 1

The POPUI instruction is used for user-defined stacks in the register file. The contents of the register file location addressed by the user stack pointer are loaded into the destination. The user stack pointer is then incremented.

Flags: No flags are affected.

Format:

			Bytes	**Cycles**	**Opcode**	**Addr Mode**	
					(Hex)	**dst**	**src**
opc	src	dst	3	8	93	R	IR

Example: Given: Register 00H = 01H and register 01H = 70H:

POPUI 02H,@00H → Register 00H = 02H, register 01H = 70H, register 02H = 70H

If general register 00H contains the value 01H and register 01H the value 70H, the statement "POPUI 02H,@00H" loads the value 70H into the destination general register 02H. The user stack pointer (register 00H) is then incremented by one, changing its value from 01H to 02H.

PUSH — Push To Stack

PUSH src

Operation: SP ← SP – 1

@SP ← src

A PUSH instruction decrements the stack pointer value and loads the contents of the source (src) into the location addressed by the decremented stack pointer. The operation then adds the new value to the top of the stack.

Flags: No flags are affected.

Format:

		Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**
opc	src	2	8 (internal clock)	70	R
			8 (external clock)		
			8 (internal clock)		
			8 (external clock)	71	IR

Examples: Given: Register 40H = 4FH, register 4FH = 0AAH, SPH = 00H, and SPL = 00H:

PUSH	40H	→	Register 40H = 4FH, stack register 0FFH = 4FH, SPH = 0FFH, SPL = 0FFH
PUSH	@40H	→	Register 40H = 4FH, register 4FH = 0AAH, stack register 0FFH = 0AAH, SPH = 0FFH, SPL = 0FFH

In the first example, if the stack pointer contains the value 0000H, and general register 40H the value 4FH, the statement "PUSH 40H" decrements the stack pointer from 0000 to 0FFFFH. It then loads the contents of register 40H into location 0FFFFH and adds this new value to the top of the stack.

PUSHUD — Push User Stack (Decrementing)

PUSHUD dst,src

Operation: IR ← IR – 1

dst ← src

This instruction is used to address user-defined stacks in the register file. PUSHUD decrements the user stack pointer and loads the contents of the source into the register addressed by the decremented stack pointer.

Flags: No flags are affected.

Format:

			Bytes	**Cycles**	**Opcode**	**Addr Mode**	
					(Hex)	**dst**	**src**
opc	dst	src	3	8	82	IR	R

Example: Given: Register 00H = 03H, register 01H = 05H, and register 02H = 1AH:

PUSHUD @00H,01H → Register 00H = 02H, register 01H = 05H, register 02H = 05H

If the user stack pointer (register 00H, for example) contains the value 03H, the statement "PUSHUD @00H,01H" decrements the user stack pointer by one, leaving the value 02H. The 01H register value, 05H, is then loaded into the register addressed by the decremented user stack pointer.

PUSHUI — Push User Stack (Incrementing)

PUSHUI dst,src

Operation: IR ← IR + 1

dst ← src

This instruction is used for user-defined stacks in the register file. PUSHUI increments the user stack pointer and then loads the contents of the source into the register location addressed by the incremented user stack pointer.

Flags: No flags are affected.

Format:

			Bytes	**Cycles**	**Opcode**	**Addr Mode**	
					(Hex)	**dst**	**src**
opc	dst	src	3	8	83	IR	R

Example: Given: Register 00H = 03H, register 01H = 05H, and register 04H = 2AH:

PUSHUI @00H,01H → Register 00H = 04H, register 01H = 05H, register 04H = 05H

If the user stack pointer (register 00H, for example) contains the value 03H, the statement "PUSHUI @00H,01H" increments the user stack pointer by one, leaving the value 04H. The 01H register value, 05H, is then loaded into the location addressed by the incremented user stack pointer.

RCF — Reset Carry Flag

RCF RCF

Operation: C ← 0

The carry flag is cleared to logic zero, regardless of its previous value.

Flags: **C:** Cleared to "0".

No other flags are affected.

Format:

	Bytes	**Cycles**	**Opcode (Hex)**
opc	1	4	CF

Example: Given: C = "1" or "0":

The instruction RCF clears the carry flag (C) to logic zero.

RET — Return

RET

Operation: PC ← @SP

SP ← SP + 2

The RET instruction is normally used to return to the previously executing procedure at the end of a procedure entered by a CALL instruction. The contents of the location addressed by the stack pointer are popped into the program counter. The next statement that is executed is the one that is addressed by the new program counter value.

Flags: No flags are affected.

Format:

	Bytes	**Cycles**	**Opcode (Hex)**
opc	1	8 (internal stack)	AF
		10 (internal stack)	

Example: Given: SP = 00FCH, (SP) = 101AH, and PC = 1234:

RET → PC = 101AH, SP = 00FEH

The statement "RET" pops the contents of stack pointer location 00FCH (10H) into the high byte of the program counter. The stack pointer then pops the value in location 00FEH (1AH) into the PC's low byte and the instruction at location 101AH is executed. The stack pointer now points to memory location 00FEH.

RL — Rotate Left

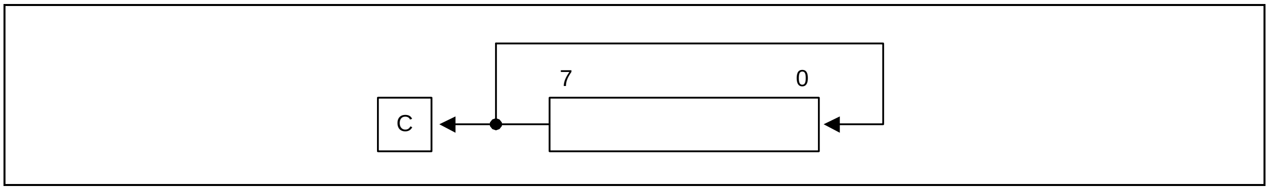
RL dst

Operation: C ← dst (7)

dst (0) ← dst (7)

dst (n + 1) ← dst (n), n = 0–6

The contents of the destination operand are rotated left one bit position. The initial value of bit 7 is moved to the bit zero (LSB) position and also replaces the carry flag.



Flags:

C: Set if the bit rotated from the most significant bit position (bit 7) was "1".
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Set if arithmetic overflow occurred; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode <u>dst</u>**
opc | dst	2	4	90	R
		4	91	IR

Examples: Given: Register 00H = 0AAH, register 01H = 02H and register 02H = 17H:

RL	00H	→	Register 00H = 55H, C = "1"
RL	@01H	→	Register 01H = 02H, register 02H = 2EH, C = "0"

In the first example, if general register 00H contains the value 0AAH (10101010B), the statement "RL 00H" rotates the 0AAH value left one bit position, leaving the new value 55H (01010101B) and setting the carry and overflow flags.

RLC — Rotate Left Through Carry

RLC dst

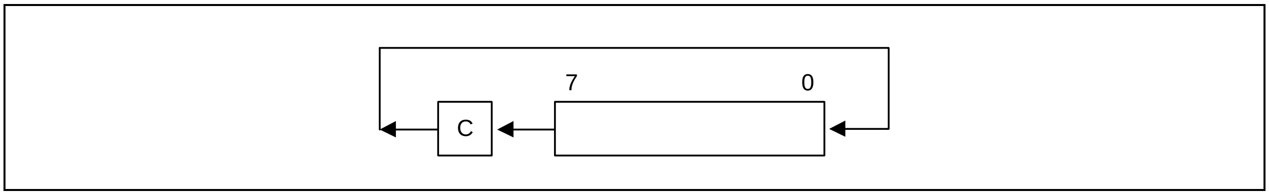
Operation:

dst (0) ← C

C ← dst (7)

dst (n + 1) ← dst (n), n = 0–6

The contents of the destination operand with the carry flag are rotated left one bit position. The initial value of bit 7 replaces the carry flag (C); the initial value of the carry flag replaces bit zero.



Flags:

C: Set if the bit rotated from the most significant bit position (bit 7) was "1".
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst	2	4	10	R
		4	11	IR

Examples:

Given: Register 00H = 0AAH, register 01H = 02H, and register 02H = 17H, C = "0":

RLC	00H	→	Register 00H = 54H, C = "1"
RLC	@01H	→	Register 01H = 02H, register 02H = 2EH, C = "0"

In the first example, if general register 00H has the value 0AAH (10101010B), the statement "RLC 00H" rotates 0AAH one bit position to the left. The initial value of bit 7 sets the carry flag and the initial value of the C flag replaces bit zero of register 00H, leaving the value 55H (01010101B). The MSB of register 00H resets the carry flag to "1" and sets the overflow flag.

RR — Rotate Right

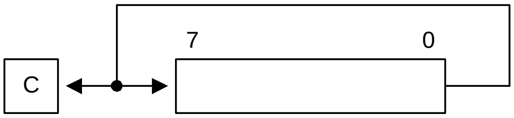
RR dst

Operation: C ← dst (0)

dst (7) ← dst (0)

dst (n) ← dst (n + 1), n = 0–6

The contents of the destination operand are rotated right one bit position. The initial value of bit zero (LSB) is moved to bit 7 (MSB) and also replaces the carry flag (C).



Flags:

C: Set if the bit rotated from the least significant bit position (bit zero) was "1".
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
opc | dst	2	4	E0	R
		4	E1	IR

Examples: Given: Register 00H = 31H, register 01H = 02H, and register 02H = 17H:

RR 00H → Register 00H = 98H, C = "1"

RR @01H → Register 01H = 02H, register 02H = 8BH, C = "1"

In the first example, if general register 00H contains the value 31H (00110001B), the statement "RR 00H" rotates this value one bit position to the right. The initial value of bit zero is moved to bit 7, leaving the new value 98H (10011000B) in the destination register. The initial bit zero also resets the C flag to "1" and the sign flag and overflow flag are also set to "1".

RRC — Rotate Right Through Carry

RRC dst

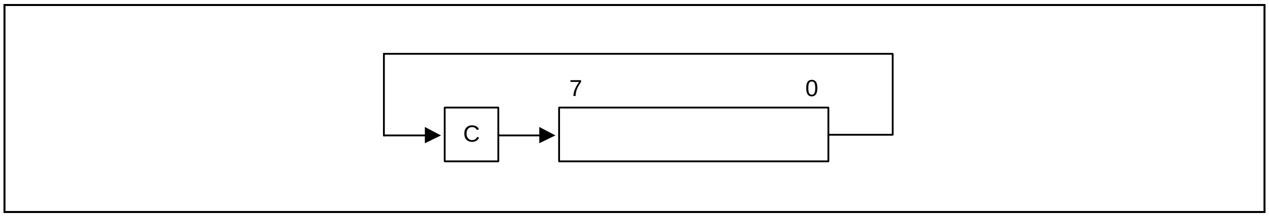
Operation:

dst (7) ← C

C ← dst (0)

dst (n) ← dst (n + 1), n = 0–6

The contents of the destination operand and the carry flag are rotated right one bit position. The initial value of bit zero (LSB) replaces the carry flag; the initial value of the carry flag replaces bit 7 (MSB).



Flags:

C: Set if the bit rotated from the least significant bit position (bit zero) was "1".
Z: Set if the result is "0" cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc	dst	2	4	C0	R
			4	C1	IR

Examples:

Given: Register 00H = 55H, register 01H = 02H, register 02H = 17H, and C = "0":

RRC 00H → Register 00H = 2AH, C = "1"
RRC @01H → Register 01H = 02H, register 02H = 0BH, C = "1"

In the first example, if general register 00H contains the value 55H (01010101B), the statement "RRC 00H" rotates this value one bit position to the right. The initial value of bit zero ("1") replaces the carry flag and the initial value of the C flag ("1") replaces bit 7. This leaves the new value 2AH (00101010B) in destination register 00H. The sign flag and overflow flag are both cleared to "0".

SB0 — Select Bank 0

SB0

Operation: BANK ← 0

The SB0 instruction clears the bank address flag in the FLAGS register (FLAGS.0) to logic zero, selecting bank 0 register addressing in the set 1 area of the register file.

Flags: No flags are affected.

Format:

	Bytes	**Cycles**	**Opcode**
			(Hex)
opc	1	4	4F

Example: The statement

SB0

clears FLAGS.0 to "0", selecting bank 0 register addressing.

SB1 — Select Bank 1

SB1

Operation: BANK ← 1

The SB1 instruction sets the bank address flag in the FLAGS register (FLAGS.0) to logic one, selecting bank 1 register addressing in the set 1 area of the register file. (Bank 1 is not implemented in some S3C8-series microcontrollers.)

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	5F

Example: The statement

SB1

sets FLAGS.0 to "1", selecting bank 1 register addressing, if implemented.

SBC — Subtract with Carry

SBC dst,src

Operation: dst ← dst − src − c

The source operand, along with the current value of the carry flag, is subtracted from the destination operand and the result is stored in the destination. The contents of the source are unaffected. Subtraction is performed by adding the two's-complement of the source operand to the destination operand. In multiple precision arithmetic, this instruction permits the carry ("borrow") from the subtraction of the low-order operands to be subtracted from the subtraction of high-order operands.

Flags:

C: Set if a borrow occurred (src > dst); cleared otherwise.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if the operands were of opposite sign and the sign of the result is the same as the sign of the source; cleared otherwise.
D: Always set to "1".
H: Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise, indicating a "borrow".

Format:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**	**Addr Mode src**
opc | dst | src	2	4	32	r	r
		6	33	r	lr
opc | src | dst	3	6	34	R	R
		6	35	R	IR
opc | dst | src	3	6	36	R	IM

Examples: Given: R1 = 10H, R2 = 03H, C = "1", register 01H = 20H, register 02H = 03H, and register 03H = 0AH:

SBC	R1,R2	→	R1 = 0CH, R2 = 03H
SBC	R1,@R2	→	R1 = 05H, R2 = 03H, register 03H = 0AH
SBC	01H,02H	→	Register 01H = 1CH, register 02H = 03H
SBC	01H,@02H	→	Register 01H = 15H,register 02H = 03H, register 03H = 0AH
SBC	01H,#8AH	→	Register 01H = 95H; C, S, and V = "1"

In the first example, if working register R1 contains the value 10H and register R2 the value 03H, the statement "SBC R1,R2" subtracts the source value (03H) and the C flag value ("1") from the destination (10H) and then stores the result (0CH) in register R1.

SCF — Set Carry Flag

SCF

Operation: C ← 1

The carry flag (C) is set to logic one, regardless of its previous value.

Flags: **C:** Set to "1".

No other flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	DF

Example: The statement

SCF

sets the carry flag to logic one.

SRA — Shift Right Arithmetic

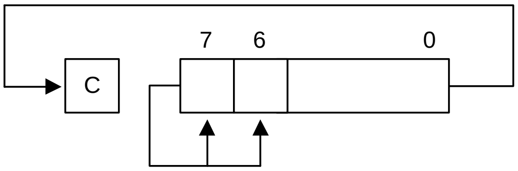
SRA dst

Operation: dst (7) ← dst (7)

C ← dst (0)

dst (n) ← dst (n + 1), n = 0–6

An arithmetic shift-right of one bit position is performed on the destination operand. Bit zero (the LSB) replaces the carry flag. The value of bit 7 (the sign bit) is unchanged and is shifted into bit position 6.



Flags:

C: Set if the bit shifted from the LSB position (bit zero) was "1".
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
V: Always cleared to "0".
D: Unaffected.
H: Unaffected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst	2	4	D0	R
		4	D1	IR

Examples: Given: Register 00H = 9AH, register 02H = 03H, register 03H = 0BCH, and C = "1":

SRA 00H → Register 00H = 0CD, C = "0"

SRA @02H → Register 02H = 03H, register 03H = 0DEH, C = "0"

In the first example, if general register 00H contains the value 9AH (10011010B), the statement "SRA 00H" shifts the bit values in register 00H right one bit position. Bit zero ("0") clears the C flag and bit 7 ("1") is then shifted into the bit 6 position (bit 7 remains unchanged). This leaves the value 0CDH (11001101B) in destination register 00H.

SRP/SRP0/SRP1 — Set Register Pointer

SRP src

SRP0 src

SRP1 src

Operation:

If src (1) = 1 and src (0) = 0 then:	RP0 (3–7)	←	src (3–7)
If src (1) = 0 and src (0) = 1 then:	RP1 (3–7)	←	src (3–7)
If src (1) = 0 and src (0) = 0 then:	RP0 (4–7)	←	src (4–7),
	RP0 (3)	←	0
	RP1 (4–7)	←	src (4–7),
	RP1 (3)	←	1

The source data bits one and zero (LSB) determine whether to write one or both of the register pointers, RP0 and RP1. Bits 3–7 of the selected register pointer are written unless both register pointers are selected. RP0.3 is then cleared to logic zero and RP1.3 is set to logic one.

Flags: No flags are affected.

Format:

		Bytes	**Cycles**	**Opcode**	**Addr Mode**
				(Hex)	**src**
opc	src	2	4	31	IM

Examples: The statement

SRP #40H

sets register pointer 0 (RP0) at location 0D6H to 40H and register pointer 1 (RP1) at location 0D7H to 48H.

The statement "SRP0 #50H" sets RP0 to 50H, and the statement "SRP1 #68H" sets RP1 to 68H.

STOP — Stop Operation

STOP

Operation:

The STOP instruction stops the both the CPU clock and system clock and causes the microcontroller to enter Stop mode. During Stop mode, the contents of on-chip CPU registers, peripheral registers, and I/O port control and data registers are retained. Stop mode can be released by an external reset operation or by external interrupts. For the reset operation, the RESET pin must be held to Low level until the required oscillation stabilization interval has elapsed.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode	Addr Mode	
			(Hex)	dst	src
opc	1	4	7F	–	–

Example: The statement

STOP

halts all microcontroller operations.

SUB — Subtract

SUB dst,src

Operation: dst ← dst – src

The source operand is subtracted from the destination operand and the result is stored in the destination. The contents of the source are unaffected. Subtraction is performed by adding the two's complement of the source operand to the destination operand.

Flags:

C: Set if a "borrow" occurred; cleared otherwise.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if the operands were of opposite signs and the sign of the result is of the same as the sign of the source operand; cleared otherwise.
D: Always set to "1".
H: Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise indicating a "borrow".

Format:

	Bytes	**Cycles**	**Opcode**	**Addr Mode**	
			(Hex)	**dst**	**src**
opc, dst | src	2	4	22	r	r
		6	23	r	Ir
opc, src, dst	3	6	24	R	R
		6	25	R	IR
opc, dst, src	3	6	26	R	IM

Examples: Given: R1 = 12H, R2 = 03H, register 01H = 21H, register 02H = 03H, register 03H = 0AH:

SUB	R1,R2	→	R1 = 0FH, R2 = 03H
SUB	R1,@R2	→	R1 = 08H, R2 = 03H
SUB	01H,02H	→	Register 01H = 1EH, register 02H = 03H
SUB	01H,@02H	→	Register 01H = 17H, register 02H = 03H
SUB	01H,#90H	→	Register 01H = 91H; C, S, and V = "1"
SUB	01H,#65H	→	Register 01H = 0BCH; C and S = "1", V = "0"

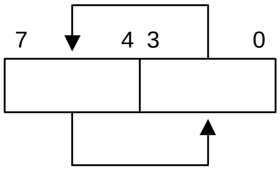
In the first example, if working register R1 contains the value 12H and if register R2 contains the value 03H, the statement "SUB R1,R2" subtracts the source value (03H) from the destination value (12H) and stores the result (0FH) in destination register R1.

SWAP — Swap Nibbles

SWAP dst

Operation: dst (0 – 3) ↔ dst (4 – 7)

The contents of the lower four bits and upper four bits of the destination operand are swapped.



Flags:

C: Undefined.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Undefined.
D: Unaffected.
H: Unaffected.

Format:

opc	dst

Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
2	4	F0	R
	4	F1	IR

Examples: Given: Register 00H = 3EH, register 02H = 03H, and register 03H = 0A4H:

SWAP	00H	→	Register 00H = 0E3H
SWAP	@02H	→	Register 02H = 03H, register 03H = 4AH

In the first example, if general register 00H contains the value 3EH (00111110B), the statement "SWAP 00H" swaps the lower and upper four bits (nibbles) in the 00H register, leaving the value 0E3H (11100011B).

TCM — Test Complement Under Mask

TCM dst,src

Operation: (NOT dst) AND src

This instruction tests selected bits in the destination operand for a logic one value. The bits to be tested are specified by setting a "1" bit in the corresponding position of the source operand (mask). The TCM statement complements the destination operand, which is then ANDed with the source mask. The zero (Z) flag can then be checked to determine the result. The destination and source operands are unaffected.

Flags:

C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Always cleared to "0".
D: Unaffected.
H: Unaffected.

Format:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode dst**	**src**
opc, dst | src	2	4	62	r	r
		6	63	r	lr
opc, src, dst	3	6	64	R	R
		6	65	R	IR
opc, dst, src	3	6	66	R	IM

Examples: Given: R0 = 0C7H, R1 = 02H, R2 = 12H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

TCM	R0,R1	→	R0 = 0C7H, R1 = 02H, Z = "1"
TCM	R0,@R1	→	R0 = 0C7H, R1 = 02H, register 02H = 23H, Z = "0"
TCM	00H,01H	→	Register 00H = 2BH, register 01H = 02H, Z = "1"
TCM	00H,@01H	→	Register 00H = 2BH, register 01H = 02H, register 02H = 23H, Z = "1"
TCM	00H,#34	→	Register 00H = 2BH, Z = "0"

In the first example, if working register R0 contains the value 0C7H (11000111B) and register R1 the value 02H (00000010B), the statement "TCM R0,R1" tests bit one in the destination register for a "1" value. Because the mask value corresponds to the test bit, the Z flag is set to logic one and can be tested to determine the result of the TCM operation.

TM — Test Under Mask

TM dst,src

Operation: dst AND src

This instruction tests selected bits in the destination operand for a logic zero value. The bits to be tested are specified by setting a "1" bit in the corresponding position of the source operand (mask), which is ANDed with the destination operand. The zero (Z) flag can then be checked to determine the result. The destination and source operands are unaffected.

Flags:

C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Always reset to "0".
D: Unaffected.
H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	dst | src		2	4	72	r	r
				6	73	r	Ir
opc	src	dst	3	6	74	R	R
				6	75	R	IR
opc	dst	src	3	6	76	R	IM

Examples: Given: R0 = 0C7H, R1 = 02H, R2 = 18H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

TM	R0,R1	→	R0 = 0C7H, R1 = 02H, Z = "0"
TM	R0,@R1	→	R0 = 0C7H, R1 = 02H, register 02H = 23H, Z = "0"
TM	00H,01H	→	Register 00H = 2BH, register 01H = 02H, Z = "0"
TM	00H,@01H	→	Register 00H = 2BH, register 01H = 02H, register 02H = 23H, Z = "0"
TM	00H,#54H	→	Register 00H = 2BH, Z = "1"

In the first example, if working register R0 contains the value 0C7H (11000111B) and register R1 the value 02H (00000010B), the statement "TM R0,R1" tests bit one in the destination register for a "0" value. Because the mask value does not match the test bit, the Z flag is cleared to logic zero and can be tested to determine the result of the TM operation.

WFI — Wait for Interrupt

WFI

Operation:

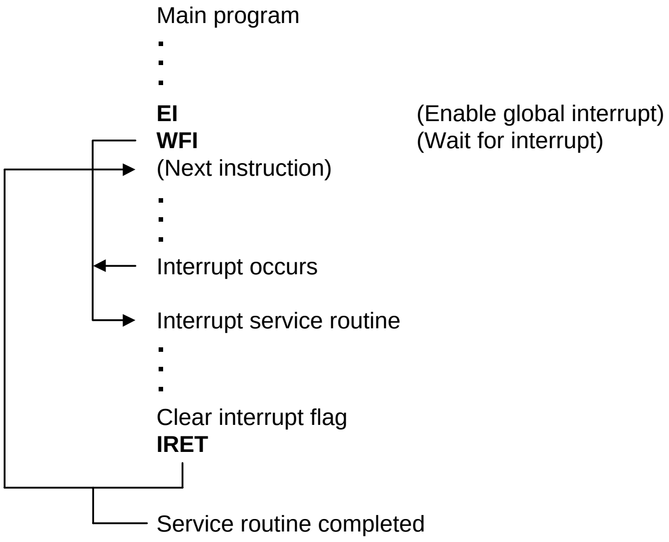
The CPU is effectively halted until an interrupt occurs, except that DMA transfers can still take place during this wait state. The WFI status can be released by an internal interrupt, including a fast interrupt .

Flags: No flags are affected.

Format:

	Bytes	**Cycles**	**Opcode (Hex)**
opc	1	4n (n = 1, 2, 3, ...)	3F

Example: The following sample program structure shows the sequence of operations that follow a "WFI" statement:



XOR — Logical Exclusive OR

XOR dst,src

Operation: dst ← dst XOR src

The source operand is logically exclusive-ORed with the destination operand and the result is stored in the destination. The exclusive-OR operation results in a "1" bit being stored whenever the corresponding bits in the operands are different; otherwise, a "0" bit is stored.

Flags:

C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Always reset to "0".
D: Unaffected.
H: Unaffected.

Format:

	Bytes	**Cycles**	**Opcode (Hex)**	**Addr Mode**	
				dst	**src**
opc | dst | src	2	4	B2	r	r
		6	B3	r	lr
opc | src | dst	3	6	B4	R	R
		6	B5	R	IR
opc | dst | src	3	6	B6	R	IM

Examples: Given: R0 = 0C7H, R1 = 02H, R2 = 18H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

XOR	R0,R1	→	R0 = 0C5H, R1 = 02H
XOR	R0,@R1	→	R0 = 0E4H, R1 = 02H, register 02H = 23H
XOR	00H,01H	→	Register 00H = 29H, register 01H = 02H
XOR	00H,@01H	→	Register 00H = 08H, register 01H = 02H, register 02H = 23H
XOR	00H,#54H	→	Register 00H = 7FH

In the first example, if working register R0 contains the value 0C7H and if register R1 contains the value 02H, the statement "XOR R0,R1" logically exclusive-ORs the R1 value with the R0 value and stores the result (0C5H) in the destination register R0.

NOTES

7 CLOCK CIRCUIT

OVERVIEW

The clock frequency generated for the S3C80M4/F80M4 by an external crystal can range from 0.4 MHz to 10 MHz. The maximum CPU clock frequency is 10 MHz. The X_{IN} and X_{OUT} pins connect the external oscillator or clock source to the on-chip clock circuit.

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 (fxx divided by 1, 2, 8, or 16)

— System clock control register, CLKCON

— Clock output control register, CLOCON

— STOP control register, STPCON

CPU CLOCK NOTATION

In this document, the following notation is used for descriptions of the CPU clock;

fx: main clock

fxx: selected system clock

MAIN OSCILLATOR CIRCUITS

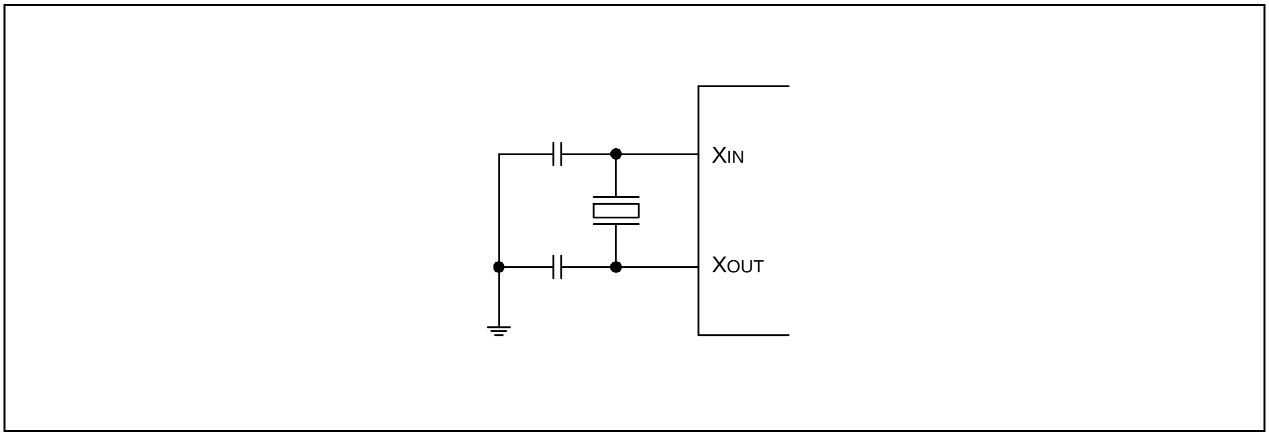


Figure 7-1. Crystal/Ceramic Oscillator (fx)

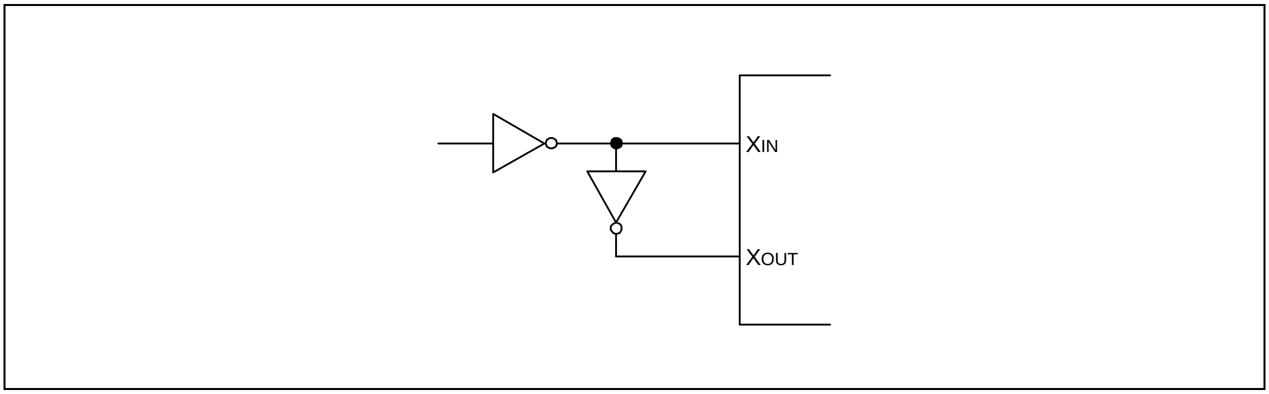


Figure 7-2. External Oscillator (fx)

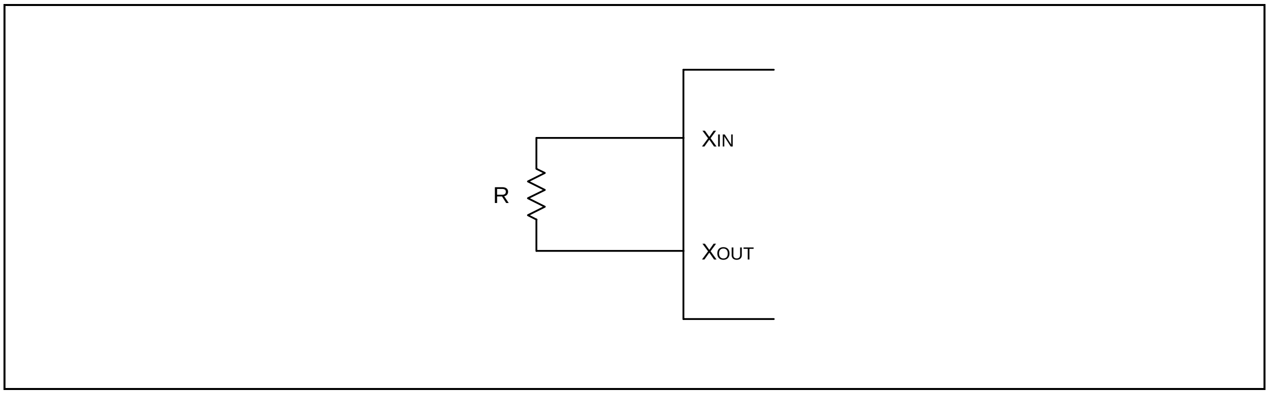


Figure 7-3. RC Oscillator (fx)

CLOCK STATUS DURING POWER-DOWN MODES

The two power-down modes, Stop mode and Idle mode, affect the system clock as follows:

— In Stop mode, the main oscillator is halted. Stop mode is released, and the oscillator is started, by a reset operation or an external interrupt (with RC delay noise filter), and can be released by internal interrupt too when the sub-system oscillator is running and watch timer is operating with sub-system clock.

— In Idle mode, the internal clock signal is gated to the CPU, but not to interrupt structure, timers and timer/ counters. Idle mode is released by a reset or by an external or internal interrupt.

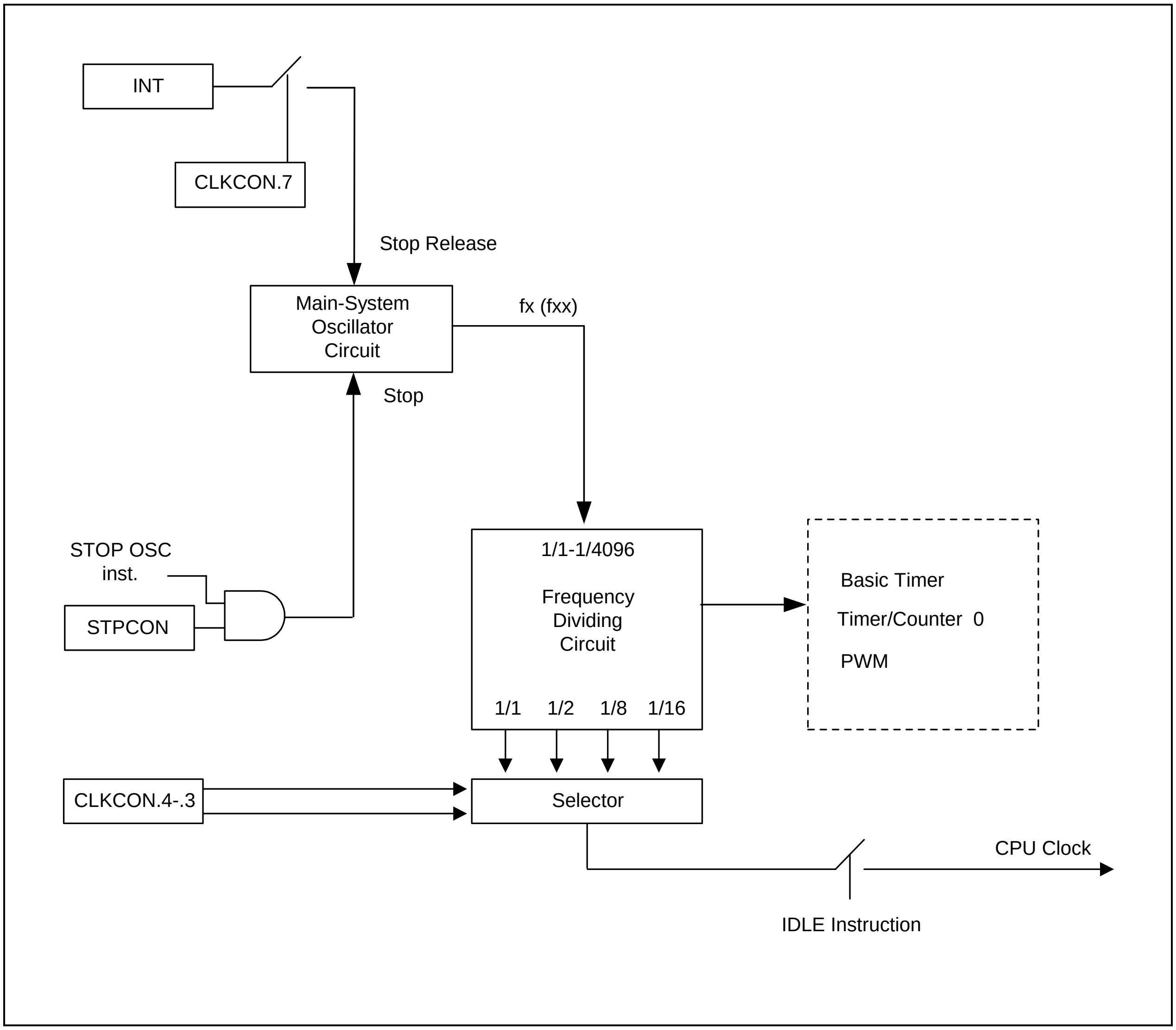


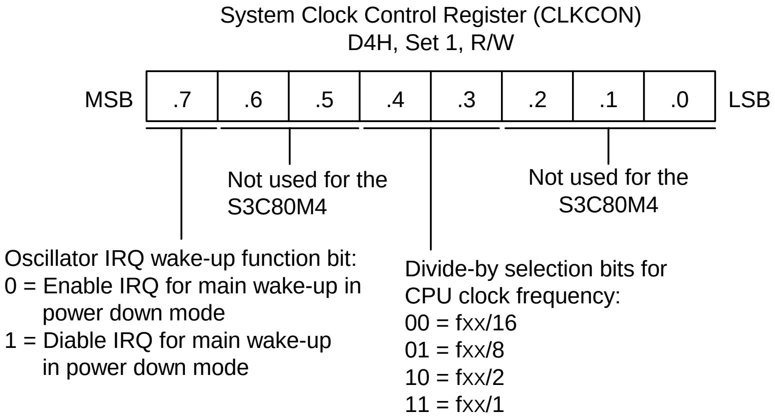
Figure 7-4. System Clock Circuit Diagram

SYSTEM CLOCK CONTROL REGISTER (CLKCON)

The system clock control register, CLKCON, is located in the set 1, address D4H. It is read/write addressable and has the following functions:

— Oscillator frequency divide-by value

After the main oscillator is activated, and the fxx/16 (the slowest clock speed) is selected as the CPU clock. If necessary, you can then increase the CPU clock speed fxx/8, fxx/2, or fxx/1.


System Clock Control Register (CLKCON)
D4H, Set 1, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Not used for the S3C80M4
Not used for the S3C80M4
Oscillator IRQ wake-up function bit:
0 = Enable IRQ for main wake-up in power down mode
1 = Diable IRQ for main wake-up in power down mode
Divide-by selection bits for CPU clock frequency:
00 = fxx/16
01 = fxx/8
10 = fxx/2
11 = fxx/1

NOTE: After a reset, the slowest clock (divided by 16) is selected as the system clock. To select faster speeds, load the appropriate values to CLKCON.3 and CLKCON.4.

Figure 7-5. System Clock Control Register (CLKCON)

CLOCK OUTPUT CONTROL REGISTER (CLOCON)

The clock output control register, CLOCON, is located in the bank 0 of set1, address E3H. It is read/write addressable and has the following functions;

— Clock Output Frequency Selection

After a reset, fxx/64 is select for Clock Output Frequency because the reset value of CLOCON.1-.0 is "0".

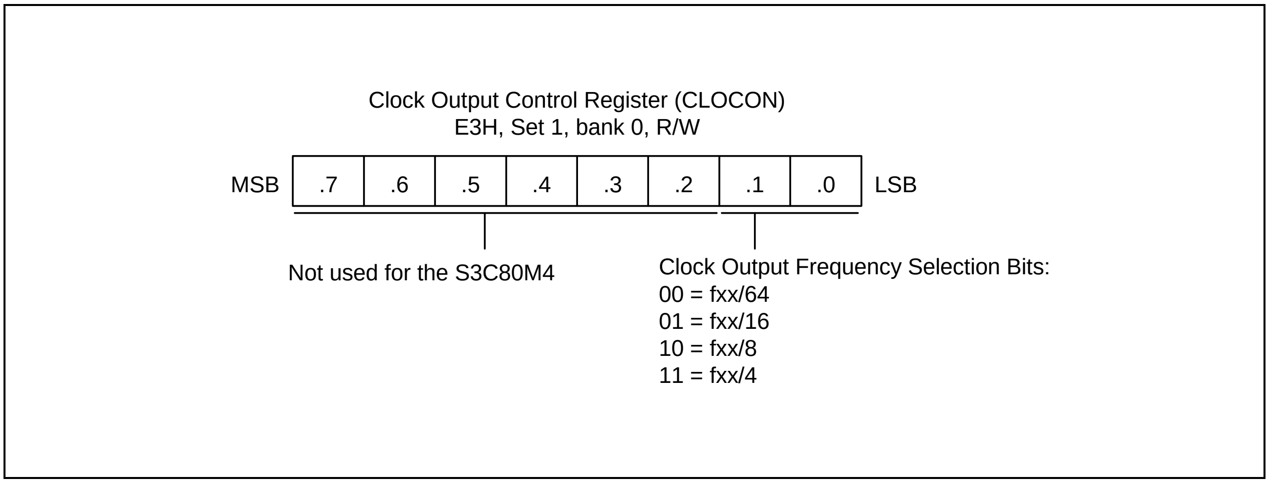


Figure 7-6. Clock Output Control Register (CLOCON)

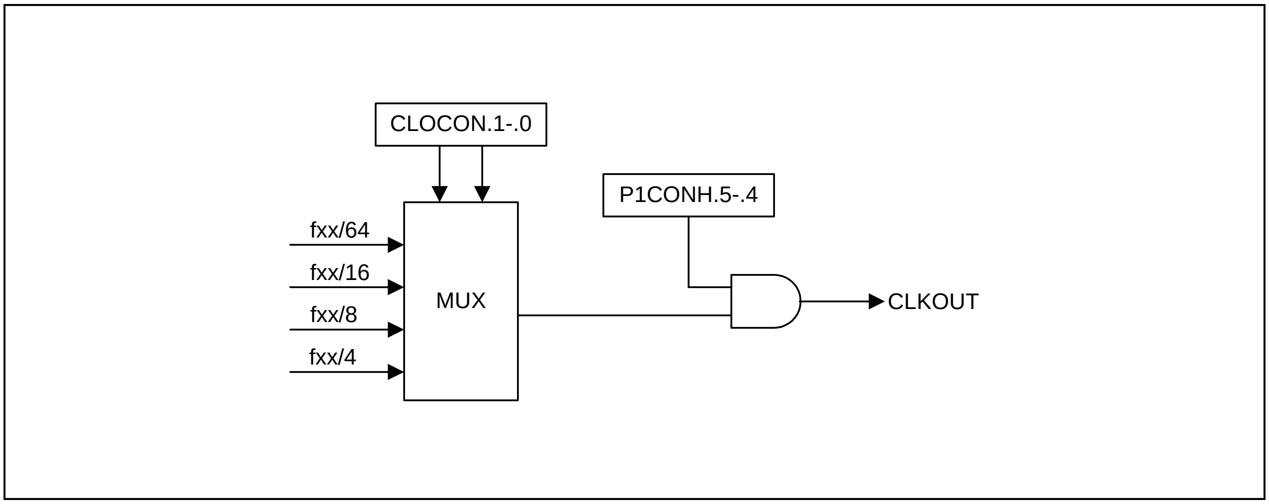


Figure 7-7. Clock Output Block Diagram

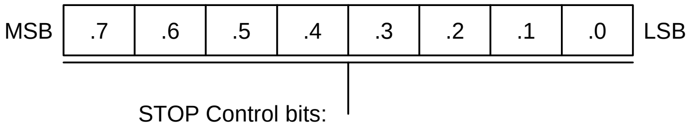
STOP CONTROL REGISTER (STPCON)

The STOP control register, STPCON, is located in the bank 0 of set1, address FBH. It is read/write addressable and has the following functions:

— Enable/Disable STOP instruction

After a reset, the STOP instruction is disabled, because the value of STPCON is "other values".
If necessary, you can use the STOP instruction by setting the value of STPCON to "10100101B".

STOP Control Register (STPCON)
FBH, Set 1,bank 0, R/W



STOP Control bits:
Other values = Disable STOP instruction
10100101 = Enable STOP instruction

NOTE: Before executing the STOP instruction, set the STPCON register as "10100101b". Otherwise the STOP instruction will not be executed and reset will be generated.

Figure 7-8. STOP Control Register (STPCON)

PROGRAMMING TIP — How to Use Stop Instruction

This example shows how to go STOP mode when a main clock is selected as the system clock.

```
LD      STOPCON,#1010010B     ;  Enable STOP instruction
STOP                          ;  Enter STOP mode
NOP
NOP
NOP                           ;  Release STOP mode
LD      STOPCON,#00000000B    ;  Disable STOP instruction
```

8 RESET and POWER-DOWN

SYSTEM RESET

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 the S3C80M4/F80M4 into a known operating status.

To allow time for internal 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. The minimum required time of a reset operation for oscillation stabilization is 1 millisecond.

Whenever a reset occurs during normal operation (that is, when both V_{DD} and RESET are High level), the nRESET pin is forced Low level and the reset operation starts. All system and peripheral control registers are then reset to their default hardware values

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

— All interrupt is disabled.

— The watchdog function (basic timer) is enabled.

— Ports 0-1 and set to input mode, and all pull-up resistors are disabled for the I/O port.

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

— The program counter (PC) is loaded with the program reset address in the ROM, 0100H.

— When the programmed oscillation stabilization time interval has elapsed, the instruction stored in ROM location 0100H (and 0101H) is fetched and executed at normal mode by smart option.

NORMAL MODE RESET OPERATION

A reset enables access to the S3C80M4 (4Kbyte) on-chip ROM. (The external interface is not automatically configured).

NOTE

To program the duration of the oscillation stabilization interval, you make the appropriate settings to the basic timer control register, BTCON, *before* entering Stop mode. Also, if you do not want to use the basic timer watchdog function (which causes a system reset if a basic timer counter overflow occurs), you can disable it by writing "1010B" to the upper nibble of BTCON.

HARDWARE RESET VALUES

Table 8-1, 8-2 list the reset values for CPU and system registers, peripheral control registers, and peripheral data registers following a reset operation. The following notation is used to represent reset values:

— A "1" or a "0" shows the reset bit value as logic one or logic zero, respectively.

— An "x" means that the bit value is undefined after a reset.

— A dash ("–") means that the bit is either not used or not mapped, but read 0 is the bit value.

Table 8-1. S3C80M4/F80M4 Set 1 Register and Values After RESET

Register Name	Mnemonic	Address		Bit Values After RESET							
		Dec	Hex	7	6	5	4	3	2	1	0
Locations D0H-D2H are not mapped.											
Basic timer control register	BTCON	211	D3H	0	0	0	0	0	0	0	0
System clock control register	CLKCON	212	D4H	0	–	–	0	0	–	–	–
System flags register	FLAGS	213	D5H	x	x	x	x	x	x	0	0
Register pointer 0	RP0	214	D6H	1	1	0	0	0	–	–	–
Register pointer 1	RP1	215	D7H	1	1	0	0	1	–	–	–
Stack pointer (high byte)	SPH	216	D8H	x	x	x	x	x	x	x	x
Stack pointer (low byte)	SPL	217	D9H	x	x	x	x	x	x	x	x
Instruction pointer (high byte)	IPH	218	DAH	x	x	x	x	x	x	x	x
Instruction pointer (low byte)	IPL	219	DBH	x	x	x	x	x	x	x	x
Interrupt request register	IRQ	220	DCH	0	0	0	0	0	0	0	0
Interrupt mask register	IMR	221	DDH	x	x	x	x	x	x	x	x
System mode register	SYM	222	DEH	0	–	–	x	x	x	0	0
Register page pointer	PP	223	DFH	0	0	0	0	0	0	0	0

Table 8-2. S3C80M4/F80M4 Set 1, Bank 0 Register and Values After RESET

Register Name	Mnemonic	Address		Bit Values After RESET							
		Dec	Hex	7	6	5	4	3	2	1	0
Port 0 Data Register	P0	224	E0H	0	0	0	0	0	0	0	0
Port 1 Data Register	P1	225	E1H	0	0	0	0	0	0	0	0
Location E2H is not mapped.											
Clock Output Control Register	CLOCON	227	E3H	–	–	–	–	–	–	0	0
Timer 0 Counter Register	T0CNT	228	E4H	0	0	0	0	0	0	0	0
Timer 0 Data Register	T0DATA	229	E5H	1	1	1	1	1	1	1	1
Timer 0 Control Register	T0CNT	230	E6H	0	0	0	0	0	0	0	0
PWM Data Register	PWMDATA	231	E7H	0	0	0	0	0	0	0	0
PWM Control Register	PWMCON	232	E8H	0	0	–	0	0	0	0	0
Locations E9H-EEH are not mapped.											
Port 1 Control Register (High Byte)	P1CONH	240	EFH	–	–	0	0	0	0	0	0
Port 1 Control Register (Low Byte)	P1CONL	241	F0H	0	0	0	0	0	0	0	0
Port 1 Pull-up Resistor Enable Register	P1PUR	242	F1H	–	1	1	1	0	0	0	0
Port 0 Control Register (High Byte)	P0CONH	243	F2H	0	1	0	0	0	0	0	0
Port 0 Control Register (Low Byte)	P0CONL	244	F3H	0	0	0	0	0	0	0	0
Port 0 Interrupt Control Register	P0INT	245	F4H	0	0	0	0	0	0	0	0
Port 0 Interrupt Pending Register	P0PND	246	F5H	0	0	0	0	0	0	0	0
Locations F6H-FAH are not mapped.											
STOP control register	STPCON	251	FBH	0	0	0	0	0	0	0	0
Location FCH is not mapped.											
Basic Timer Counter	BTCNT	253	FDH	0	0	0	0	0	0	0	0
Location FEH is not mapped.											
Interrupt Priority Register	IPR	255	FFH	x	x	x	x	x	x	x	x

POWER-DOWN MODES

STOP MODE

Stop mode is invoked by the instruction STOP (opcode 7FH). In Stop mode, the operation of the CPU and all peripherals is halted. That is, the on-chip main oscillator stops and the supply current is reduced to less than 3μA. All system functions stop when the clock “freezes”, but data stored in the internal register file is retained. Stop mode can be released in one of two ways: by a reset or by interrupts, for more details see Figure 7-4.

NOTE

> Do not use stop mode if you are using an external clock source because X_{IN} input must be restricted internally to V_{SS} to reduce current leakage.

Using nRESET to Release Stop Mode

Stop mode is released when the nRESET signal is released and returns to high level: all system and peripheral control registers are reset to their default hardware values and the contents of all data registers are retained. A reset operation automatically selects a slow clock fxx/16 because CLKCON.3 and CLKCON.4 are cleared to ‘00B’. After the programmed oscillation stabilization interval has elapsed, the CPU starts the system initialization routine by fetching the program instruction stored in ROM location 0100H (and 0101H)

Using an External Interrupt to Release Stop Mode

External interrupts with an RC-delay noise filter circuit can be used to release Stop mode. Which interrupt you can use to release Stop mode in a given situation depends on the microcontroller’s current internal operating mode. The external interrupts in the S3C80M4/F80M4 interrupt structure that can be used to release Stop mode are:

— External interrupts P0.0–P0.3 (INT0–INT3)

Please note the following conditions for Stop mode release:

— If you release Stop mode using an external interrupt, the current values in system and peripheral control registers are unchanged except STPCON register.

— If you use an internal or external interrupt for Stop mode release, you can also program the duration of the oscillation stabilization interval. To do this, you must make the appropriate control and clock settings *before* entering Stop mode.

— When the Stop mode is released by external interrupt, the CLKCON.4 and CLKCON.3 bit-pair setting remains unchanged and the currently selected clock value is used.

— The external interrupt is serviced when the Stop mode release occurs. Following the IRET from the service routine, the instruction immediately following the one that initiated Stop mode is executed.

Using an Internal Interrupt to Release Stop Mode

Activate any enabled interrupt, causing Stop mode to be released. Other things are same as using external interrupt.

How to Enter into Stop Mode

Handling STPCON register then writing STOP instruction (keep the order).

```
LD  STPCON,#10100101B
STOP
NOP
NOP
NOP
```

IDLE MODE

Idle mode is invoked by the instruction IDLE (opcode 6FH). In idle mode, CPU operations are halted while some peripherals remain active. During idle mode, the internal clock signal is gated away from the CPU, but all peripherals timers remain active. Port pins retain the mode (input or output) they had at the time idle mode was entered.

There are two ways to release idle mode:

1. Execute a reset. All system and peripheral control registers are reset to their default values and the contents of all data registers are retained. The reset automatically selects the slow clock fxx/16 because CLKCON.4 and CLKCON.3 are cleared to ‘00B’. If interrupts are masked, a reset is the only way to release idle mode.
2. Activate any enabled interrupt, causing idle mode to be released. When you use an interrupt to release idle mode, the CLKCON.4 and CLKCON.3 register values remain unchanged, and the currently selected clock value is used. The interrupt is then serviced. When the return-from-interrupt (IRET) occurs, the instruction immediately following the one that initiated idle mode is executed.

NOTES

9 I/O PORTS

OVERVIEW

The S3C80M4/F80M4 microcontroller has two bit-programmable I/O ports, P0–P1. The port 0 is a 8-bit port, the port 1 is a 7-bit port. This gives a total of 15 I/O pins. Each port can be flexibly configured to meet application design requirements. The CPU accesses ports by directly writing or reading port registers. No special I/O instructions are required.

Table 9-1 gives you a general overview of the S3C80M4/F80M4 I/O port functions.

Table 9-1. S3C80M4/F80M4 Port Configuration Overview

Port	Configuration Options
0	1-bit programmable I/O port. Schmitt trigger input or push-pull output mode selected by software; software assignable pull-ups. P0.0–P0.3 can be used as inputs for external interrupts INT0–INT3 (with interrupt enable and pending control). Alternately P0.6 can be used as PWM.
1	1-bit programmable I/O port. Input or push-pull, open-drain output mode selected by software; software assignable pull-ups. Alternately P1.0, P1.0, P1.6 can be used as T0OUT, T0CLK, CLKOUT.

PORT DATA REGISTERS

Table 9-2 gives you an overview of the register locations of all four S3C80M4/F80M4 I/O port data registers. Data registers for ports 0 and 1 have the general format shown in Figure 9-1.

Table 9-2. Port Data Register Summary

Register Name	Mnemonic	Decimal	Hex	Location	R/W
Port 0 data register	P0	224	E0H	Set 1, Bank 0	R/W
Port 1 data register	P1	225	E1H	Set 1, Bank 0	R/W

PORT 0

Port 0 is an 8-bit I/O port with individually configurable pins. Port 0 pins are accessed directly by writing or reading the port 0 data register, P0 at location E0H in set 1, bank 0. P0.0–P0.7 can serve inputs, as output push pull or you can configure the following alternative functions:

— Low-byte pins (P0.0–P0.3): INT0–INT3

— High-byte pins (P0.4–P0.7): PWM

Port 0 Control Register (P0CONH, P0CONL)

Port 0 has two 8-bit control registers: P0CONH for P0.4-P0.7 and P0CONL for P0.0-P0.3. A reset clears the P0CONH and P0CONL registers to "40H" and "00H", configuring all pins to input mode. In input mode, three different selections are available:

— Schmitt trigger input with interrupt generation on falling signal edges.

— Schmitt trigger input with interrupt generation on rising signal edges.

— Schmitt trigger input with interrupt generation on falling/rising signal edges.

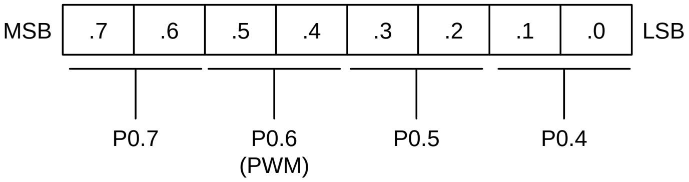
Port 0 Interrupt Enable and Pending Registers (P0INT)

To process external interrupts at the port 0 pins, the additional control registers are provided: the port 0 interrupt enable register P0INT (F4H, set 1, bank 0) and the port 0 interrupt pending register P0PND (F5H, set 1, bank 0).

The port 0 interrupt pending register P0PND lets you check for interrupt pending conditions and clear the pending condition when the interrupt service routine has been initiated. The application program detects interrupt requests by polling the P0PND register at regular intervals.

When the interrupt enable bit of any port 0 pin is “1”, a rising or falling signal edge at that pin will generate an interrupt request. The corresponding P0PND bit is then automatically set to “1” and the IRQ level goes low to signal the CPU that an interrupt request is waiting. When the CPU acknowledges the interrupt request, application software must the clear the pending condition by writing a “0” to the corresponding P0PND bit.

Port 0 Control Register, High Byte (P0CONH)
F2H, Set 1, Bank 0, R/W

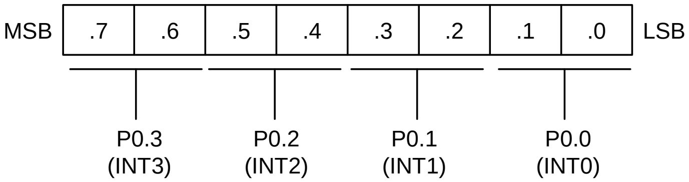


P0CONH bit-pair pin configuration settings:

00	Schmitt trigger input mode
01	Schmitt trigger input mode, pull-up
10	Alternative function (PWM,not used for P0.7/P0.5/P0.4)
11	Output mode, push-pull

Figure 9-1. Port 0 High-Byte Control Register (P0CONH)

Port 0 Control Register, Low Byte (P0CONL)
F3H, Set 1, Bank 0, R/W

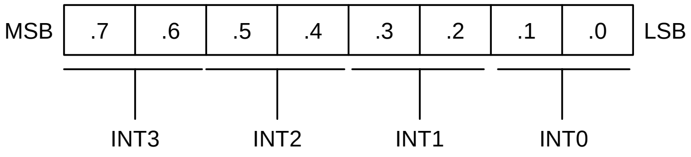


P0CONL bit-pair pin configuration settings:

00	Schmitt trigger input mode
01	Schmitt trigger input mode, pull-up
10	Not available
11	Output mode, push-pull

Figure 9-2. Port 0 Low-Byte Control Register (P0CONL)

Port 0 Interrupt Control Register (P0INT)
F4H, Set 1, Bank 0, R/W

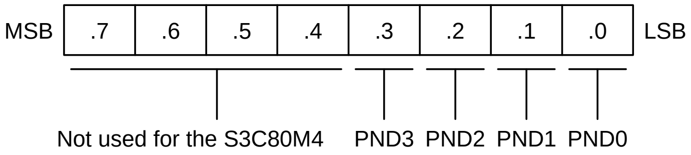


P0INT bit configuration settings:

00	Disable interrupt
01	Enable interrupt by falling edge
10	Enable interrupt by rising edge
11	Enable interrupt by both falling and rising edge

Figure 9-3. Port 0 Interrupt Control Register

Port 0 Interrupt Pending Register (P0PND)
F5H, Set 1, Bank 0, R/W



P0PND bit configuration settings:

0	Interrupt request is not pending, pending bit clear when write 0
1	Interrupt request is pending

Figure 9-4. Port 0 Interrupt Pending Register (P0PND)

PORT 1

Port 1 is an 7-bit I/O port with individually configurable pins. Port 1 pins are accessed directly by writing or reading the port 1 data register, P1 at location E1H in set 1, bank 0. P1.0–P1.6 can serve inputs, as outputs (push pull or open-drain) or you can configure the following alternative functions:

— Low-byte pins (P1.0-P1.3): T0OUT, T0CLK

— High-byte pins (P1.4-P1.6): CLKOUT

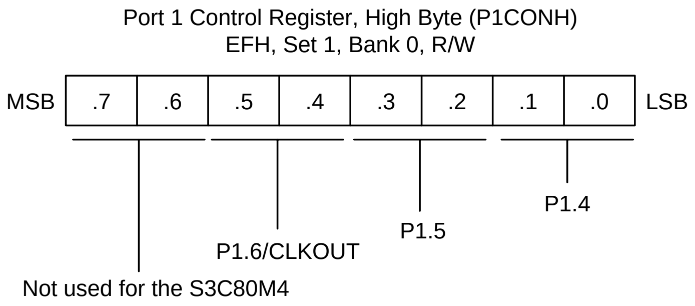
Port 1 Control Register (P1CONH, P1CONL)

Port 1 has two 8-bit control registers: P1CONH for P1.4–P1.6 and P1CONL for P1.0–P1.3. A reset clears the P1CONH and P1CONL registers to “00H”, configuring all pins to input mode. You use control registers settings to select input or output mode (push-pull or open drain) and enable the alternative functions.

When programming the port, please remember that any alternative peripheral I/O function you configure using the port 1 control registers must also be enabled in the associated peripheral module.

Port 1 Pull-up Resistor Enable Register (P1PUR)

Using the port 1 pull-up resistor enable register, P1PUR (F1H, set 1, bank 0), you can configure pull-up resistors to individual port 1 pins.

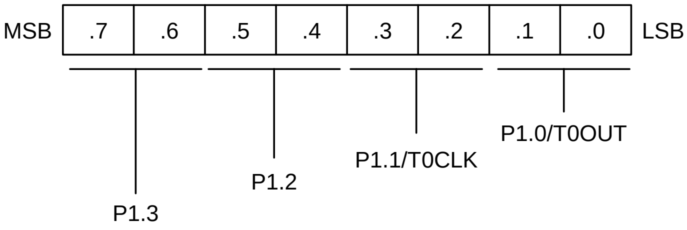


P1CONH bit-pair pin configuration settings:

00	Input mode
01	Output mode, N-channel open-drain
10	Alternative function (CLKOUT, not used for P1.5/P1.4)
11	Output mode, Push-pull

Figure 9-5. Port 1 High-Byte Control Register (P1CONH)

Port 1 Control Register, Low Byte (P1CONL)
F0H, Set 1, Bank 0, R/W

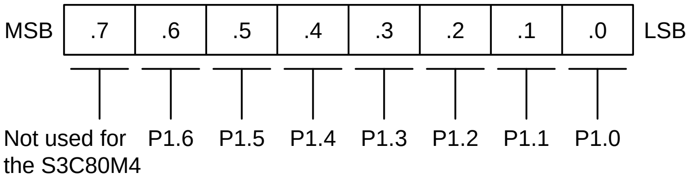


P1CONL bit-pair pin configuration settings:

00	Input mode (T0CLK)
01	Output mode, N-channel open-drain
10	Alternative function (T0OUT, not used for P1.3/P1.2/P1.1)
11	Output mode, push-pull

Figure 9-6. Port 1 Low-Byte Control Register (P1CONL)

Port 1 Pull-up Resistor Enable Register (P1PUR)
F1H, Set 1, Bank 0, R/W



P1PUR bit configuration settings:

0	Pull-up Disable
1	Pull-up Enable

Figure 9-7. Port 1 Pull-up Resistor Enable Register (P1PUR)

10 BASIC TIMER

OVERVIEW

S3C80M4/F80M4 has an 8-bit basic timer .

BASIC TIMER (BT)

You can use the basic timer (BT) in two different ways:

— As a watchdog timer to provide an automatic reset mechanism in the event of a system malfunction, or

— To signal the end of the required oscillation stabilization interval after a reset or a Stop mode release.

The functional components of the basic timer block are:

— Clock frequency divider (fxx divided by 4096, 1024, 128, or 16) with multiplexer

— 8-bit basic timer counter, BTCNT (set 1, Bank 0, FDH, read-only)

— Basic timer control register, BTCON (set 1, D3H, read/write)

BASIC TIMER CONTROL REGISTER (BTCON)

The basic timer control register, BTCON, is used to select the input clock frequency, to clear the basic timer counter and frequency dividers, and to enable or disable the watchdog timer function. It is located in set 1, address D3H, and is read/write addressable using Register addressing mode.

A reset clears BTCON to "00H". This enables the watchdog function and selects a basic timer clock frequency of fxx/4096. To disable the watchdog function, you must write the signature code "1010B" to the basic timer register control bits BTCON.7–BTCON.4.

The 8-bit basic timer counter, BTCNT (set 1, bank 0, FDH), can be cleared at any time during the normal operation by writing a "1" to BTCON.1. To clear the frequency dividers, write a "1" to BTCON.0.

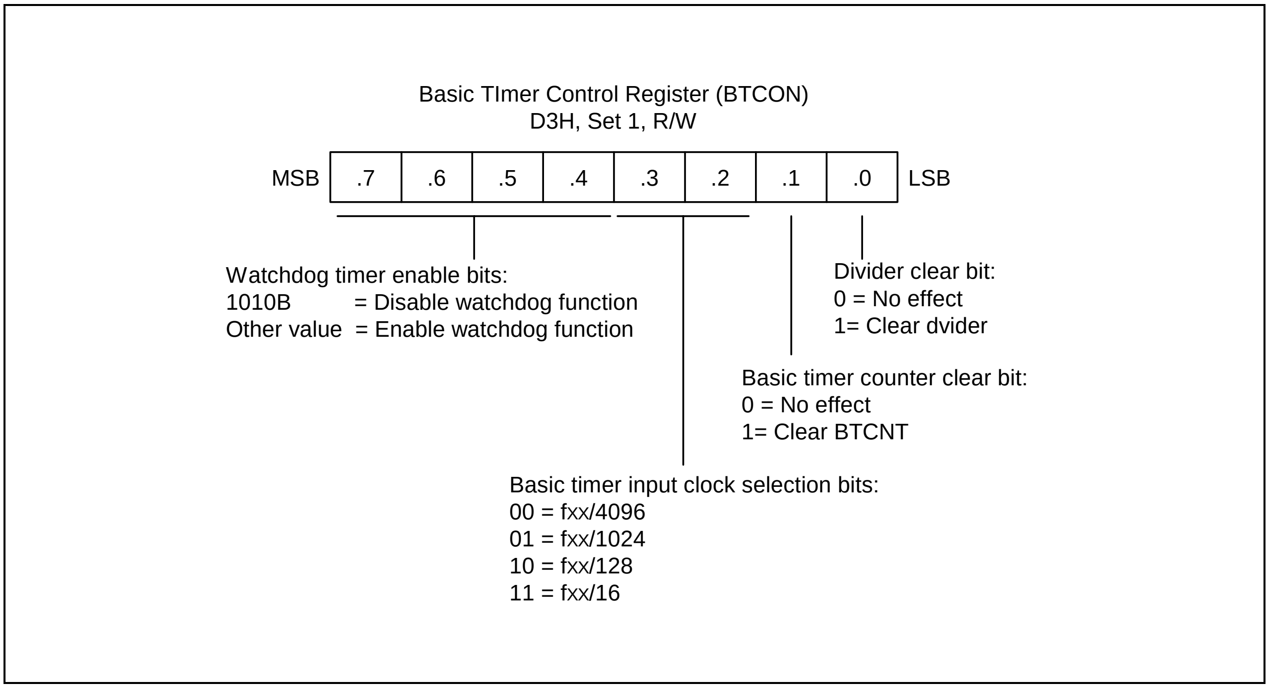

Basic TImer Control Register (BTCON)
D3H, Set 1, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Watchdog timer enable bits:
1010B = Disable watchdog function
Other value = Enable watchdog function
Divider clear bit:
0 = No effect
1= Clear dvider
Basic timer counter clear bit:
0 = No effect
1= Clear BTCNT
Basic timer input clock selection bits:
00 = fxx/4096
01 = fxx/1024
10 = fxx/128
11 = fxx/16

Figure 10-1. Basic Timer Control Register (BTCON)

BASIC TIMER FUNCTION DESCRIPTION

Watchdog Timer Function

You can program the basic timer overflow signal (BTOVF) to generate a reset by setting BTCON.7–BTCON.4 to any value other than "1010B". (The "1010B" value disables the watchdog function.) A reset clears BTCON to "00H", automatically enabling the watchdog timer function. A reset also selects the CPU clock (as determined by the current CLKCON register setting), divided by 4096, as the BT clock.

The MCU is reset whenever a basic timer counter overflow occurs, During normal operation, the application program must prevent the overflow, and the accompanying reset operation, from occurring, To do this, the BTCNT value must be cleared (by writing a “1” to BTCON.1) at regular intervals.

If a system malfunction occurs due to circuit noise or some other error condition, the BT counter clear operation will not be executed and a basic timer overflow will occur, initiating a reset. In other words, during the normal operation, the basic timer overflow loop (a bit 7 overflow of the 8-bit basic timer counter, BTCNT) is always broken by a BTCNT clear instruction. If a malfunction does occur, a reset is triggered automatically.

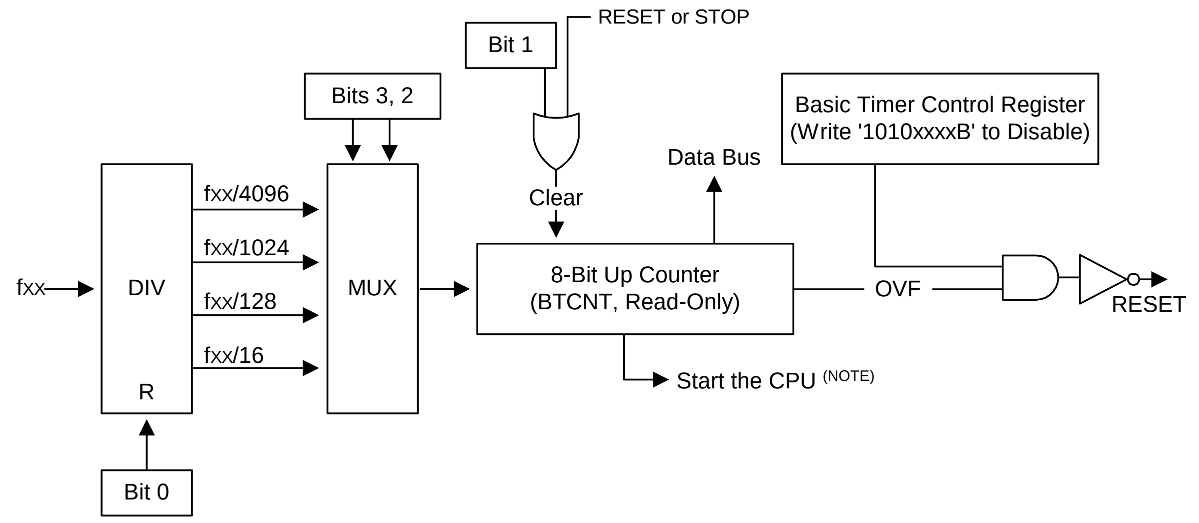
Oscillation Stabilization Interval Timer Function

You can also use the basic timer to program a specific oscillation stabilization interval after a reset or when stop mode has been released by an external interrupt.

In stop mode, whenever a reset or an external interrupt occurs, the oscillator starts. The BTCNT value then starts increasing at the rate of fxx/4096 (for reset), or at the rate of the preset clock source (for an external interrupt). When BTCNT.4 overflows, a signal is generated to indicate that the stabilization interval has elapsed and to gate the clock signal off to the CPU so that it can resume the normal operation.

In summary, the following events occur when stop mode is released:

1. During the stop mode, a power-on reset or an external interrupt occurs to trigger the Stop mode release and oscillation starts.
2. If a power-on reset occurred, the basic timer counter will increase at the rate of fxx/4096. If an interrupt is used to release stop mode, the BTCNT value increases at the rate of the preset clock source.
3. Clock oscillation stabilization interval begins and continues until bit 4 of the basic timer counter overflows.
4. When a BTCNT.4 overflow occurs, the normal CPU operation resumes.



NOTE: During a power-on reset operation, the CPU is idle during the required oscillation stabilization interval (until bit 4 of the basic timer counter overflows).

Figure 10-2. Basic Timer Block Diagram

11 8-BIT TIMER 0

OVERVIEW

The 8-bit timer 0 is an 8-bit general-purpose timer/counter.

Timer 0 has the following functional components:

— Clock frequency divider (fxx divided by 1024, 256, 64, 8 or 1) with multiplexer

— External clock input pin (T0CLK)

— 8-bit counter (T0CNT), 8-bit comparator, and 8-bit reference data register (T0DATA)

— I/O pins for match output (T0OUT)

— Timer 0 interrupt (IRQ0, vector EEH) generation

— Timer 0 control register, T0CON (set 1, Bank 0, E6H, read/write)

TIMER 0 FUNCTION DESCRIPTION

Interval Timer Mode

The timer 0 can generate an interrupt, the timer 0 match interrupt (T0INT). T0INT belongs to interrupt level IRQ0, and is assigned the separate vector address, EEH.

The T0INT pending condition should be cleared by software when it has been serviced. Even though T0INT is disabled, the application's service routine can detect a pending condition of T0INT by the software and execute its sub-routine. When this case is used, the T0INT pending bit must be cleared by application sub-routine by writing a "0" to the T0CON.0 pending bit.

In interval timer mode, a match signal is generated when the counter value is identical to the value written to the timer 0 reference data register, T0DATA. The match signal generates a timer 0 match interrupt (T0INT, vector EEH) and clears the counter.

If, for example, you write the value "10H" to T0DATA, the counter will increment until it reaches "10H". At this point, the timer 0 interrupt request is generated, the counter value is reset, and counting resumes

TIMER 0 CONTROL REGISTER (T0CON)

You use the timer 0 control register, T0CON, to

— Enable the timer 0 operating mode (interval timer)
— Select the timer 0 input clock frequency
— Clear the timer 0 counter, T0CNT
— Enable the timer 0 interrupt
— Clear timer 0 interrupt pending condition

T0CON is located in set 1, Bank 0 at address E6H, and is read/write addressable using Register addressing mode.

A reset clears T0CON to '00H'. This sets timer 0 to normal interval timer mode, selects an input clock frequency of fxx/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.

To enable the timer 0 interrupt (IRQ0, vector EEH), you must write T0CON.2, and T0CON.1 to "1". To detect an interrupt pending condition, when T0INT is disabled, the application program polls pending bit, T0CON.0. When a "1" is detected, a timer 0 interrupt is pending. When the interrupt request has been serviced, the pending condition must be cleared by software by writing a "0" to the timer 0 interrupt pending bit, T0CON.0.

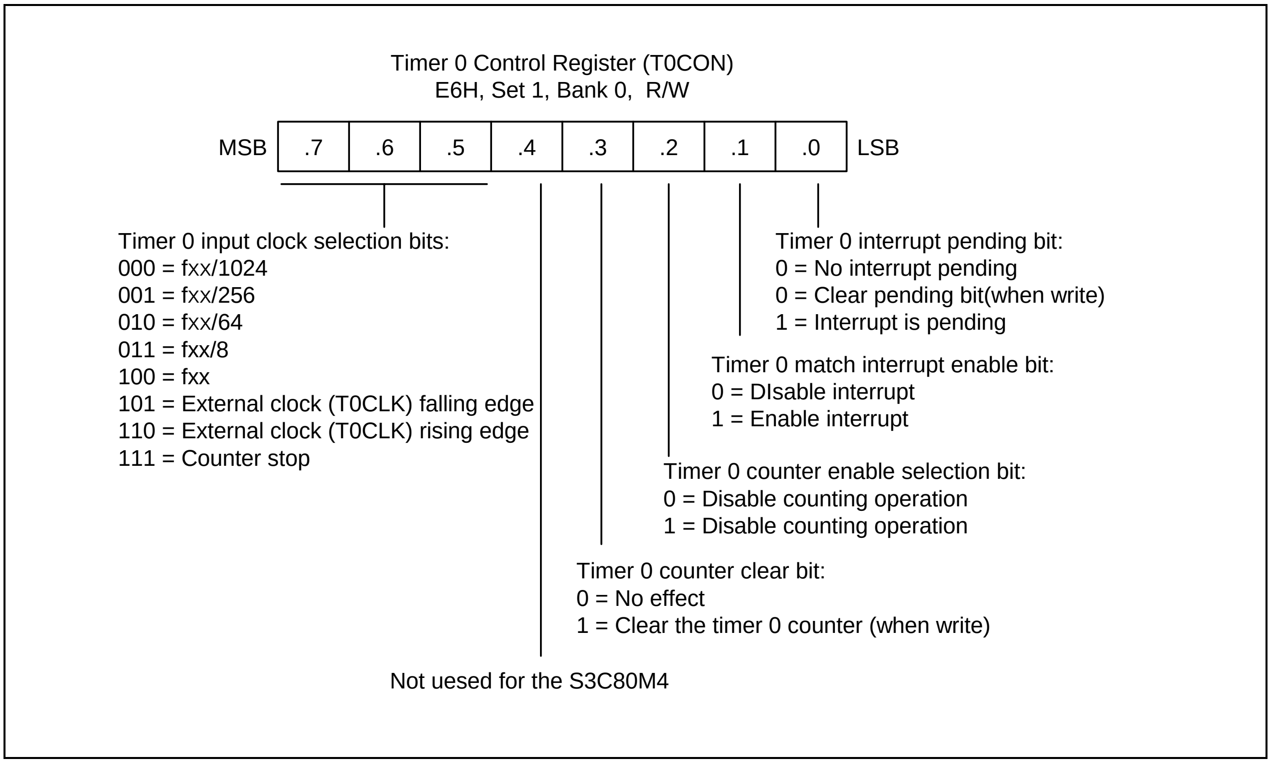

Timer 0 Control Register (T0CON)
E6H, Set 1, Bank 0, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Timer 0 input clock selection bits:
000 = fxx/1024
001 = fxx/256
010 = fxx/64
011 = fxx/8
100 = fxx
101 = External clock (T0CLK) falling edge
110 = External clock (T0CLK) rising edge
111 = Counter stop
Timer 0 interrupt pending bit:
0 = No interrupt pending
0 = Clear pending bit(when write)
1 = Interrupt is pending
Timer 0 match interrupt enable bit:
0 = DIsable interrupt
1 = Enable interrupt
Timer 0 counter enable selection bit:
0 = Disable counting operation
1 = Disable counting operation
Timer 0 counter clear bit:
0 = No effect
1 = Clear the timer 0 counter (when write)
Not uesed for the S3C80M4

Figure 11-1. Timer 0 Control Register (T0CON)

BLOCK DIAGRAM

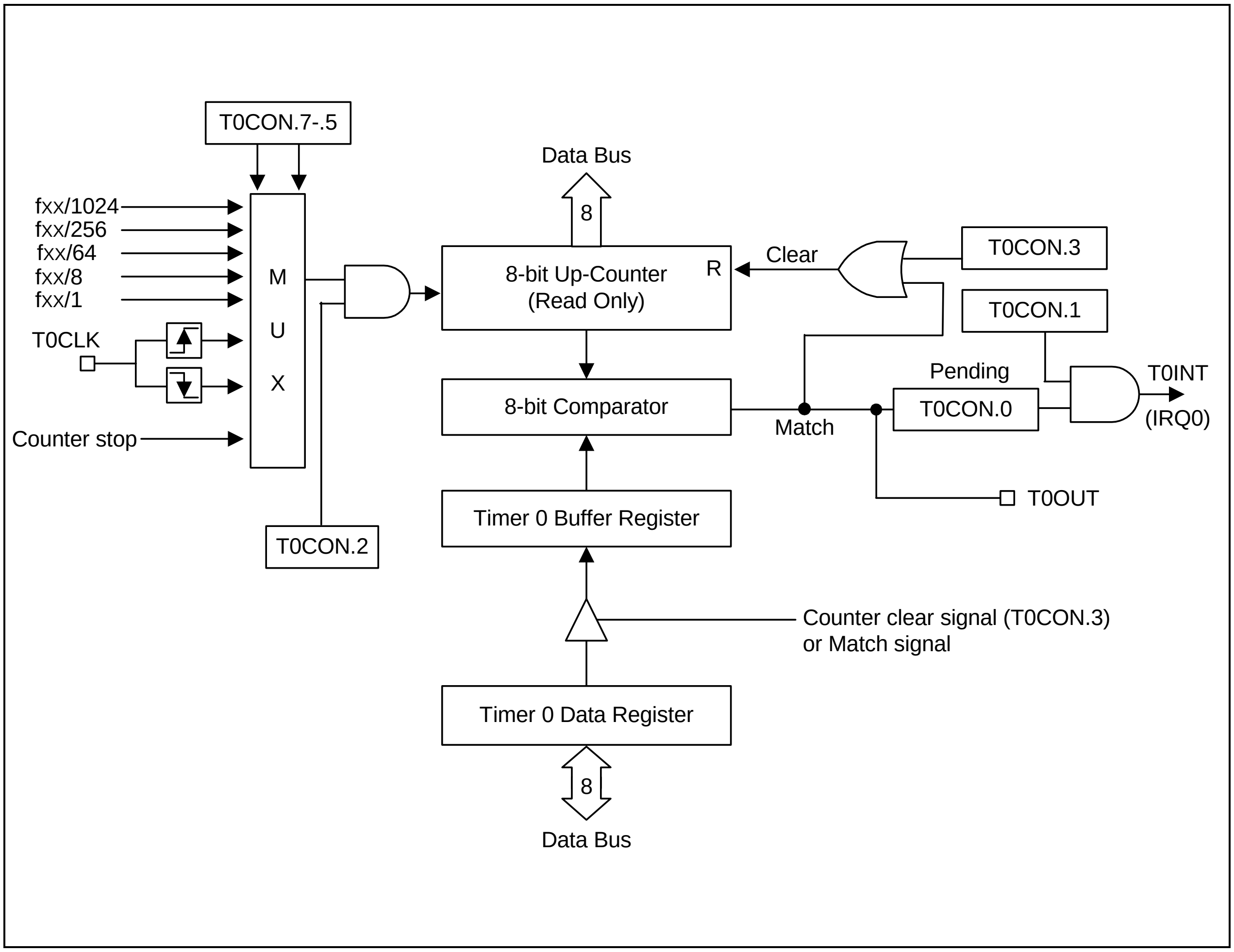


Figure 11-2. Timer 0 Functional Block Diagram

NOTES

12 8-BIT PULSE WIDTH MODULATION

OVERVIEW

The S3C80M4/F80M4 microcontroller has a 8-bit PWM.
The PWM have the following components:

— Clock frequency dividers (f_{OSC} divider by 64, 8, 2 and 1)

— 6-bit counter, 6-bit comparators and data registers (PWMDATA)

— 8-bit counter overflow interrupt generations

— Selectors for data reload 6- and 8- bit overflow

— PWM control register, PWMON (set 1, bank 0, E8H, read/write)

8-BIT PULSE WIDTH MODULATION (PWMCON)

The PWM control register, PWMCON is used to select the PWM interrupt to enable or disable the PWM function. It is located in set 1, bank 0 at address E8H, and is read/write addressable using register addressing mode.
A reset clears PWMCON to "00H". This disable the PWM interrupt, selects an input clock frequency of fosc/64, disables all PWM interrupt. So, if you want to use the PWM, you must write PWMCON.5 to "1" and write P0CONH.5-.4 to "10".

To enable the PWM interrupt (IRQ2, vector EAH), you must write PWMCON.2, and PWMCON.1 to "1". To detect an interrupt pending condition when PWMINT is disabled, the application program polls pending bit, PWMCON.0. When a "1" is detected, a PWM interrupt is pending. When PWMINT sub-routine has been serviced, the pending condition must be cleared by software by writing a "0" to the PWM interrupt pending bit, PWMCON.0.

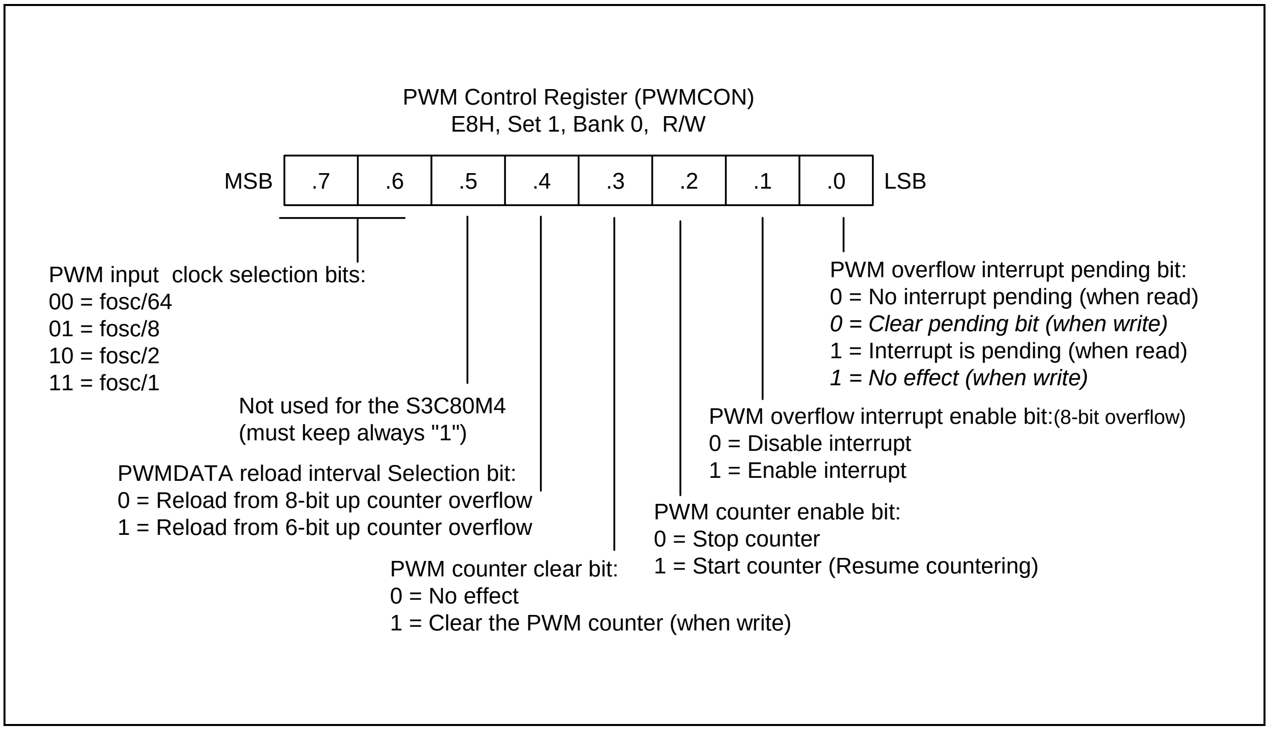

PWM Control Register (PWMCON)
E8H, Set 1, Bank 0, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
PWM input clock selection bits:
00 = fosc/64
01 = fosc/8
10 = fosc/2
11 = fosc/1
Not used for the S3C80M4
(must keep always "1")
PWMDATA reload interval Selection bit:
0 = Reload from 8-bit up counter overflow
1 = Reload from 6-bit up counter overflow
PWM counter clear bit:
0 = No effect
1 = Clear the PWM counter (when write)
PWM counter enable bit:
0 = Stop counter
1 = Start counter (Resume countering)
PWM overflow interrupt enable bit:(8-bit overflow)
0 = Disable interrupt
1 = Enable interrupt
PWM overflow interrupt pending bit:
0 = No interrupt pending (when read)
0 = Clear pending bit (when write)
1 = Interrupt is pending (when read)
1 = No effect (when write)

Figure 12-1. PWM Control Register (PWMCON)

BLOCK DIAGRAM

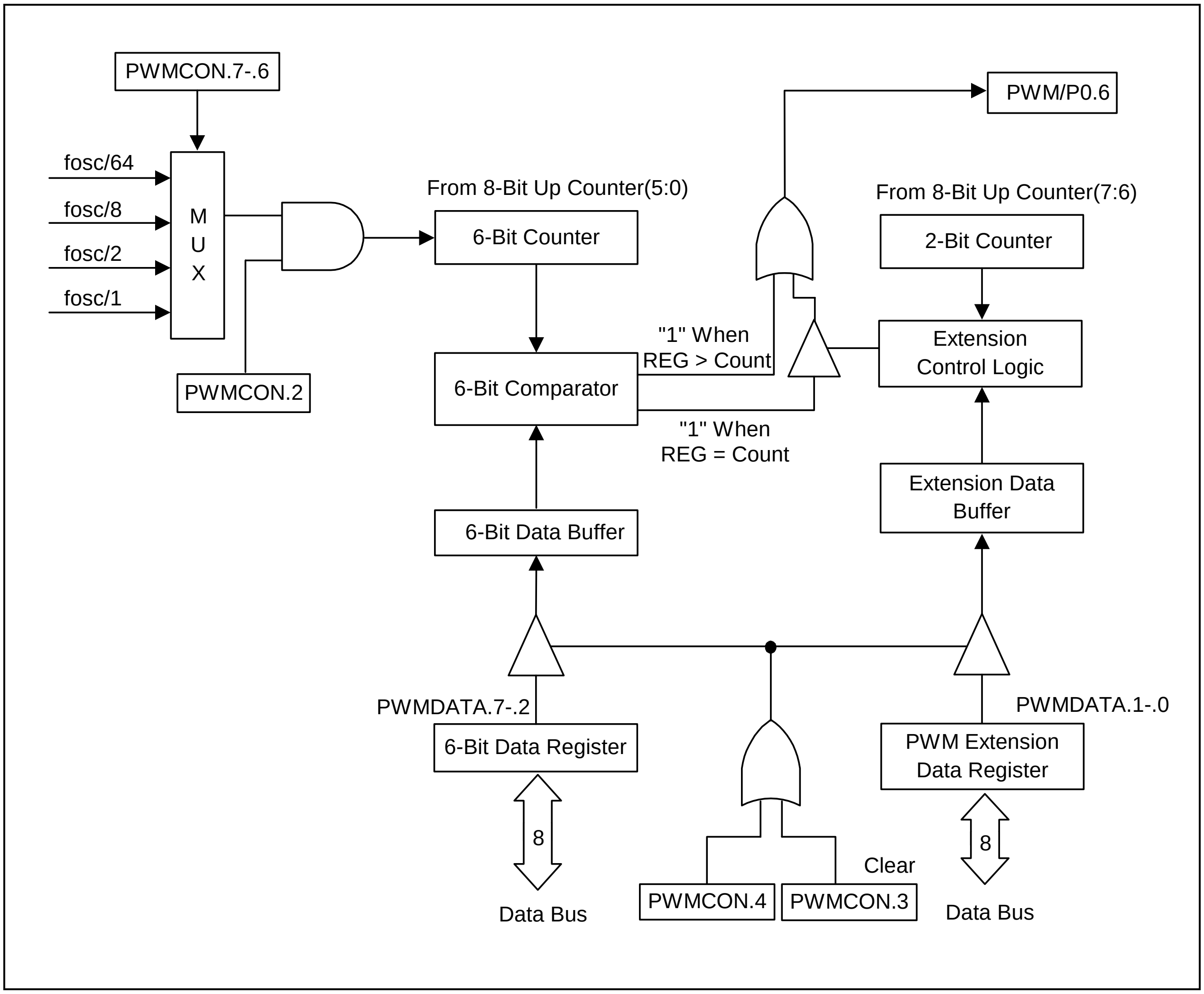


Figure 12-2. PWM Circuit Diagram

NOTES

13 ELECTRICAL DATA

OVERVIEW

In this chapter, S3C80M4/F80M4 electrical characteristics are presented in tables and graphs. The information is arranged in the following order:

- Absolute maximum ratings
- D.C. electrical characteristics
- Input/output capacitance
- A.C. electrical characteristics
- Oscillation characteristics
- Oscillation stabilization time
- Data retention supply voltage in stop mode
- Operating voltage range

Table13-1. 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	Ports 0-1	– 0.3 to V_{DD} + 0.3	
Output voltage	V_O	–	– 0.3 to V_{DD} + 0.3	
Output current high	I_{OH}	One I/O pin active	– 15	mA
		All I/O pins active	– 60	
Output current low	I_{OL}	One I/O pin active	+ 30(Peak value)	
		Total pin current for ports	+ 100(Peak value)	
Operating temperature	T_A	–	– 25 to + 85	°C
Storage temperature	T_{STG}	–	– 65 to + 150	

Table 13-2. D.C. Electrical Characteristics

(T_A = –25 °C to + 85 °C, V_{DD} = 2.4 V to 5.5V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Operating voltage	V_{DD}	f_X = 0.4 – 4.2 MHz	2.4	–	5.5	V
		f_X = 0.4 – 10.0 MHz	2.7	–	5.5	
Input high voltage	V_{IH1}	All input pins except V_{IH2}, V_{IH3}	$0.7V_{DD}$	–	V_{DD}	
	V_{IH2}	Ports0, Ports1.0 - 1.3, nRESET	$0.8V_{DD}$		V_{DD}	
	V_{IH3}	X_{IN}, X_{OUT}	V_{DD}-0.1		V_{DD}	
Input low voltage	V_{IL1}	All input pins except V_{IL2}, V_{IL3}	–	–	$0.3V_{DD}$	
	V_{IL2}	Ports0, Ports1.0 - 1.3, nRESET			$0.2V_{DD}$	
	V_{IL3}	X_{IN}, X_{OUT}			0.1	

Table 13-2. D.C. Electrical Characteristics (Continued)

(T_A = –25 °C to + 85 °C, V_{DD} = 2.4V to 5.5V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Output high voltage	V_{OH}	V_{DD} = 4.5V to 5.5V I_{OH} = –1 mA All output pins	V_{DD}–1.0	–	–	V
Output low voltage	V_{OL1}	V_{DD} = 4.5V to 5.5V I_{OL} = 15 mA Ports1.0–.3	–	–	2.0	
	V_{OL2}	V_{DD} = 4.5V to 5.5V I_{OL} = 10 mA All output ports except V_{OL1}	–	–	2.0	
Input high leakage current	I_{LIH1}	V_{IN} = V_{DD} All input pins except I_{LIH2}	–	–	3	μA
	I_{LIH2}	V_{IN} = $V_{DD,}$ X_{IN}, X_{OUT}			20	
Input low leakage current	I_{LIL1}	V_{IN} = 0 V All input pins except for nRESET, I_{LIL2}	–	–	–3	
	I_{LIL2}	V_{IN} = 0 V, X_{IN}, X_{OUT}			–20	
Output high leakage current	I_{LOH}	V_{OUT} = V_{DD} All output pins	–	–	3	
Output low leakage current	I_{LOL}	V_{OUT} = 0 V All output pins	–	–	–3	
Oscillator feed back resistors	R_{OSC1}	V_{DD} = 5 V, T_A=25 °C X_{IN} = V_{DD}, X_{OUT} = 0 V	300	600	1200	kΩ
Pull-up resistor	R_{L1}	V_{IN} = 0 V, T_A = 25 °C Port 0–1 V_{DD} = 5 V	30	60	120	
		V_{IN} = 0 V, T_A = 25 °C Port 0–1 V_{DD} = 3 V	60	110	220	

Table 13-2. D.C. Electrical Characteristics (Continued)

(T_A = –25 °C to + 85 °C, V_{DD} = 2.4 V to 5.5 V)

Parameter	Symbol	Conditions		Min	Typ	Max	Unit
Supply current (1)	I_{DD1}	Run mode: Crystal oscillator C1 = C2 = 22pF V_{DD} = 5.0V ± 10%	10 MHz	–	4.0	8.0	mA
			4.0 MHz		2.0	4.0	
		V_{DD} = 3.0V ± 10%	4.0 MHz		1.5	3.0	
	I_{DD2}	Idle mode: Crystal oscillator C1 = C2 = 22pF V_{DD} = 5.0V ± 10%	10 MHz	–	1.2	2.4	
			4.0 MHz		1.0	2.0	
		V_{DD} = 3.0V ± 10%	4.0 MHz	–	0.5	1.0	
	I_{DD3}(2)	Stop mode: V_{DD} = 5V ± 10%, T_A = 25 °C		–	100	200	μA
		V_{DD} = 3V ± 10%, T_A = 25 °C		–	80	160	

NOTES:

1. Supply current does not include current drawn through internal pull-up resistors and external output current loads.
2. I_{DD3} is current when main clock oscillation stops.
3. Every values in this table is measured when bits 4-3 of the system clock control register (CLKCON.4–.3) is set to 11B.

Table 13-3. A.C. Electrical Characteristics

(T_A = –25 °C to +85 °C, V_{DD} = 2.4 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Interrupt input high, low width	t_{INTH}, t_{INTL}	All interrupt, V_{DD} = 3.0 V	500	700	–	ns
nRESET input low width	t_{RSL}	V_{DD} = 3.0 V	10	–	–	µs

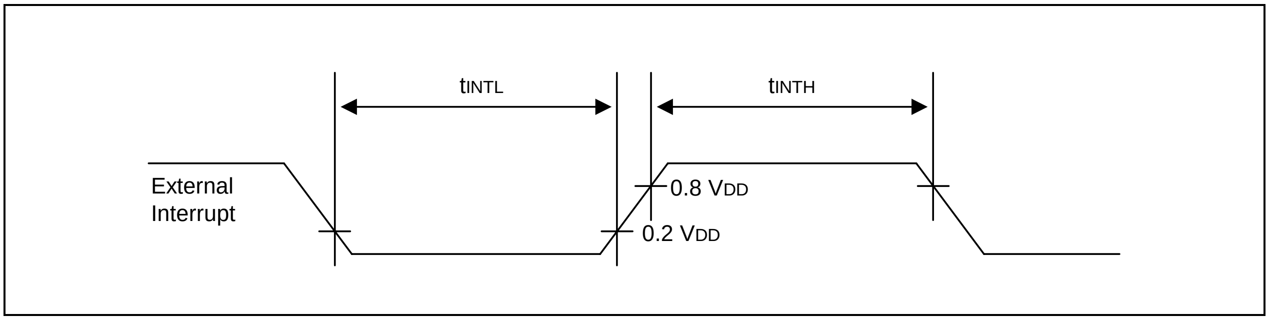


Figure 13-1. Input Timing for External Interrupts

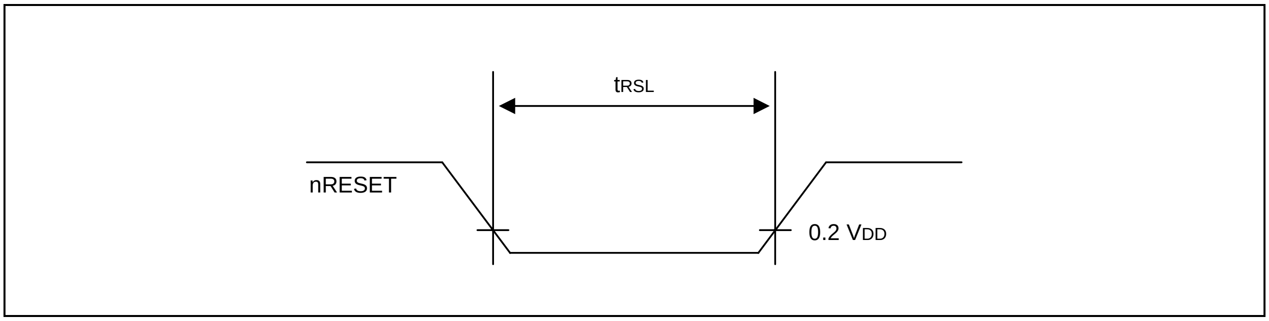


Figure 13-2. Input Timing for nRESET

Table 13-4. Input/Output Capacitance

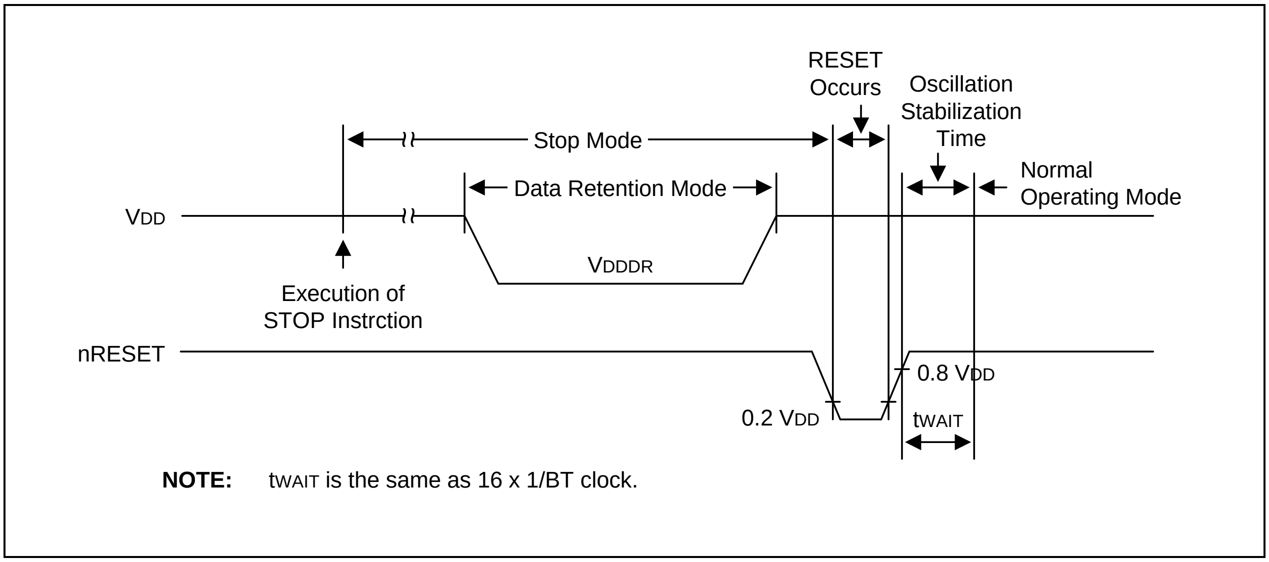
(T_A = –25 °C to +85 °C, V_{DD} = 2.4 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Input capacitance	C_{IN}	f = 1 MHz; unmeasured pins are returned to V_{SS}	–	–	10	pF
Output capacitance	C_{OUT}					
I/O capacitance	C_{IO}					

Table 13-5. Data Retention Supply Voltage in Stop Mode

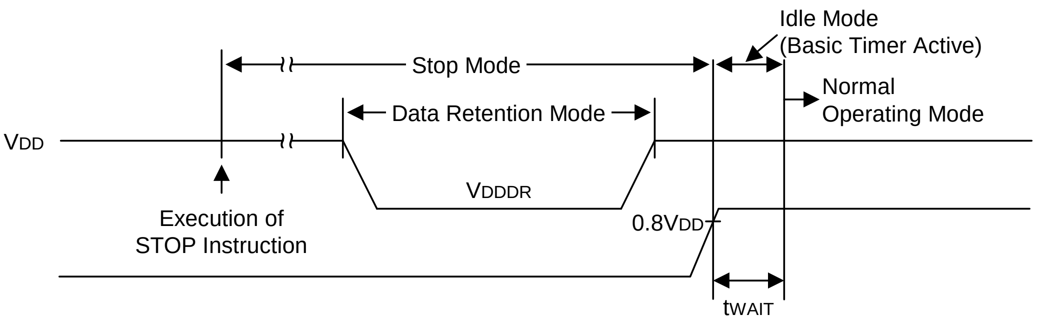
(T_A = –25 °C to + 85 °C, V_{DD} = 2.4 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Data retention supply voltage	V_{DDDR}		2.4	–	5.5	V
Data retention supply current	I_{DDDR}	V_{DDDR} = 2.4V Stop mode, T_A = 25 °C	–	–	1	uA



NOTE: tWAIT is the same as 16 x 1/BT clock.

Figure 13-3. Stop Mode Release Timing Initiated by RESET



NOTE: t_{WAIT} is the same as 16 x 1/BT clock.

Figure 13-4. Stop Mode Release Timing Initiated by Interrupt

Table13-6. Main Oscillator Characteristics

(T_A = –25 °C to +85 °C, V_{DD} = 2.4V to 5.5V)

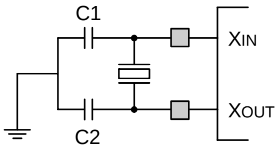
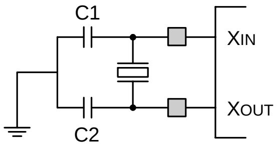
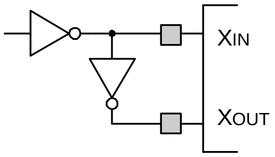
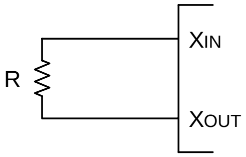
Oscillator	Clock Configuration	Parameter	Test Condition	Min	Typ	Max	Units
Crystal		Main oscillation frequency	2.7 V – 5.5 V	0.4	–	10	MHz
			2.4 V – 5.5 V	0.4	–	4.2	
Ceramic Oscillator		Main oscillation frequency	2.7 V – 5.5 V	0.4	–	10	
			2.4 V – 5.5 V	0.4	–	4.2	
External Clock		X_{IN} input frequency	2.7 V – 5.5 V	0.4	–	10	
			2.4 V – 5.5 V	0.4	–	4.2	
RC Oscillator		Frequency	5.0 V	0.4	–	2	MHz
			3.0 V	0.4	–	1	

Table 13-7. Main Oscillation Stabilization Time

(T_A = –25 °C to + 85 °C, V_{DD} = 2.4V to 5.5V)

Oscillator	Test Condition	Min	Typ	Max	Unit
Crystal	fx > 1 MHz Oscillation stabilization occurs when V_{DD} is equal to the minimum oscillator voltage range.	–	–	40	ms
Ceramic		–	–	10	ms
External clock	X_{IN} input high and low width (t_{XH}, t_{XL})	62.5	–	1250	ns

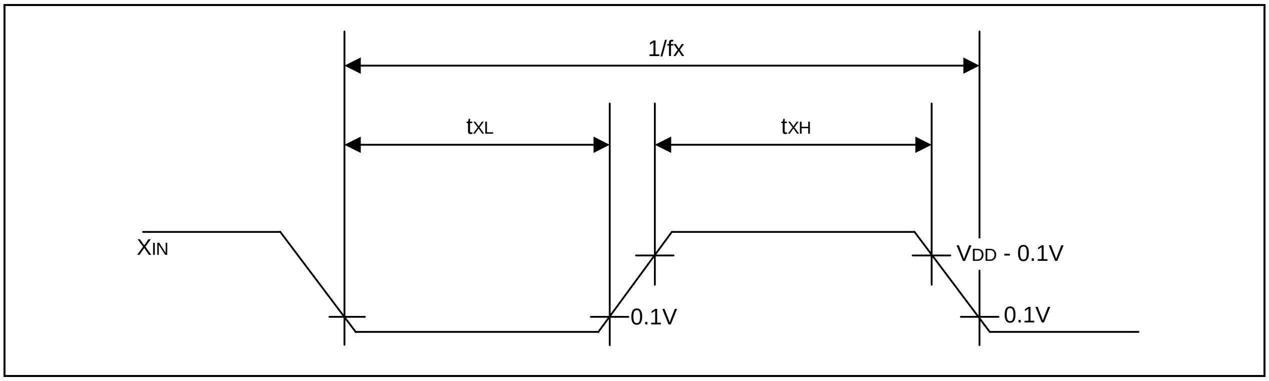


Figure 13-5. Clock Timing Measurement at X_{IN}

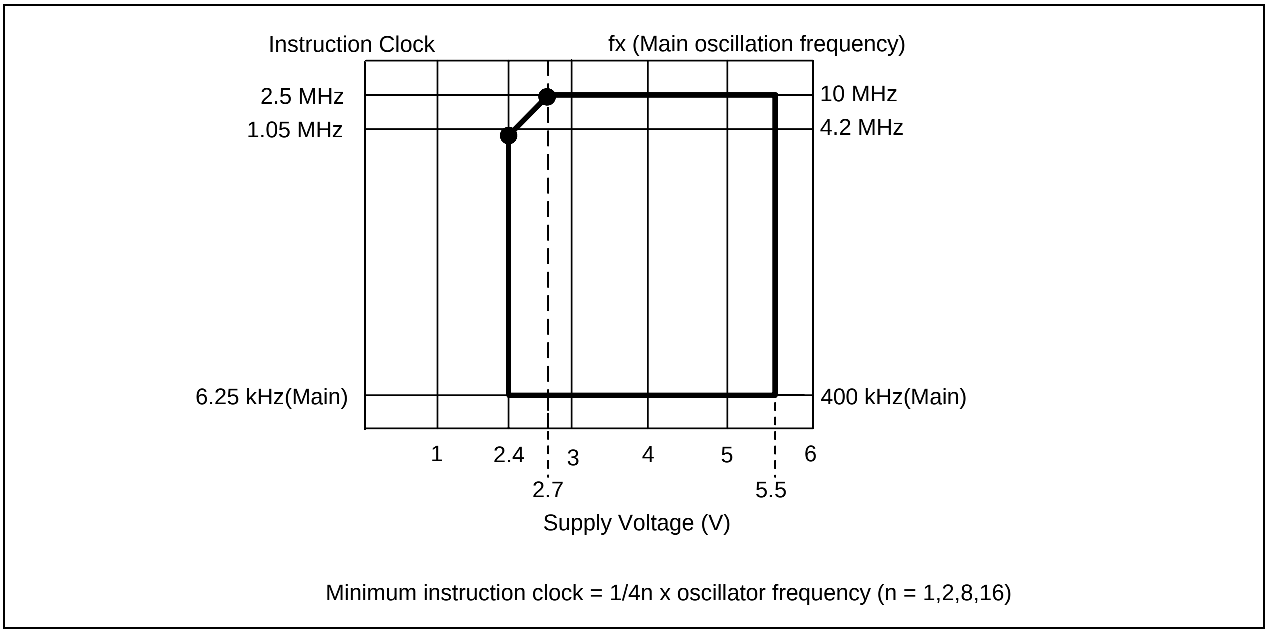


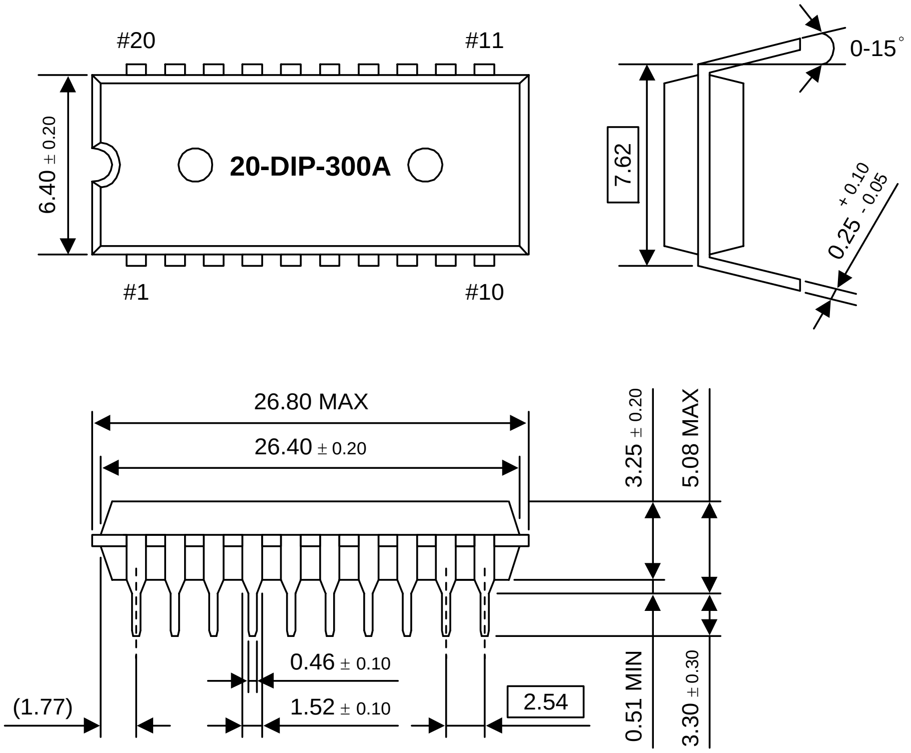
Figure 13-6. Operating Voltage Range

NOTES

14 MECHANICAL DATA

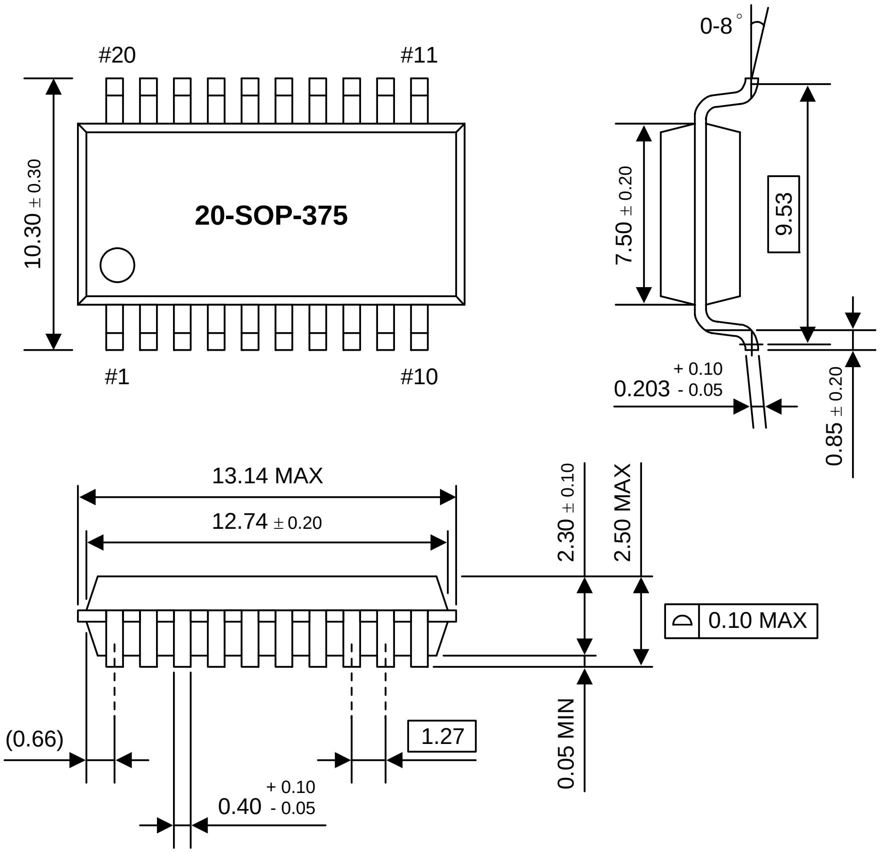
OVERVIEW

The S3C80M/F80M4 microcontroller is currently available in 20-DIP-300A/20-SOP-375 and 16-DIP-300A/16-SOP-375 package.



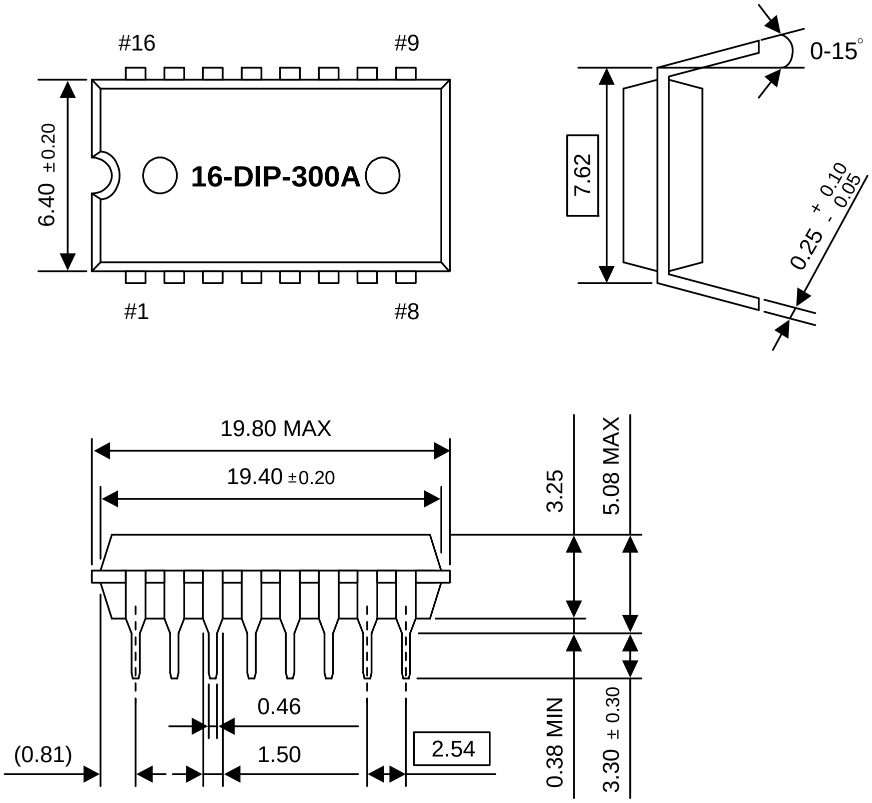
NOTE: Dimensions are in millimeters.

Figure 14-1. 20-DIP-300A Package Dimensions



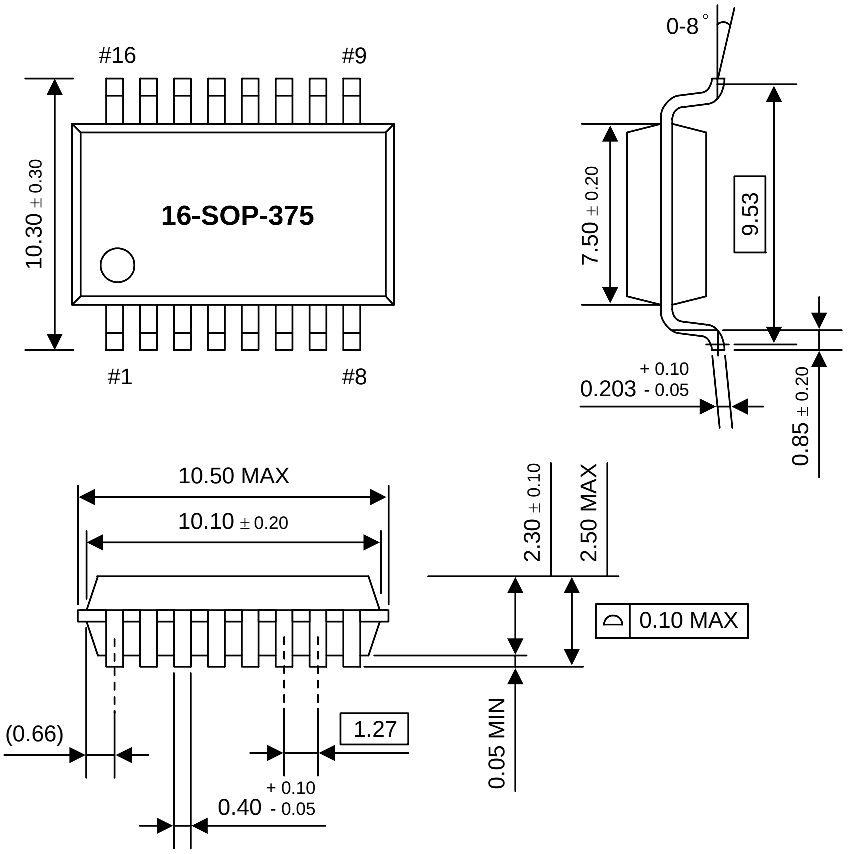
NOTE: Dimensions are in millimeters.

Figure 14-2. 20-SOP-375 Package Dimensions



NOTE: Dimensions are in millimeters.

Figure 14-3. 16-DIP-300A Package Dimensions



NOTE: Dimensions are in millimeters.

Figure 14-4. 16-SOP-375 Package Dimensions

15 S3F80M4 FLASH MCU

OVERVIEW

The S3F80M4 single-chip CMOS microcontroller is the Flash MCU version of the S3C80M4 microcontroller. It has an on-chip Flash MCU ROM instead of a masked ROM. The Flash ROM is accessed by serial data format.

The S3F80M4 is fully compatible with the S3C80M4, both in function and in pin configuration. Because of its simple programming requirements, the S3F80M4 is ideal as an evaluation chip for the S3C80M4.

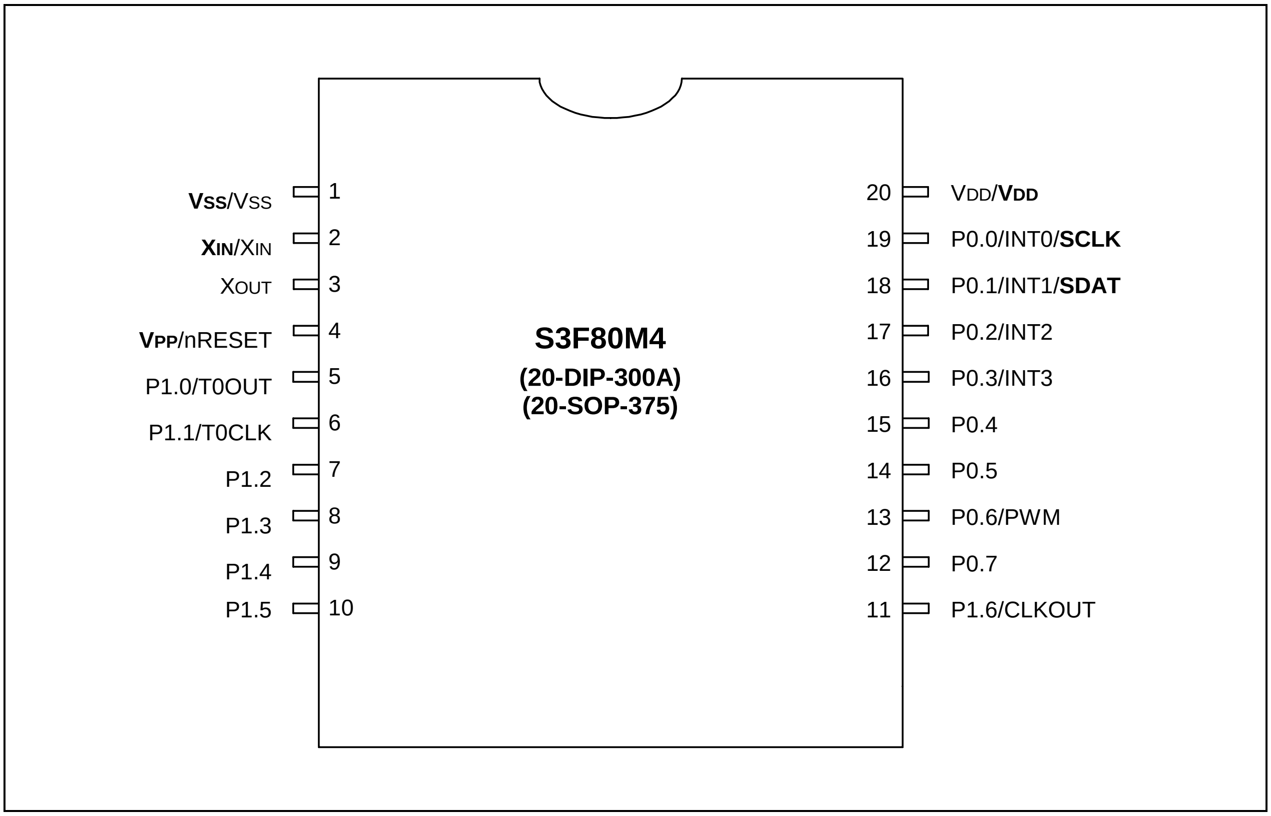


Figure 15-1. S3F80M4 Pin Assignments (20-DIP-300A, 20-SOP-375)

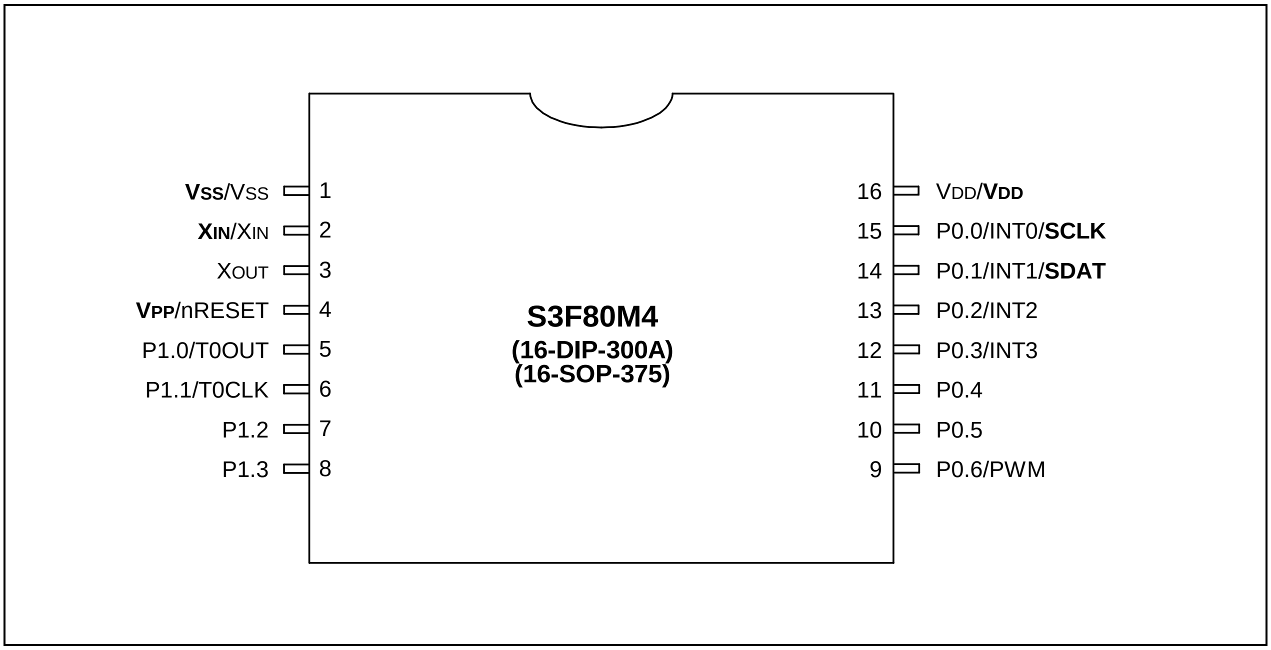


Figure 15-2. S3F80M4 Pin Assignments (16-DIP-300A, 16-SOP-375)

Table 15-1. Descriptions of Pins Used to Read/Write the EPROM

Main Chip	During Programming			
Pin Name	Pin Name	Pin No.	I/O	Function
P0.1	SDAT	18(14)	I/O	Serial data pin. Output port when reading and input port when writing. Can be assigned as a Input/push-pull output port.
P0.0	SCLK	19(15)	I/O	Serial clock pin. Input only pin.
nRESET	V_{PP}	4(4)	I	Power supply pin for Flash ROM cell writing (indicates that FLASH MCU enters into the writing mode). When 12.5 V is applied, FLASH MCU is in writing mode and when 3.3 V is applied, FLASH MCU is in reading mode. (Option)

V_{DD}
V_{SS}
	V_{DD}
	V_{SS}
		20(16)
		1(1)
			–	Power supply pin for logic circuit. V_{DD} should be tied to +3.3V during programming.

X_{IN}	X_{IN}	2(2)	I	This pin should be connected to V_{SS} in the tool program mode.

NOTE: Parentheses indicate pin number for 16-DIP-300A/16-SOP-375 package.

Table 15-2. Comparison of S3F80M4 and S3C80M4 Features

Characteristic	S3F80M4	S3C80M4
Program Memory	4K-byte Flash ROM	4K-byte mask ROM
Operating Voltage (V_{DD})	2.4 V to 5.5 V	2.4 V to 5.5 V
FLASH MCU Programming Mode	V_{DD} = 3.3 V, V_{PP} (nRESET) = 12.5 V	
Programmability	User Program multi time	Programmed at the factory

OPERATING MODE CHARACTERISTICS

When 12.5 V is supplied to the V_{PP} (nRESET) pin of the S3C80M4, the Flash ROM programming mode is entered. The operating mode (read, write, or read protection) is selected according to the input signals to the pins listed in Table 15-3 below.

Table 15-3. Operating Mode Selection Criteria

V_{DD}	V_{PP}(nRESET)	REG/nMEM	Address (A15–A0)	R/W	Mode
3.3 V	3.3 V	0	0000H	1	Flash ROM read
	12.5 V	0	0000H	0	Flash ROM program
	12.5 V	0	0000H	1	Flash ROM verify
	12.5 V	1	0E3FH	0	Flash ROM read protection

NOTE: "0" means Low level; "1" means High level.

Table 15-4. D.C. Electrical Characteristics

(T_A = –25 °C to + 85 °C, V_{DD} = 2.4 V to 5.5 V)

Parameter	Symbol	Conditions		Min	Typ	Max	Unit
Supply current[1]	I_{DD1}	Run mode: Crystal oscillator C1 = C2 = 22pF V_{DD} = 5.0V ± 10%	10 MHz	–	4.0	8.0	mA
			4.0 MHz	–	2.0	4.0	
		V_{DD} = 3.0V ± 10%	4.0 MHz	–	1.5	3.0	
	I_{DD2}	Idle mode: Crystal oscillator C1 = C2 = 22pF V_{DD} = 5.0V ± 10%	10 MHz	–	1.2	2.4	
			4.0 MHz	–	1.0	2.0	
		V_{DD} = 3.0V ± 10%	4.0 MHz	–	0.5	1.0	
	I_{DD3}[2]	Stop mode: V_{DD} = 5V ± 10%, T_A = 25 °C		–	100	200	μA
		V_{DD} = 3V ± 10%, T_A = 25 °C		–	80	160	

NOTES:

1. Supply current does not include current drawn through internal pull-up resistors and external output current loads.
2. I_{DD3} is current when main clock oscillation stops.
3. Every values in this table is measured when bits 4-3 of the system clock control register (CLKCON.4–.3) is set to 11B.

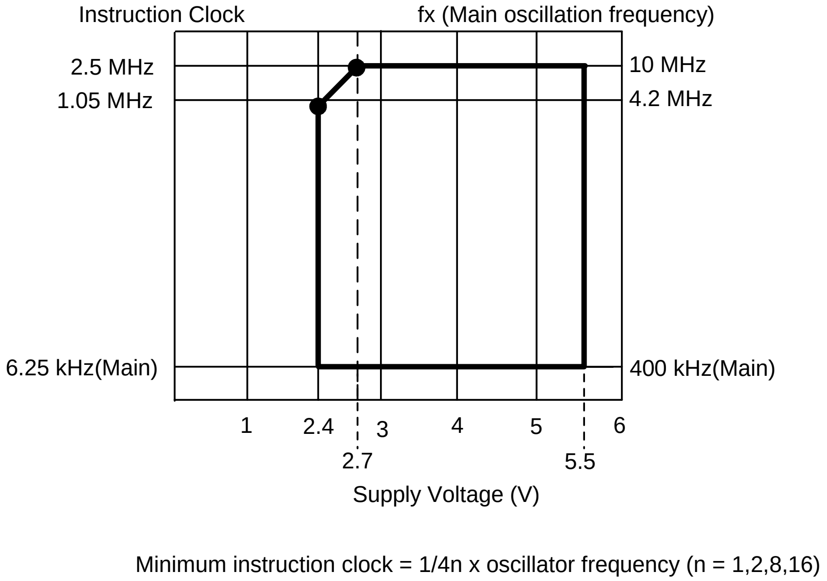


Figure 15-3. Operating Voltage Range

16 DEVELOPMENT TOOLS

OVERVIEW

Samsung provides a powerful and easy-to-use development support system in turnkey form. The development support system is configured with a host system, debugging tools, and support software. For the host system, any standard computer that operates with MS-DOS, Windows 95, and 98 as its operating system can be used. One type of debugging tool including hardware and software is provided: the sophisticated and powerful in-circuit emulator, SMDS2+, and OPENice for S3C7, S3C9, S3C8 families of microcontrollers. The SMDS2+ is a new and improved version of SMDS2. Samsung also offers support software that includes debugger, assembler, and a program for setting options.

SHINE

Samsung Host Interface for In-Circuit Emulator, SHINE, is a multi-window based debugger for SMDS2+. SHINE provides pull-down and pop-up menus, mouse support, function/hot keys, and context-sensitive hyper-linked help. It has an advanced, multiple-windowed user interface that emphasizes ease of use. Each window can be sized, moved, scrolled, highlighted, added, or removed completely.

SAMA ASSEMBLER

The Samsung Arrangeable Microcontroller (SAM) Assembler, SAMA, is a universal assembler, and generates object code in standard hexadecimal format. Assembled program code includes the object code that is used for ROM data and required SMDS program control data. To assemble programs, SAMA requires a source file and an auxiliary definition (DEF) file with device specific information.

SASM88

The SASM88 is a relocatable assembler for Samsung's S3C8-series microcontrollers. The SASM88 takes a source file containing assembly language statements and translates into a corresponding source code, object code and comments. The SASM88 supports macros and conditional assembly. It runs on the MS-DOS operating system. It produces the relocatable object code only, so the user should link object file. Object files can be linked with other object files and loaded into memory.

HEX2ROM

HEX2ROM file generates ROM code from HEX file which has been produced by assembler. ROM code must be needed to fabricate a microcontroller which has a mask ROM. When generating the ROM code (.OBJ file) by HEX2ROM, the value "FF" is filled into the unused ROM area up to the maximum ROM size of the target device automatically.

TARGET BOARDS

Target boards are available for all S3C8-series microcontrollers. All required target system cables and adapters are included with the device-specific target board.

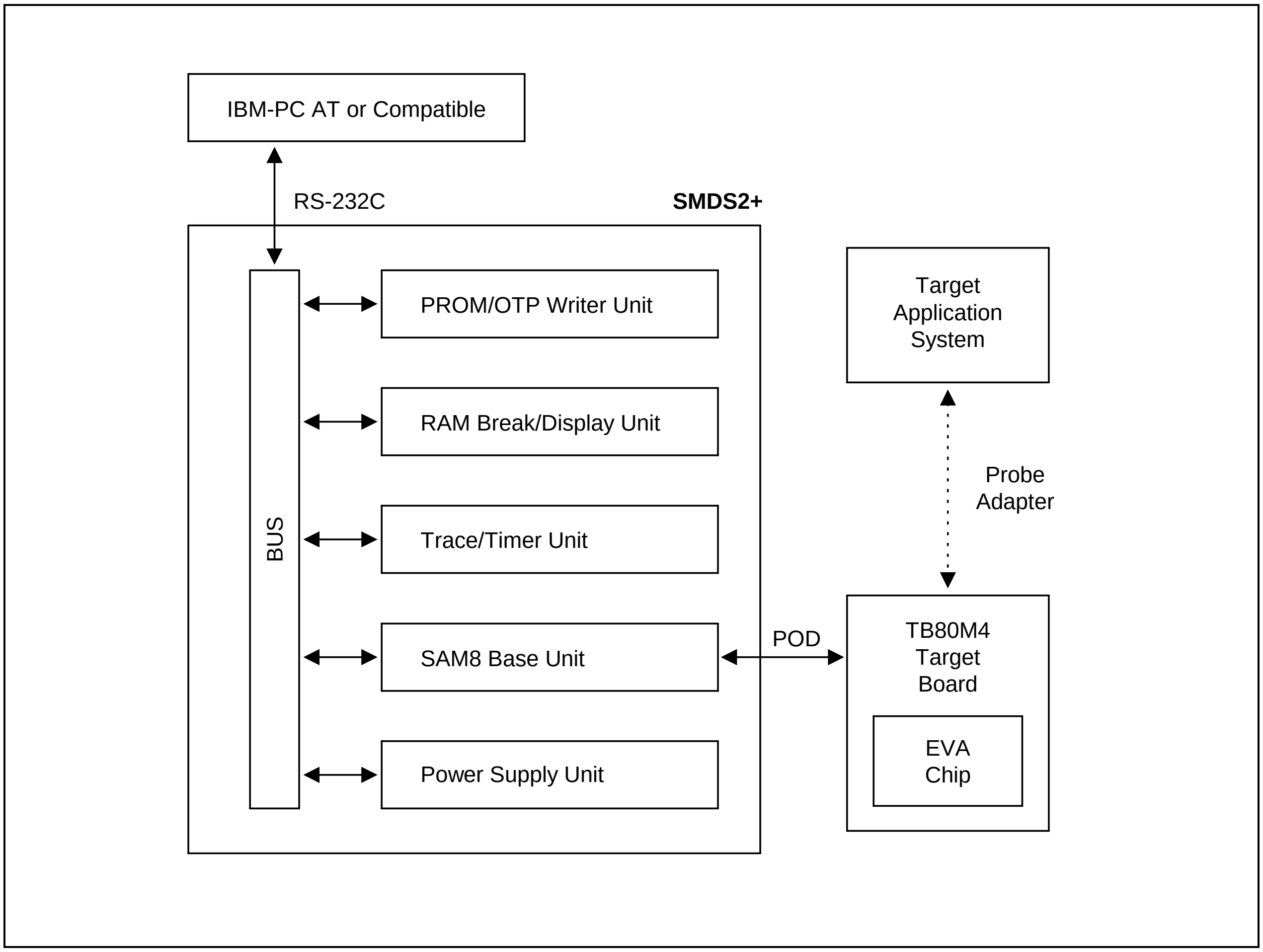


Figure 16-1. SMDS Product Configuration (SMDS2+)

TB80M4 TARGET BOARD

The TB80M4 target board is used for the S3C80M4/F80M4 microcontroller. It is supported with the SMDS2+.

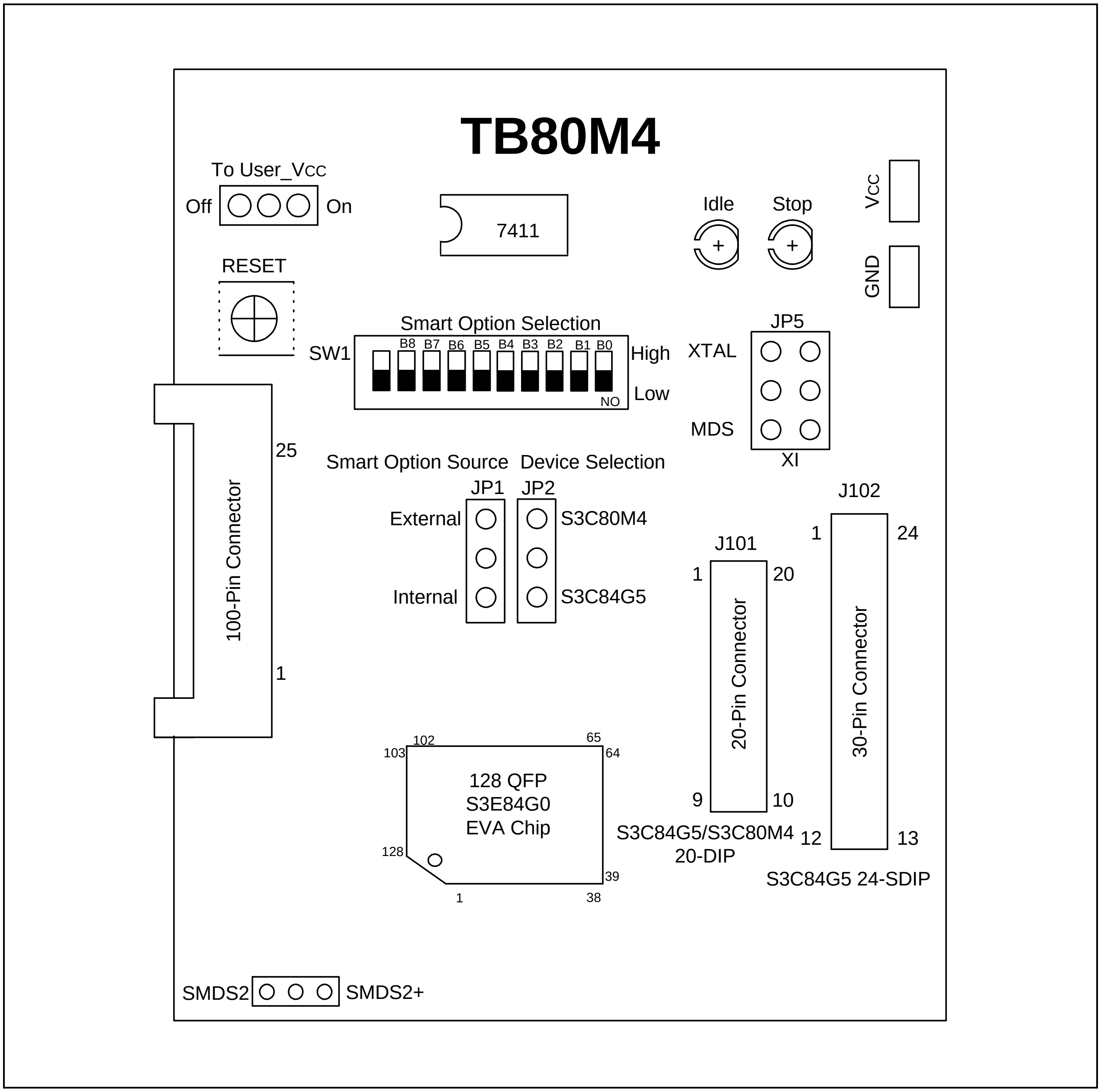
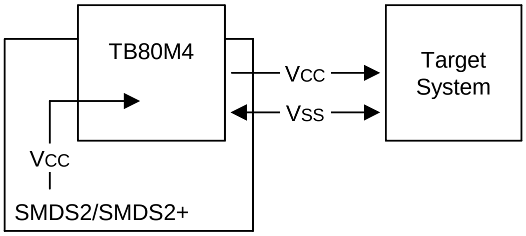
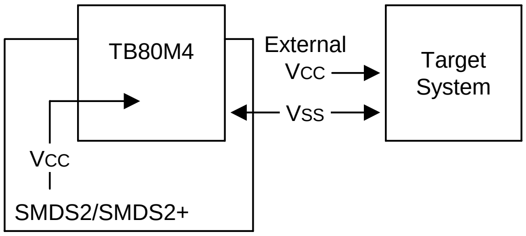


Figure 16-2. TB80M4 Target Board Configuration

Table 16-1. Power Selection Settings for TB80M4

"To User_Vcc" Settings	Operating Mode	Comments
To User_Vcc Off ○ ● ● On	 TB80M4 Vcc Vss Target System Vcc SMDS2/SMDS2+	The SMDS2/SMDS2+ supplies V_{CC} to the target board (evaluation chip) and the target system.
To User_Vcc Off ● ● ○ On	 TB80M4 External Vcc Vss Target System Vcc SMDS2/SMDS2+	The SMDS2/SMDS2+ supplies V_{CC} only to the target board (evaluation chip). The target system must have its own power supply.

NOTE: The following symbol in the "To User_Vcc" Setting column indicates the electrical short (off) configuration:

● ● ○

Table 16-2. Main-clock Selection Settings for TB80M4

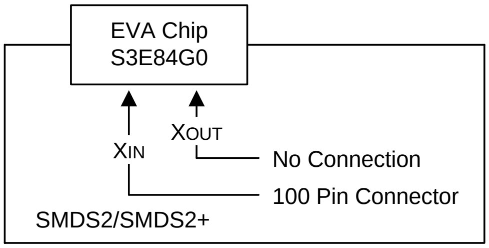
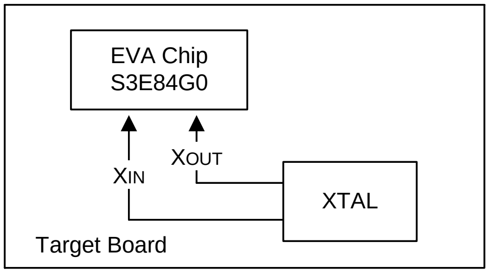
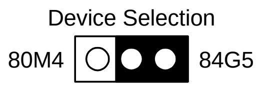
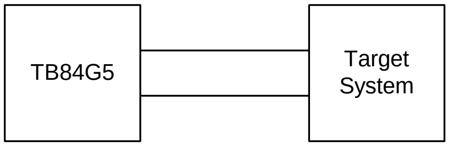
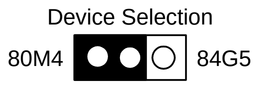
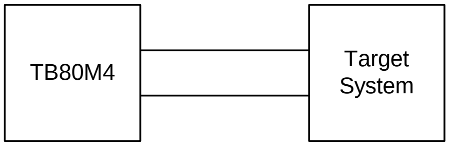
Main Clock Settings	Operating Mode	Comments
XIN XTAL ○ ● ● MDS ○ ● ●	 EVA Chip S3E84G0 XIN XOUT No Connection 100 Pin Connector SMDS2/SMDS2+	Set the XI switch to "MDS" when the target board is connected to the SMDS2/SMDS2+.
XIN XTAL ● ● ○ MDS ● ● ○		Set the XI switch to "XTAL" when the target board is used as a standalone unit, and is not connected to the SMDS2/SMDS2+.

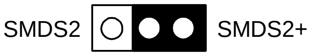
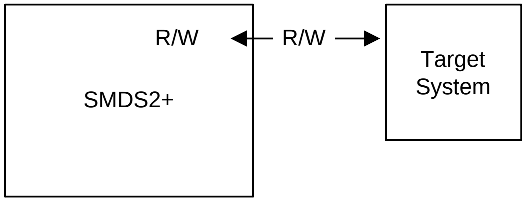
Table 16-3. Device Selection Settings for TB80M4

"Device Selection" Settings	Operating Mode	Comments
		Operate with TB84G5
		Operate with TB80M4

SMDS2+ SELECTION (SAM8)

In order to write data into program memory that is available in SMDS2+, the target board should be selected to be for SMDS2+ through a switch as follows. Otherwise, the program memory writing function is not available.

Table 16-4. The SMDS2+ Tool Selection Setting

"SMDS2+" Setting	Operating Mode
	

IDLE LED

The Yellow LED is ON when the evaluation chip (S3E84G0) is in idle mode.

STOP LED

The Red LED is ON when the evaluation chip (S3E84G0) is in stop mode.

Table 16-5. Smart Option Source Settings for TB80M4

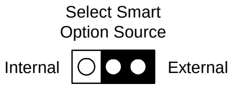
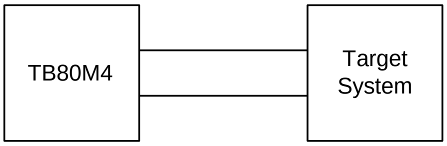
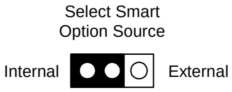
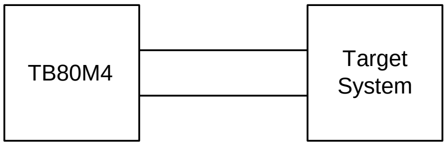
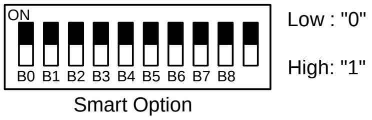
"Smart Option Source" Settings	Operating Mode	Comments
		Always must keep the External.
		Do not setting on left figure.

Table 16-6. Smart Option Switch Setting for TB80M4

"Smart Option" Setting	Comments
 ON B0 B1 B2 B3 B4 B5 B6 B7 B8 Low : "0" High: "1" Smart Option	Always must keep all High (“1”).

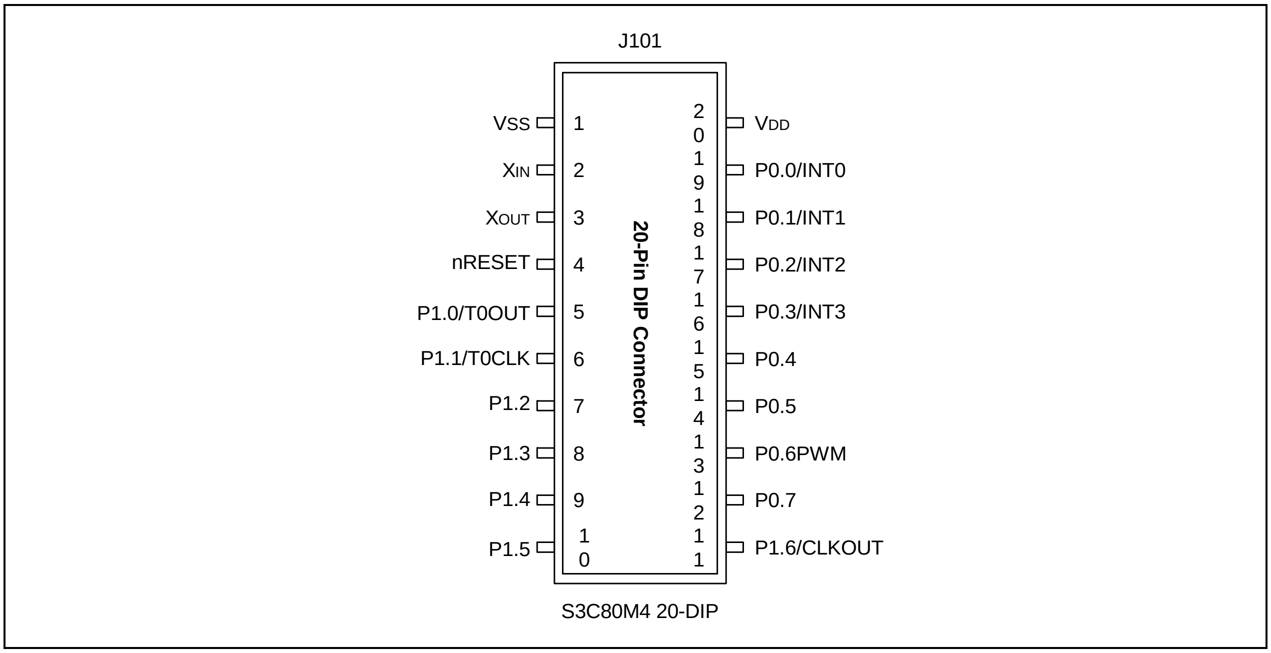


Figure 16-3. 20-Pin Connectors (J101) for TB80M4

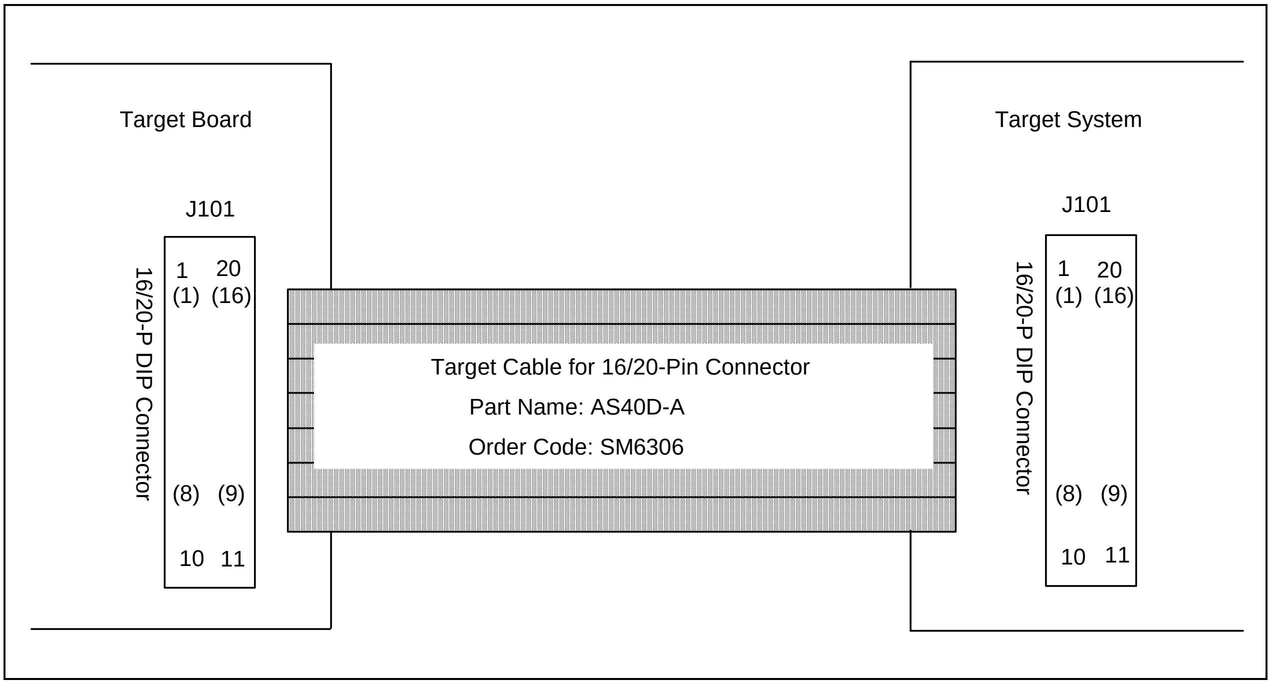


Figure 16-4. S3E80M0 Cables for 16/20-DIP Package

NOTES