Product Overview

Address Spaces

Addressing Modes

Control Registers

Interrupt Structure

Instruction Set

1 Product Overview

SAM8 PRODUCT FAMILY

Samsung's new SAM8 family 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.

A dual address/data bus architecture and a large number of bit- or nibble-configurable I/O ports provide a flexible programming environment for applications with varied memory and I/O requirements.

Timer/counters with selectable operating modes are included to support real-time operations. Many SAM8 microcontrollers have an external interface that provides access to external memory and other peripheral devices.

The 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 six CPU clocks) can be assigned to specific interrupt levels.

KS88C4400 MICROCONTROLLER

The KS88C4400 single-chip microcontroller is fabricated using a highly advanced CMOS process. Its design is based on the powerful SAM8 CPU core. Stop and Idle power-down modes were implemented to reduce power consumption. The size of the internal register file is logically expanded, increasing the addressable on-chip register space to 1040 bytes. A flexible yet sophisticated external interface is used to access up to 64-Kbytes of program and data memory.

Using the SAM8 modular design approach, the following peripherals were integrated with the SAM8 CPU core:

- Three configurable 8-bit general I/O ports
- One configurable 2-bit general I/O port
- One 8-bit n-channel, open-drain output port
- One 8-bit input port for A/D converter input or digital input
- Full-duplex serial data port with one synchronous and three asynchronous (UART) operating modes
- Two 8-bit timers with interval timer or PWM mode
- Two 16-bit timer/counters with four programmable operating modes
- Two programmable 8-bit PWM modules with corresponding output pins
- One 8-bit capture module with CAP input pin
- A/D converter with 8 selectable input pins

The KS88C4400 is a versatile microcontroller that is ideal for use in a wide range of general-purpose ROM-less applications such as CD-ROM/DVD-ROM drivers.

Figure 1–1. KS88C4400 Microcontroller

FeatureS

CPU

- SAM8 CPU core

Memory

- 1040-byte of internal register file

External Interface

- ROM-less operating mode only (EA pin = 5 V)
- 64-Kbyte external data memory area
- 64-Kbyte external program memory area
- Ports A, AD, and C are for external interface

Instruction Set

- 79 instructions
- IDLE and STOP instructions added for power-down modes

Instruction Execution Time

- 333 ns at 18 MHz f_{OSC} (minimum)

Interrupts

- 20 interrupt sources and 19 interrupt vectors
- Seven interrupt levels
- Fast interrupt processing

Timer/Counters

- Two 8-bit timers with interval timer or PWM mode (timers A and B)
- Two 16-bit timer/counters with four programmable operating modes (timers C and D)

General I/O

- Three 8-bit general I/O ports (ports 3, 4, and 5)
- One 8-bit n-channel, open-drain output port (port 6)
- One 8-bit input port (for ADC input or port 7 digital input)
- 2-bit general I/O port (port 2: P2.6 and P2.7)

Serial Port

- Full-duplex serial data port (UART)
- Four programmable operating modes

PWM and Capture

- Two output channels (PWM0, PWM1)
- 8-bit resolution with 2-bit prescaler
- 70.305-kHz frequency (18-MHz CPU clock)
- Capture module with CAP input pin

Analog-to-Digital Converter

- Eight analog input pins
- 8-bit conversion resolution
- 10.66-µs conversion speed (18-MHz CPU clock)

Operating Temperature Range

- – 20°C to + 85°C

Operating Voltage Range

- 4.5 V to 6.0 V

Package Type

- 80-pin QFP, 80–pin TQFP

Block Diagram

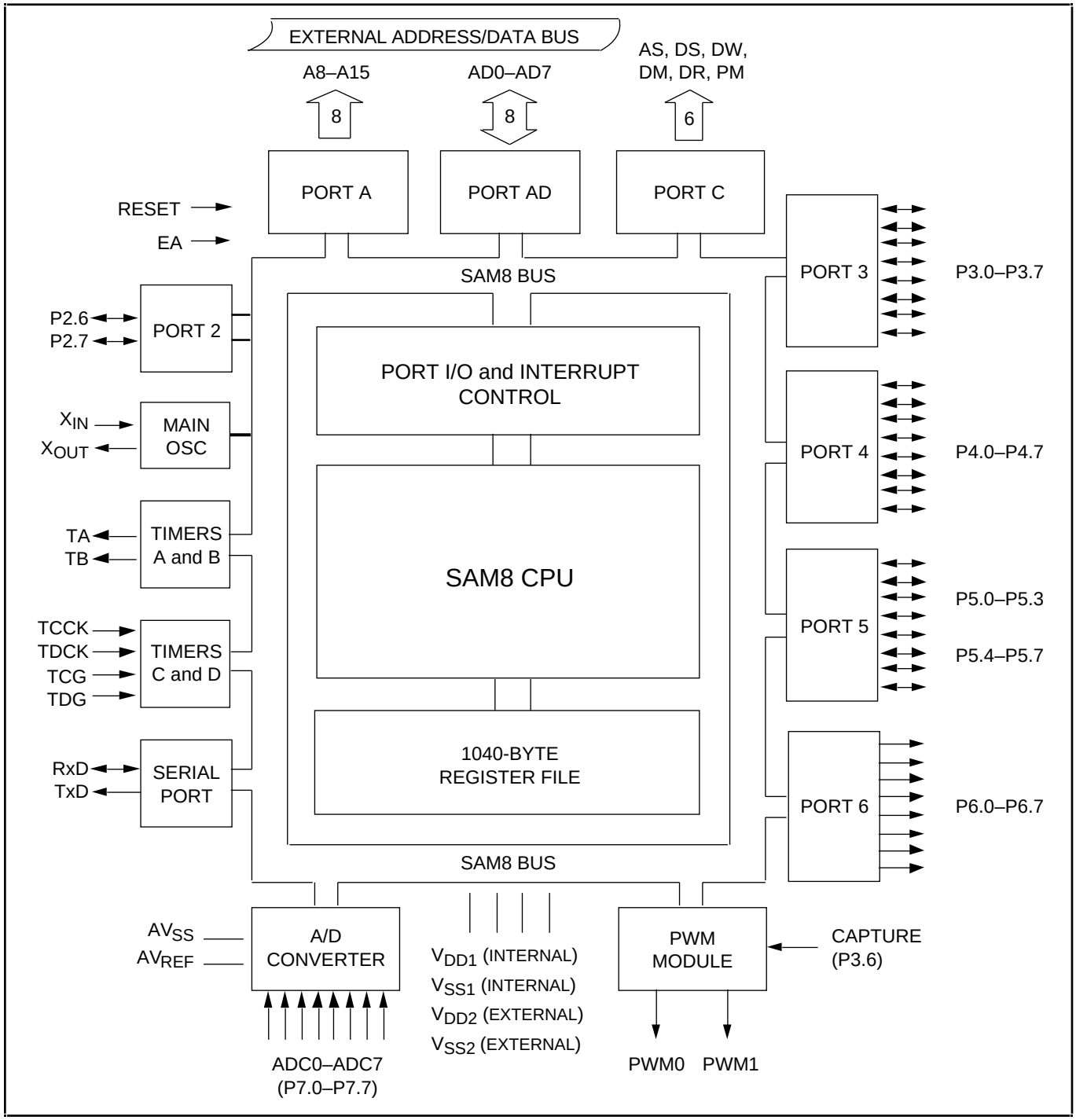
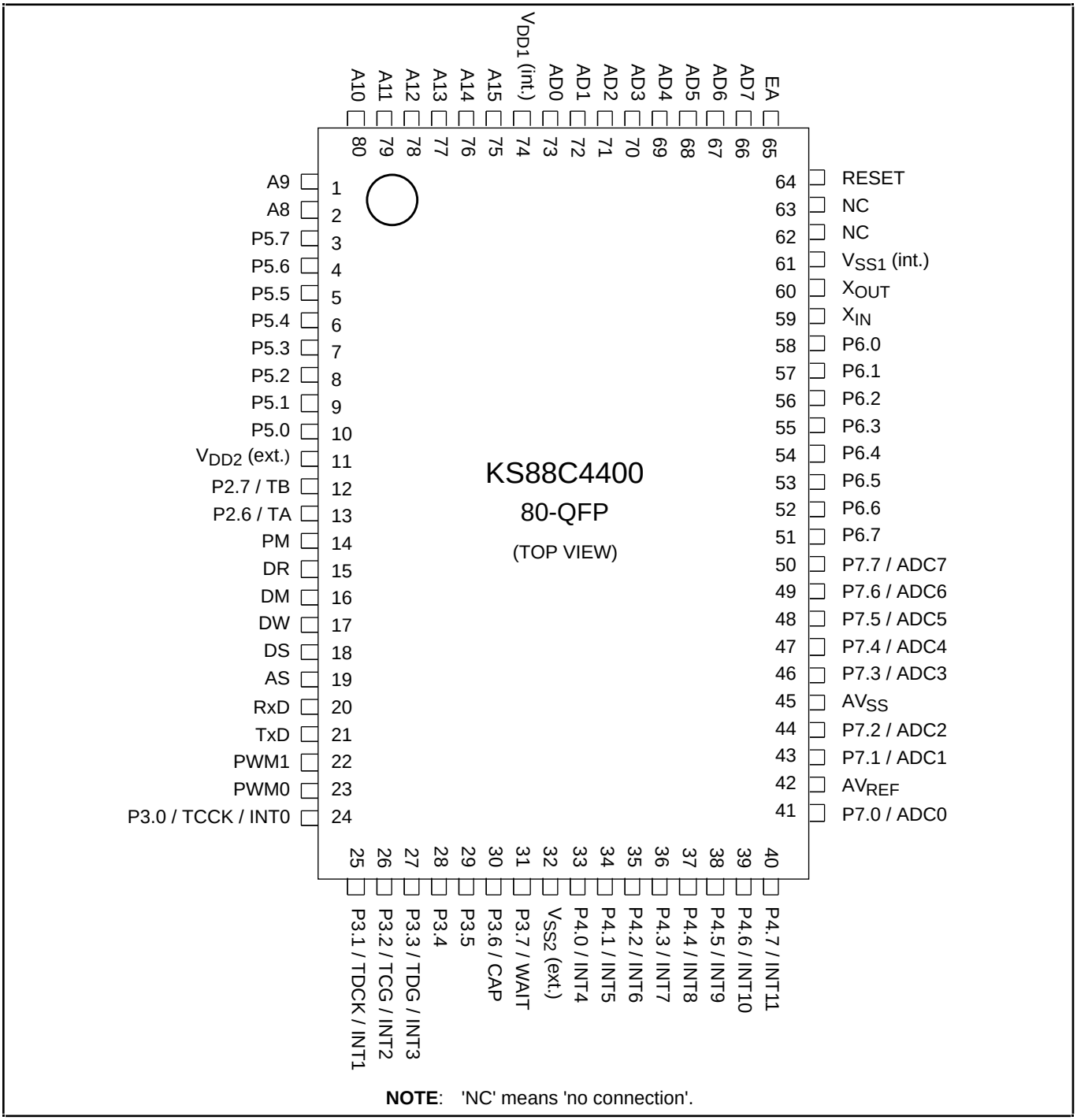


Figure 1–2. KS88C4400 Block Diagram

Pin Assignments



NOTE: 'NC' means 'no connection'.

Figure 1–3. KS88C4400 Pin Assignments

Pin Assignments (Continued)

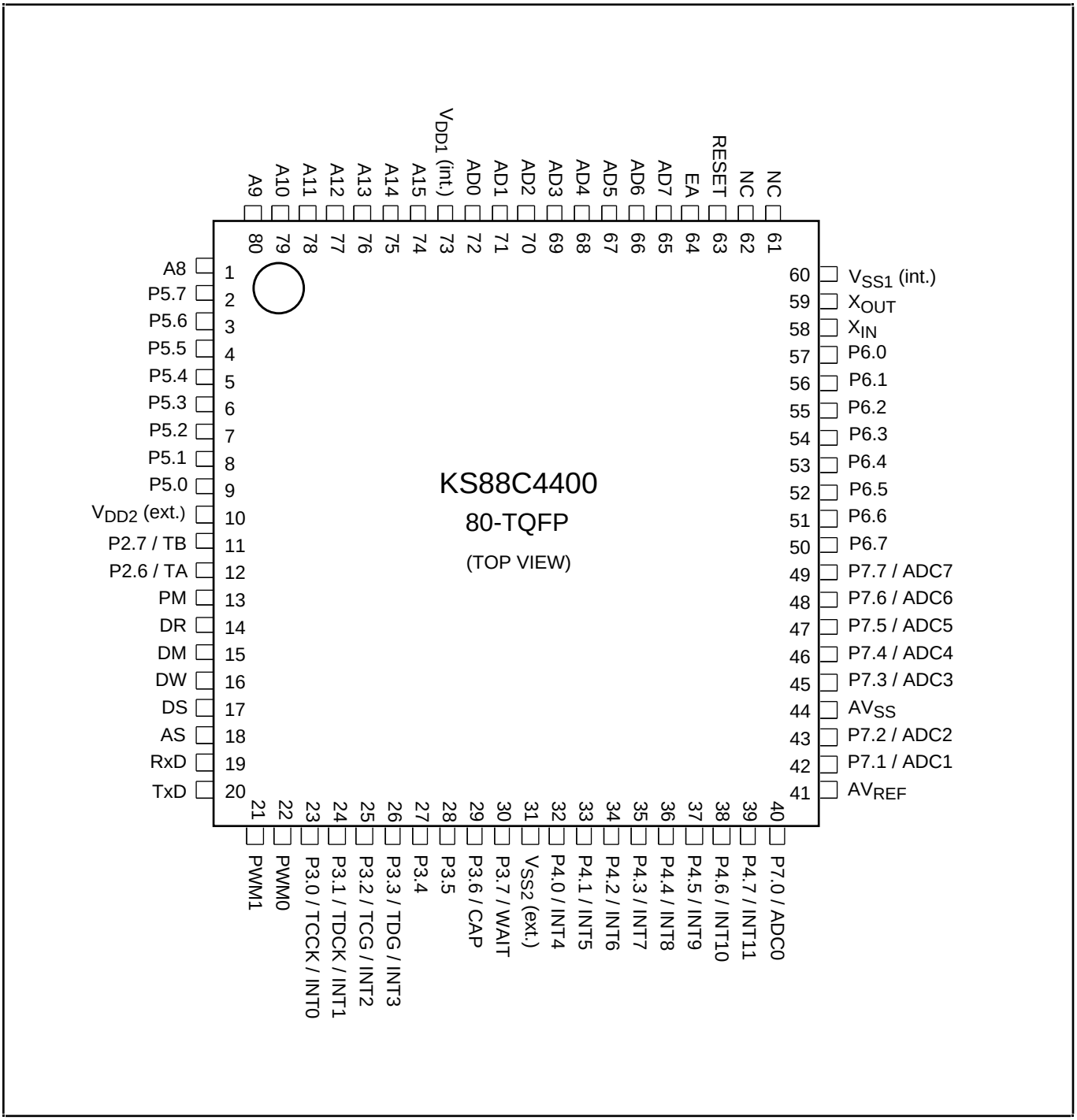


Figure 1–4. KS88C4400 Pin Assignments

Pin Descriptions

Table 1–1. KS88C4400 Pin Descriptions

Pin Name	Pin Type	Pin Description	Circuit Type	QFP Pin Number	Share Pins
A8–A15	O	Address High output port. Port A pins are for external interface address lines A8–A15.	7	2, 1, 80–75	–
AD0–AD7	I/O	Address Low and data port. Port AD pins are for external interface address/data lines AD0–AD7.	9	73–66	–
AS, DS, DW, DM, DR, PM	O	6-bit output port for external interface control signals. Port C pins support the following bus control signals: AS: address strobe DS: data strobe DW: data memory write DM: data memory select DR: data memory read PM: program memory select	7	19–14	–
P2.6, P2.7	I/O	General I/O port with bit programmable pins. Schmitt trigger input or push-pull output. Alternatively used as output pins for timer A and timer B: P2.6 / timer A output P2.7 / timer B output	5	13, 12	TA, TB
P3.0–P3.7	I/O	General I/O port with bit programmable pins. Schmitt trigger input or push-pull output with software assignable pull-ups. Input or output mode is selectable by software. P3.0–P3.3 are alternately used as inputs for external interrupts INT0–INT3, respectively (with noise filters and interrupt control): P3.0 / timer C clock input (TCCK) / INT0 P3.1 / timer D clock input (TDCK) / INT1 P3.2 / timer C gate input (TCG) / INT2 P3.3 / timer D gate input (TDG) / INT3 P3.6 / Capture data input (CAP) P3.7 / WAIT for slow memory interface	4	24–31	(See pin description)
P4.0–P4.7	I/O	General I/O port with bit programmable pins. Schmitt trigger input or push-pull, open-drain output with software assignable pull-ups. Input or output mode is selectable by software. P4.0–P4.7 can alternately be used as inputs for external interrupts INT4–INT11, respectively (with noise filters and interrupt control)	4	33–40	INT4 – INT11

Table 1–1. KS88C4400 Pin Descriptions (Continued)

Pin Name	Pin Type	Pin Description	Circuit Type	QFP Pin Number	Share Pins
P5.0–P5.7	I/O	General I/O port with nibble programmable pins. Schmitt trigger input or push-pull, open-drain output with software assignable pull-ups. Input or output mode is selectable by software.	3	10–3	–
P6.0–P6.7	O	N-channel, open-drain output port can withstand high current loads up to 9 volts.	8	58–51	–
ADC0–ADC7	I	Analog input pins for A/D converter module. Alternatively used as general-purpose digital input port 7.	2	41, 43–44, 46–50	P7.0–P7.7
AV_{REF}, AV_{SS}	–	A/D converter reference voltage and ground	–	42, 45	–
RxD	I/O	Serial data RxD pin for receive input and transmit output (mode 0).	6	20	–
TxD	O	Serial data TxD pin for transmit output and shift clock input (mode 0).	7	21	–
PWM0, PWM1	O	Pulse width modulation output pins	7	23, 22	–
TA, TB	O	Output pins for timer A and timer B	5	13, 12	P2.6, P2.7
INT0–INT11	I	External interrupt input pins	4	24–27, 33–40	P3.0–P3.3, P4.0–P4.7
TCCK, TDCK	I	External clock input for timer C and timer D	4	24, 25	P3.0, P3.1
TCG, TDG	I	Gate input pins for timer C and timer D	4	26, 27	P3.2, P3.3
CAP	I	Capture data input for PWM module	4	30	P3.6
WAIT	I	Input pin for the slow memory timing signal from the external interface	4	31	P3.7
RESET	I	System reset pin (pull-up resistor: 220 kΩ)	1	64	–
EA	I	External access (EA) pin with two modes: 0V: Not allowed for KS88C4400 5 V: Normal ROM-less operation (external interface) 9–10 V input: Factory test mode	–	65	–
V_{DD1}, V_{SS1}	–	Power input pins for CPU operation (internal)	–	74, 61	–
V_{DD2}, V_{SS2}	–	Power input pins for port output (external)	–	11, 32	–
X_{IN}, X_{OUT}	–	Main oscillator pins	–	59, 60	–
NC, NC	–	No connection pins (connect to V_{SS})	–	62, 63	–

NOTE VDD1 must be connected to VDD2 in users application circuit, VSS1 & VSS2 also.

Pin Circuits

Table 1–2. Pin Circuit Assignments for the KS88C4400

Circuit Number	Circuit Type	KS88C4400 Assignments
1	Input	RESET pin
2	Input	A/D converter input pins, ADC0–ADC7
3	I/O	Port 5
4	I/O	Ports 3 and 4, TCCK, TDCK, TCG, TDG, CAP, WAIT, INT0–INT11
5	I/O	Port 2 (P2.6/TA and P2.7/TB)
6	I/O	Serial port RxD pin
7	Output	Port A, port C, serial port TxD pin, PWM0, and PWM1
8	Output	Port 6 (n-channel, open-drain output with high-current capability)
9	I/O	Port AD

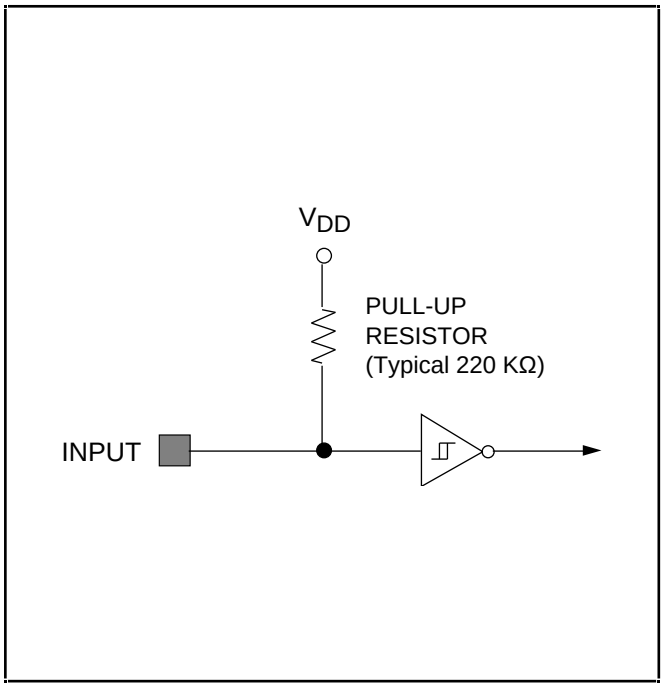


Figure 1–5. Pin Circuit Type 1 (RESET**)**

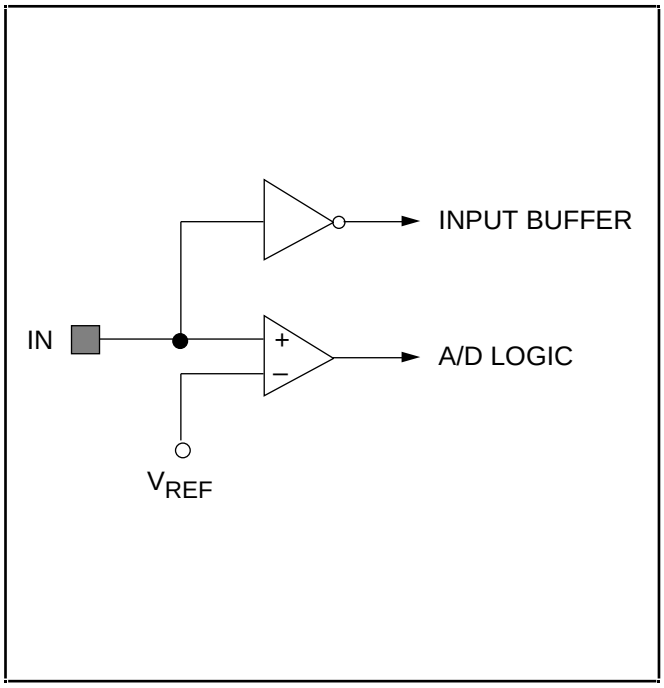


Figure 1–6. Pin Circuit Type 2 (ADC0–ADC7)

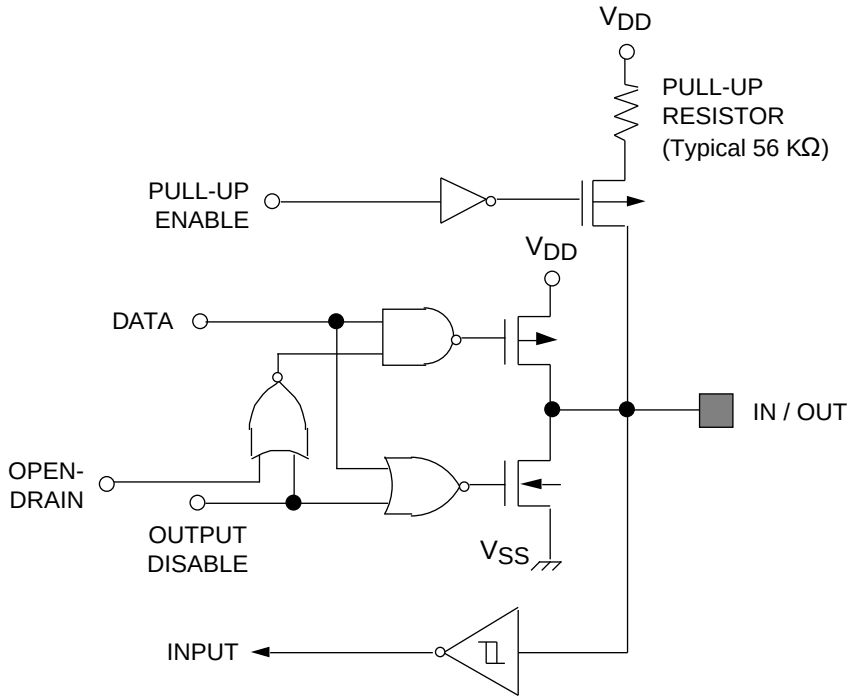


Figure 1–7. Pin Circuit Type 3 (Port 5)

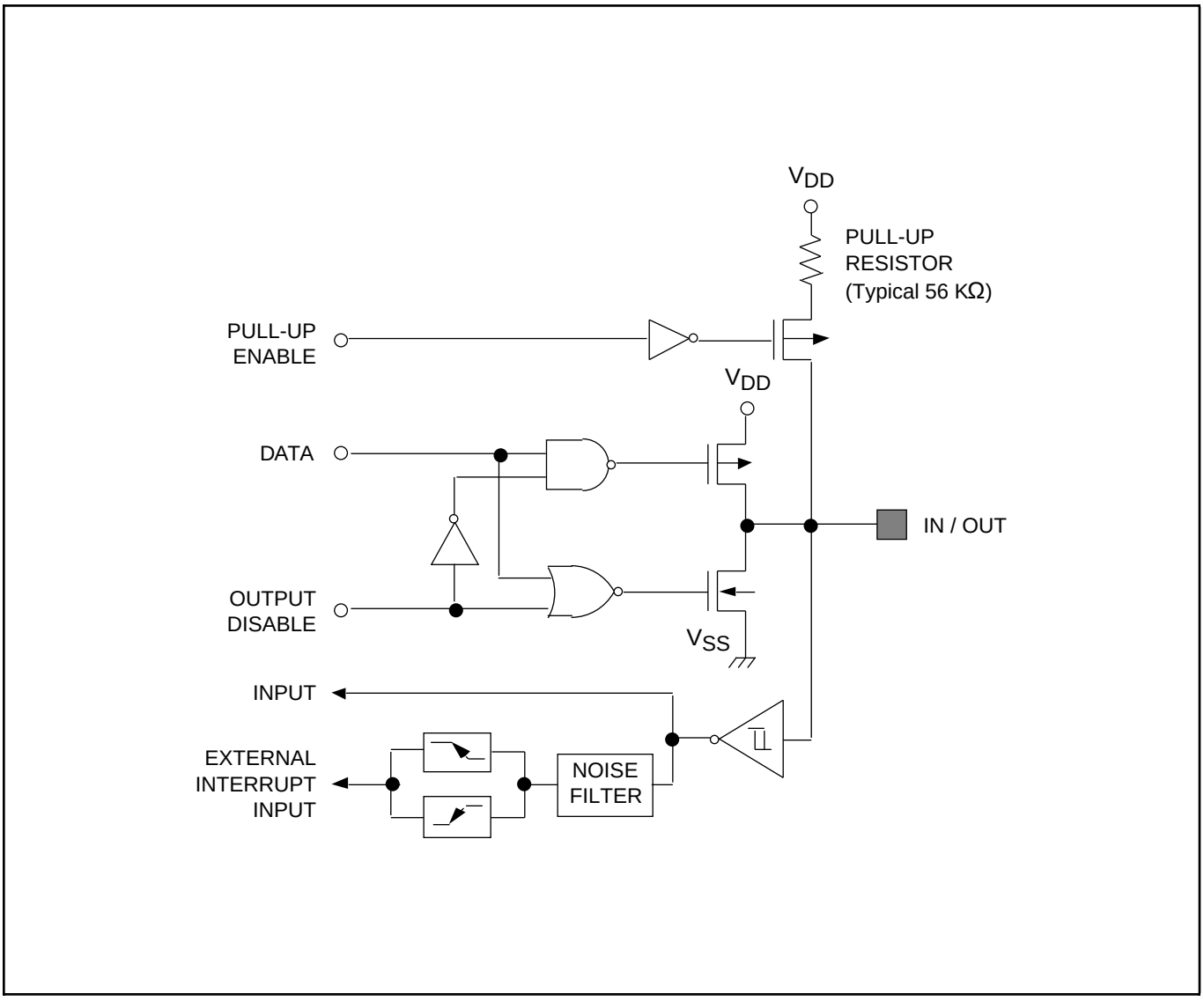


Figure 1–8. Pin Circuit Type 4
(Ports 3 and 4, TCCK, TDCK, TCG, TDG, CAP, WAIT, **INT0–INT11)**

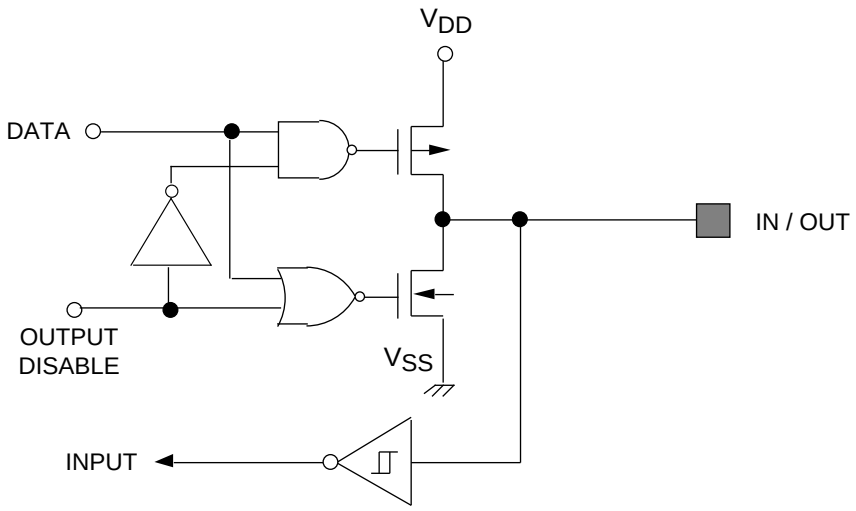


Figure 1–9. Pin Circuit Type 5 (P2.6/TA and P2.7/TB)

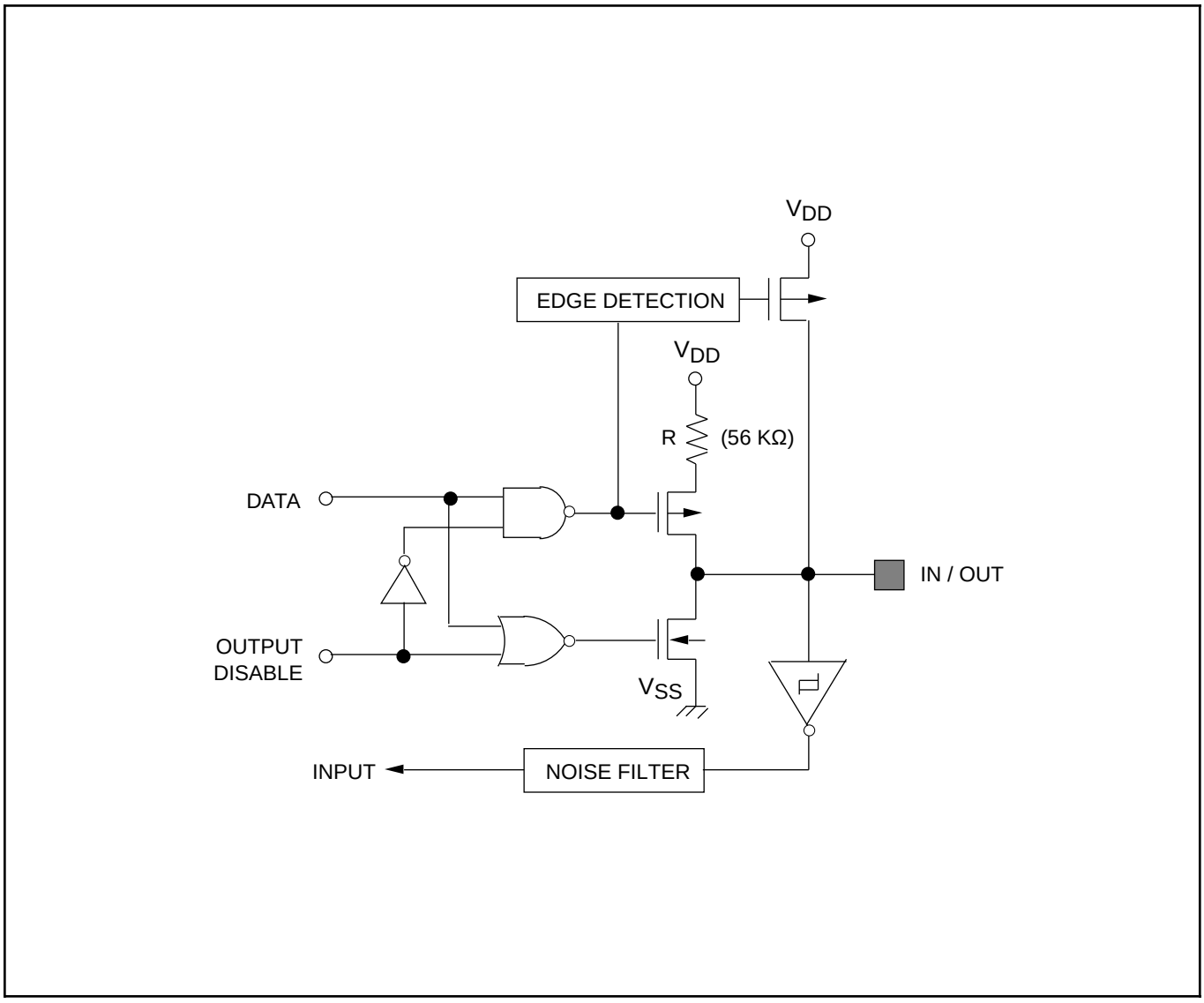


Figure 1–10. Pin Circuit Type 6 (Serial RxD Pin)

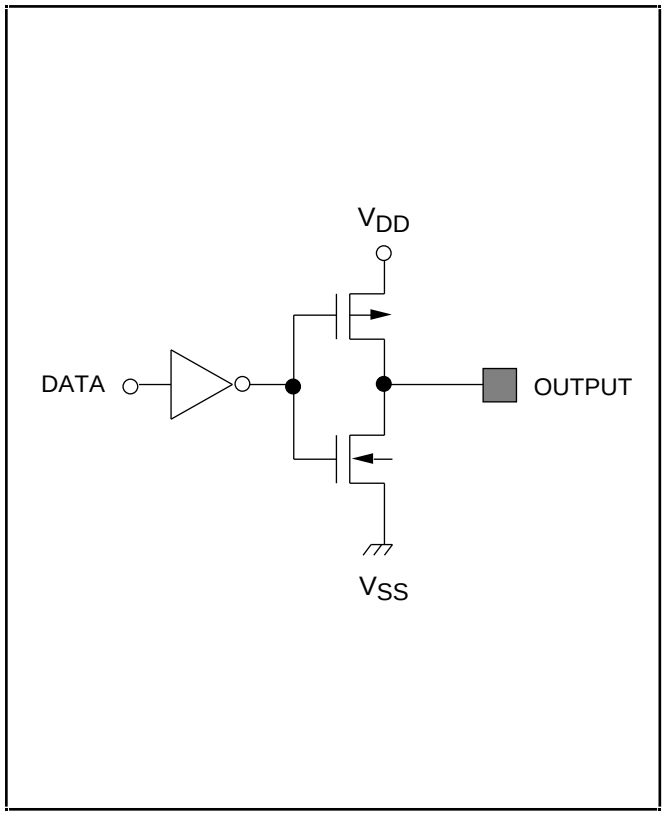
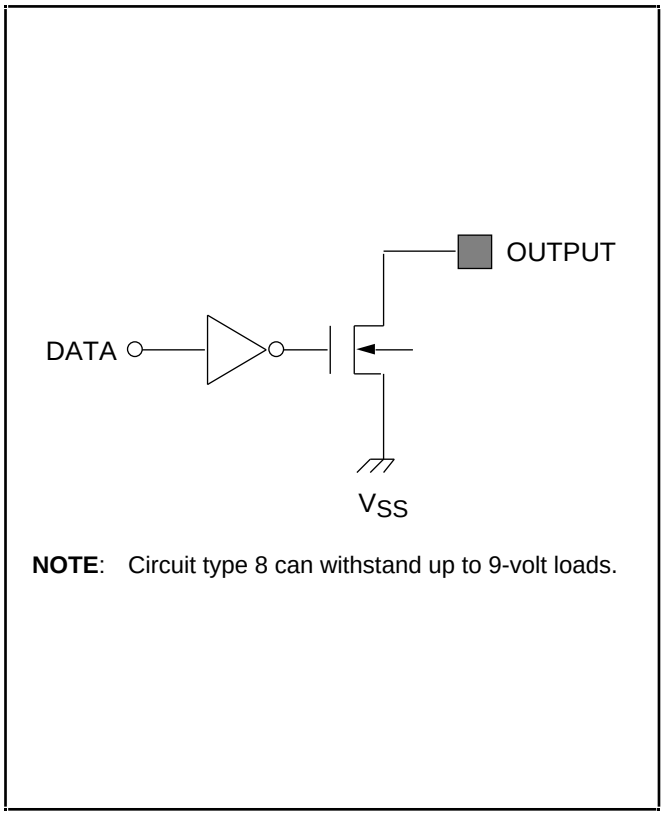


Figure 1–11. Pin Circuit Type 7 (Port A, port C, serial TxD Pin, PWM0, PWM1)



NOTE: Circuit type 8 can withstand up to 9-volt loads.

Figure 1–12. Pin Circuit Type 8 (Port 6)

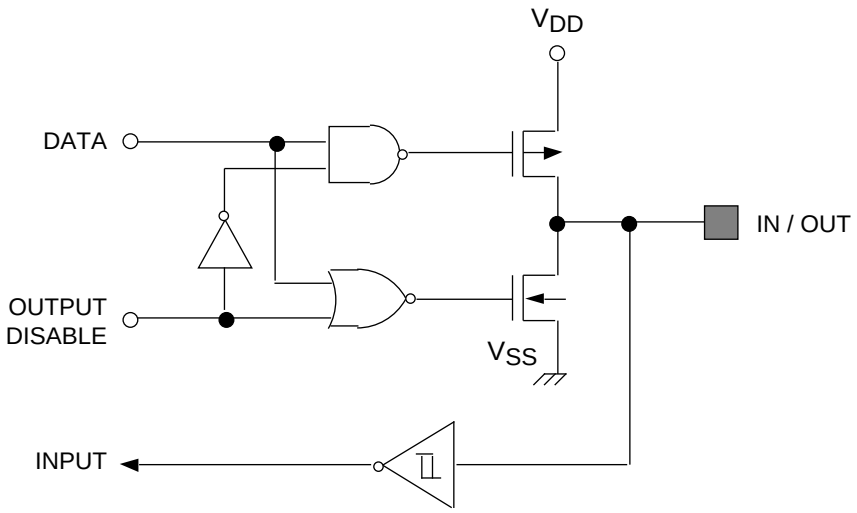


Figure 1–13. Pin Circuit Type 9 (Port AD)

2 Address spaces

OVERVIEW

The KS88C4400 microcontroller has three kinds of address space:

— External program memory
— External data memory
— Internal register file

A 16-bit address bus supports both external program memory and external data memory operations. Special instructions and related internal logic determine when the 16-bit bus carries addresses for external program memory or for external data memory locations. SAM8 bus architecture therefore supports up to 64 K bytes of program memory (ROM). Using the external interface, you can address up to 64 K bytes of program memory *and* 64 K bytes of data memory simultaneously. These spaces can be combined or kept separate.

The KS88C4400 microcontroller has 1088 registers in its internal register file.A separate 8-bit register bus carries addresses and data between the CPU and the internal register file. The most of these registers can serve as either a source or destination address, or as accumulators for data memory operations. Special 48 bytes of the register file are used for system and peripheral control functions.

ROM-less Operating Mode

The KS88C4400 microcontroller is implemented as a ROM-less device. That is, its entire program memory space (up to 64 K bytes) is configured externally. The address range of the external program memory space is, therefore, 0H–FFFFH.

You select normal (ROM-less) operating mode by applying a constant 5 volts to the EA pin *before* a power-on or reset operation. This automatically configures the external interface and directs all program memory accesses onto the 16-bit external address/data bus.

NOTE

If you connect the EA pin to V_{DD}, a power-on-reset will automatically select the correct ROM-less (external ROM) operating mode. The setting EA = 0 V is not allowed.

The external interface consists of the following pins:

— Port A (address) pins A8–A15

— Port AD (address/data) pins AD0–AD7

— Port C (control) pins for the bus control signals DM, PM, DR, DW, DS, and AS

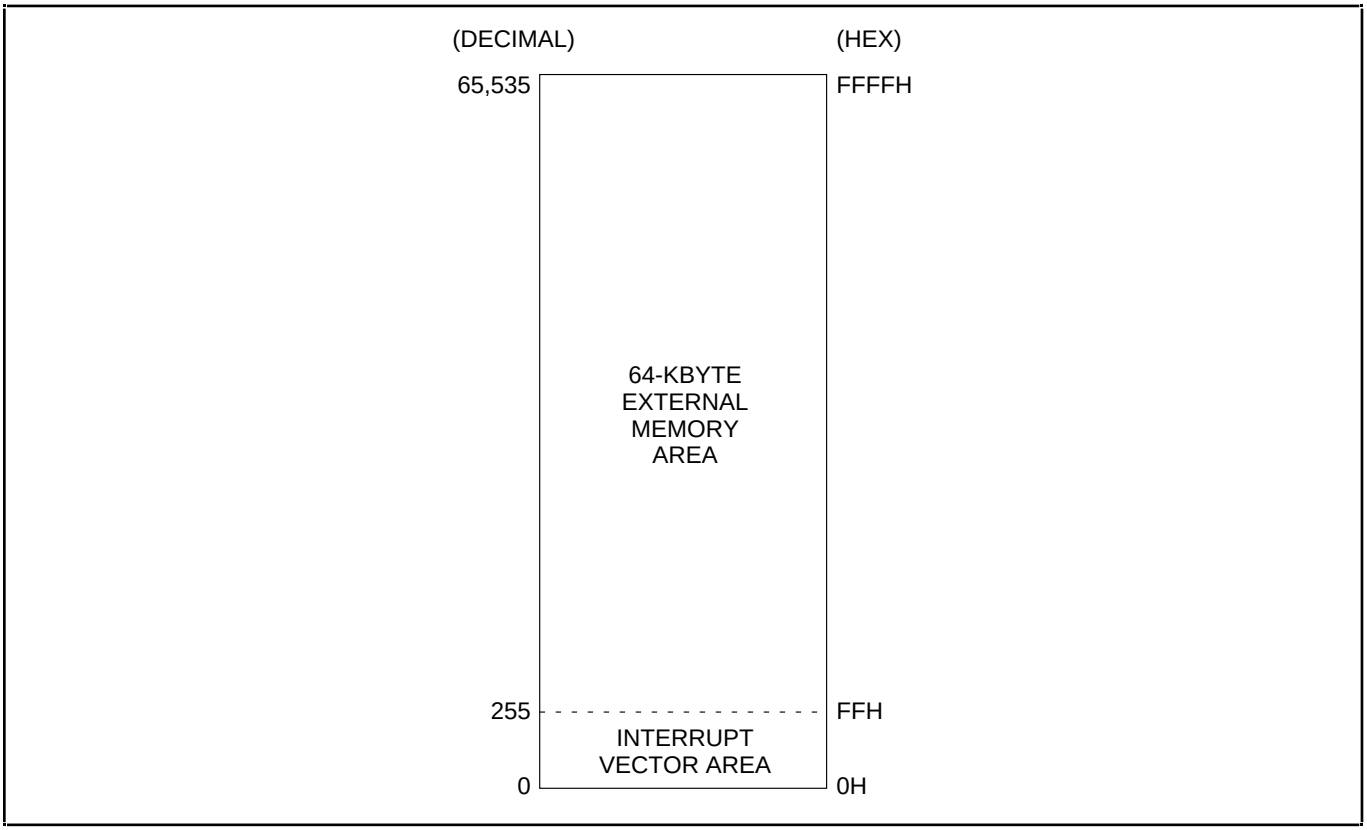


Figure 2–1. External Program Memory Address Space

Register Architecture

The physical 256 bytes space is logically partitioned into four 256 byte pages, giving a total of 1024 general purpose registers.

The upper 64 bytes of the KS88C4400's internal register file is logically expanded into two 64-byte areas, called *set 1* and *set 2*. Set1 is further partitioned into two 32-byte register banks (bank0 and bank1) and a 32-byte common area.

Set1 register locations are always addressable whenever one of the four pages are selected. Set1 locations can be addressed using indirect addressing modes only.

In addition to sets 1 and 2 and banks 0 and 1, four logical register pages are implemented, *page 0 – page 3*. The 8-bit register bus can address up to 256 bytes (0H–FFH) in any of the four pages.

The total register file area is 1120 bytes (256 x 4 pages + set1: 64 bytes + 32 bytes). Since only locations FFH–F9H are mapped in set1, bank1, the KS88C4400 register file has 1088 addressable 8-bit registers.

Of the 1088 registers, 13 bytes are CPU and system control register, 35 bytes are for peripheral control and data, 16 bytes are used as a shared working register space, and there are 1024 general-purpose registers.

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

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

Table 2–1. Register Type Summary

Register Type	Number of Bytes
CPU and system control registers	13
Peripheral, I/O, and clock control/data registers	35
Reserved working register area	16
General-purpose registers	1024
Total Addressable Bytes	1088

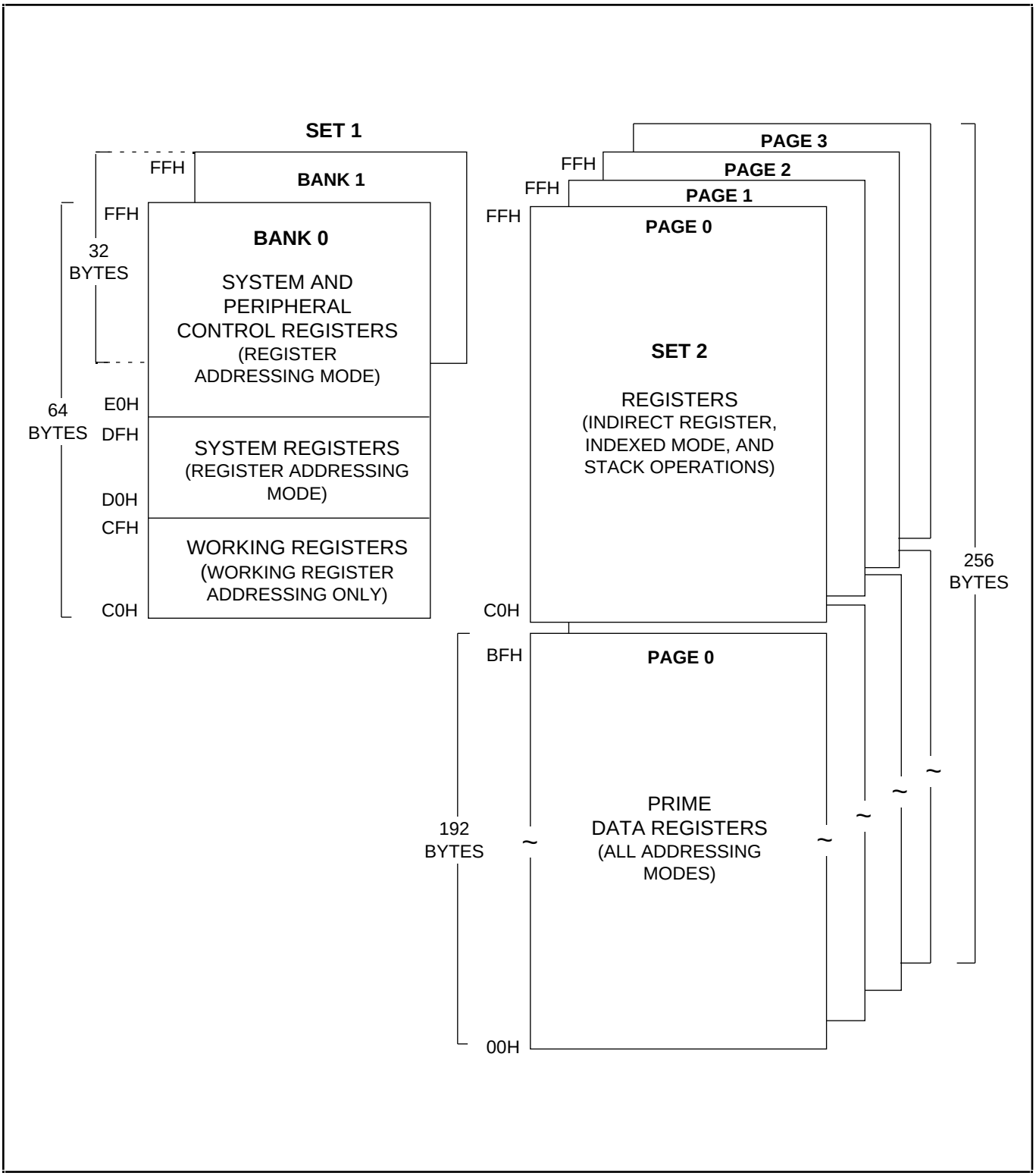


Figure 2–2. Internal Register File Organization

Register Page Pointer (PP)

In the KS88C4400, the physical area of the internal register file is logically expanded by the additional of four register pages. Page addressing is controlled by the register page pointer (PP, DFH). See Figure 2–3.

A reset clears the register page pointer value to zero, selecting page 0 addressing. To select another page, you manipulate the page selection control bits, PP.0–PP.1.

Whenever you select a different page, the current 256-byte address area (0H–FFH) is logically switched with the address range of the new page.

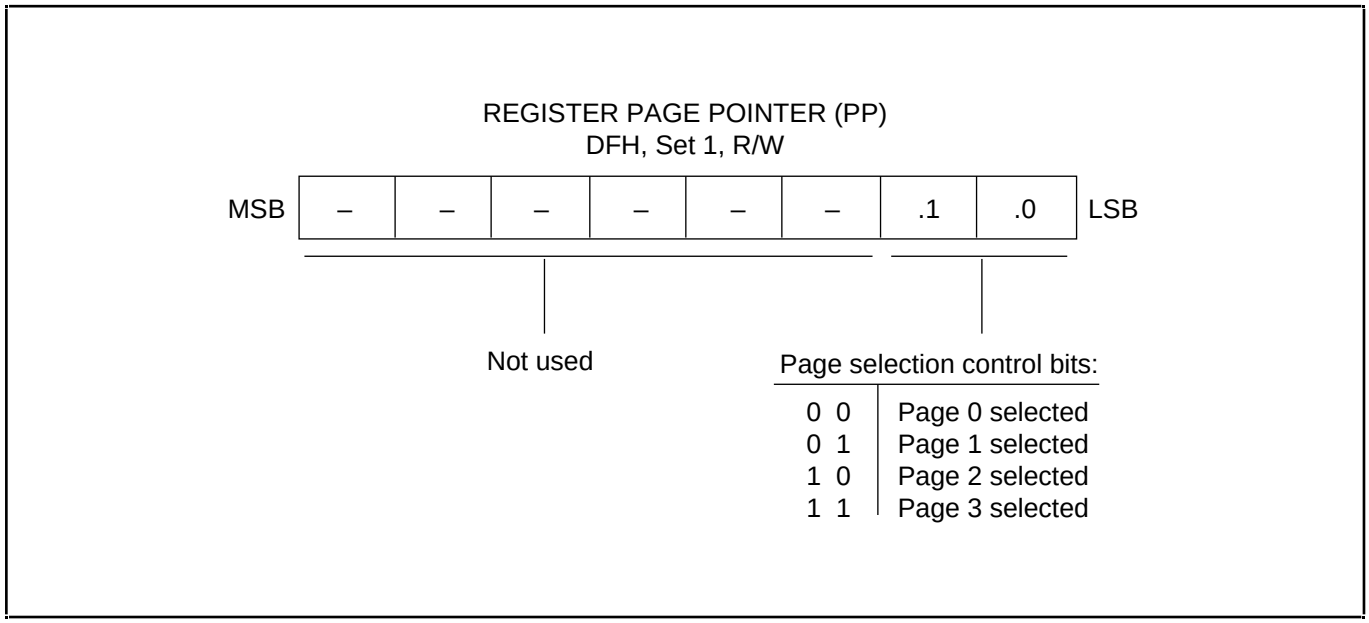


Figure 2–3. Register Page Pointer (PP)

REGISTER SET 1

The term *set 1* refers to the upper 64 bytes of the register file, locations C0H–FFH. This area can be accessed at any time, regardless of which page is currently selected.

The upper 32-byte area of this 64-byte space is divided into two 32-byte register banks, called *bank 0* and *bank 1*. You use the select register bank instructions, SB0 or SB1, to address one bank or the other. A reset operation automatically selects bank 0 addressing.

The lower 32-byte area of set 1 is not banked. This area contains 16 bytes for mapped system registers (D0H–DFH) and a 16-byte common area (C0H–CFH) for working register addressing.

Registers in set 1 are directly accessible at all times using the Register addressing mode. The 16-byte working register area can only be accessed using working register addressing, however.

Working register addressing is a function of Register addressing mode (see Section 3, "Addressing Modes," for more information).

Register Set 2

The same 64-byte physical space that is used for set 1 register locations C0H–FFH is logically duplicated to add another 64 bytes. This expanded area of the register file is called *set 2*. For the KS88C4400, the set 2 address range (C0H–FFH) is accessible on pages 0–3.

The logical division of set 1 and set 2 is maintained by means of addressing mode restrictions: While you can access set 1 using Register addressing mode only, you can only use Register Indirect addressing mode or Indexed addressing mode to access set 2.

PRIME REGISTER SPACE

The lower 192 bytes (00H–BFH) of the KS88C4400's four 256-byte register pages is called *prime register area.* Prime registers can be accessed using any of the seven addressing modes (see Section 3, "Addressing Modes").

The prime register area on page 0 is immediately addressable following a reset. In order to address prime registers on pages 1, 2, or 3, you must set the register page pointer (PP) to the appropriate source and destination values.

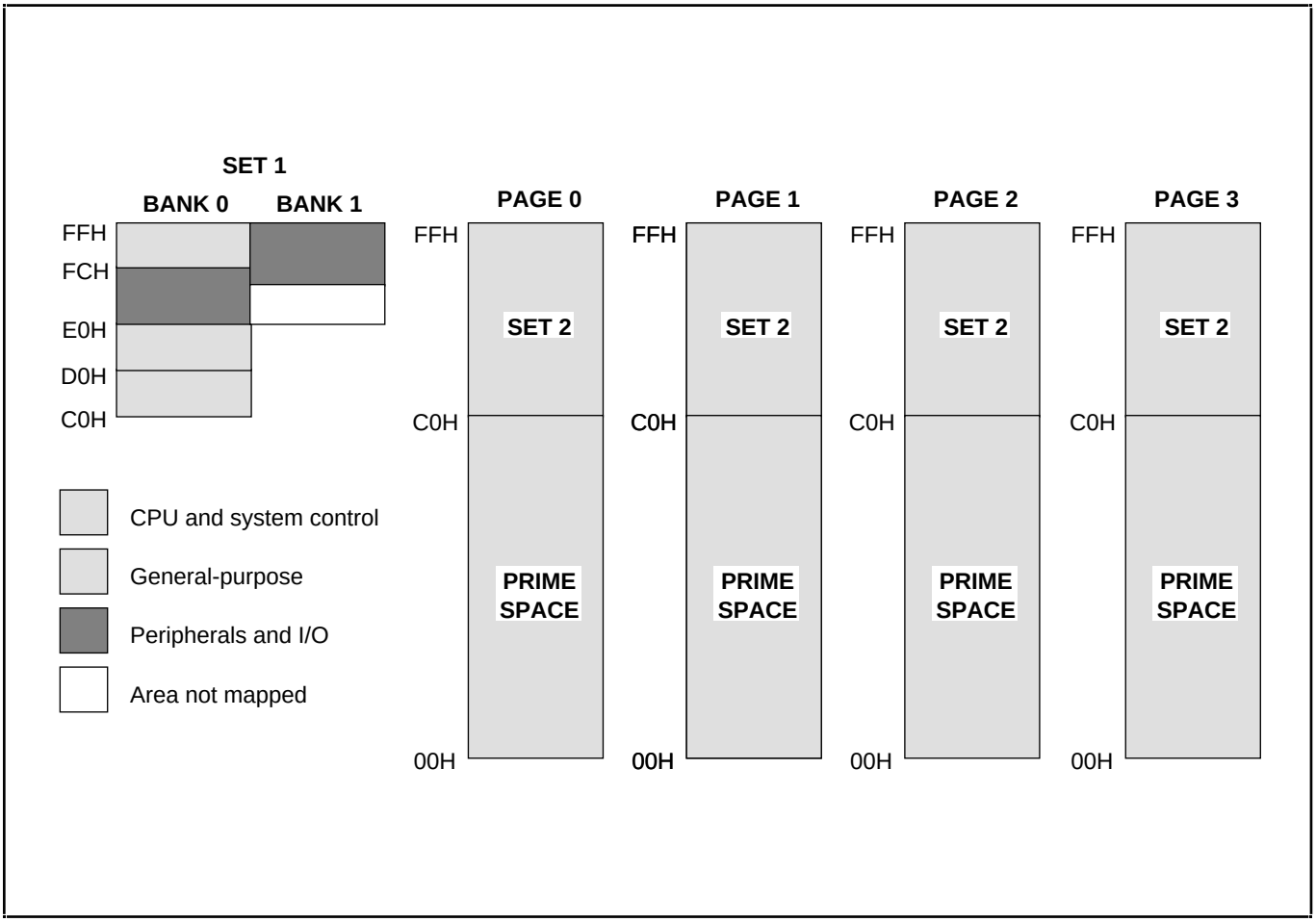


Figure 2–4. Map of Set 1, Set 2, and Prime Register Spaces

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 viewed by the programmer as consisting of 32 8-byte register groups or "slices."

Each slice consists 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 for 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 of 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).

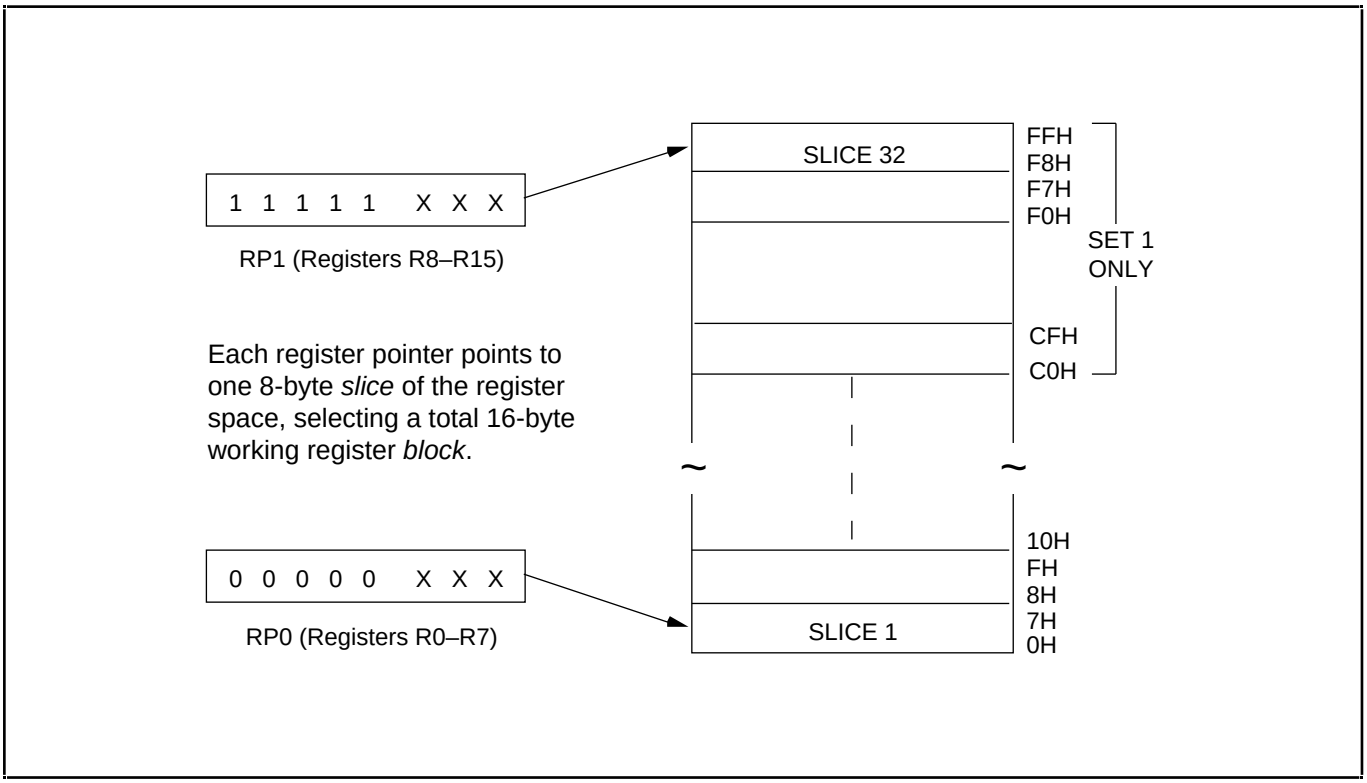


Figure 2–5. 8-Byte Working Register Areas (Slices)

Using the Register Pointers

Register pointers RP0 and RP1 are mapped to addresses D6H and D7H in set 1. They 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 locations that are pointed to by the register pointers. Please note that you cannot use the register pointers to select working register area in set 2, C0H–FFH, because these locations are accessible 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, we recommend that RP0 point to the "lower" slice and RP1 point to the "upper" slice (see Figure 2–6).

In some cases, you may need 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 the either of the two 8-byte slices in the working register block, definition of the working register area is very flexible.

☞ 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
```

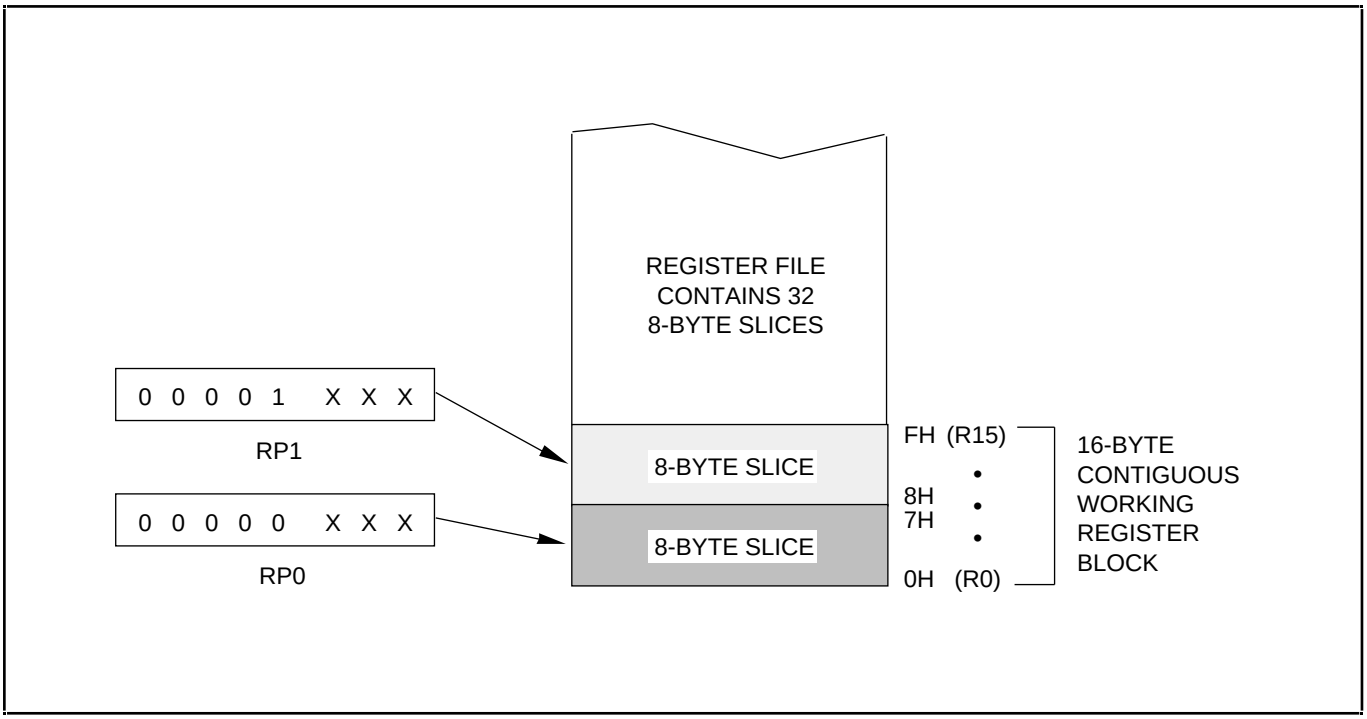


Figure 2–6. Contiguous 16-Byte Working Register Block

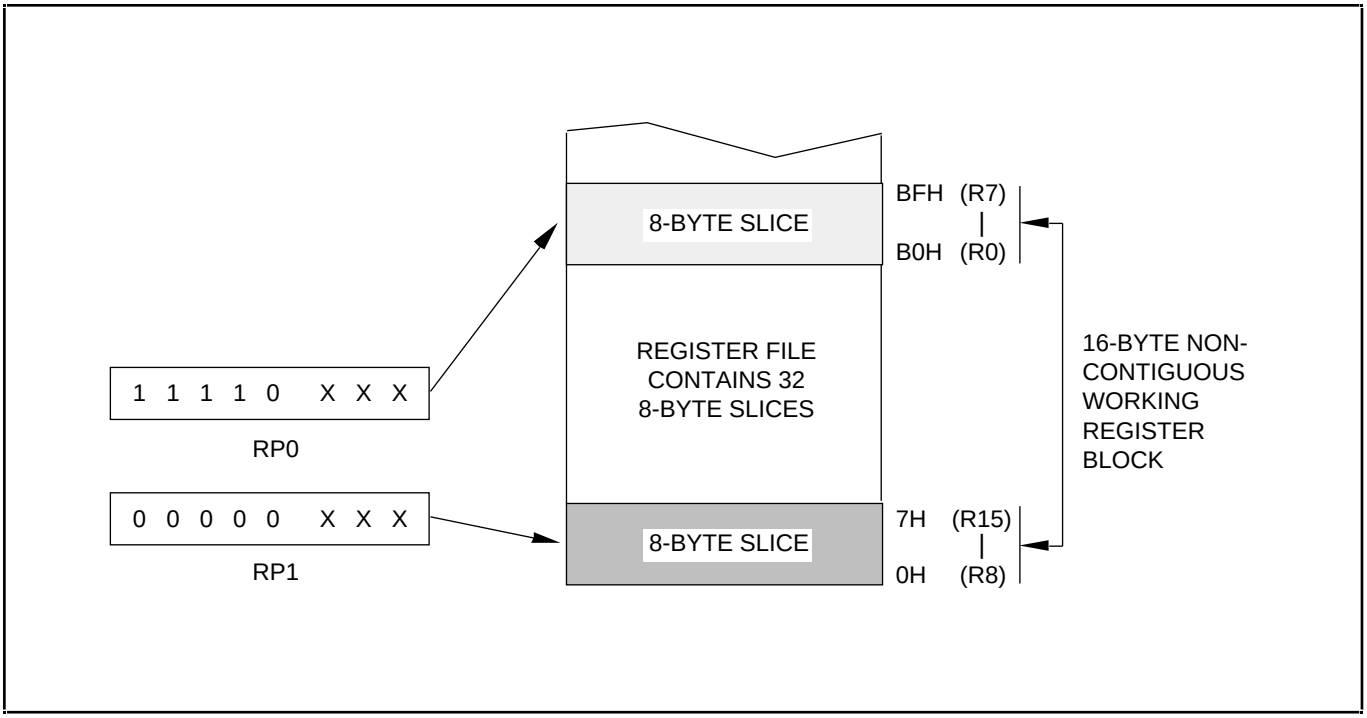


Figure 2–7. Non-Contiguous 16-Byte Working Register Block

☞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 80H through 85H contains the values 10H, 11H, 12H, 13H, 14H, and 15 H, 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 instead of 12 bytes, and its execution time is 50 cycles instead of 36 cycles.

REGISTER ADDRESSING

The SAM8 register architecture provides an efficient method of working register addressing that takes full advantage of shorter instruction formats to reduce execution time.

The Register (R) addressing mode, in which the operand value is the content of a specific register or register pair, can be used to access all locations in the register file except for set 2.

For working register addressing, the register pointers RP0 and RP1 are used to select a specific register within a selected 16-byte working register area. To increase the speed of context switches in an application program, you can use the register pointers to dynamically select different 8-byte "slices" of the register file as the program's active working register space.

Registers are addressed either as a single 8-bit register or as a paired 16-bit register. In 16-bit register pairs, the address of the first 8-bit register is always an even number and the address of the next register is an odd number.

The most significant byte of the 16-bit data is always stored in the even-numbered register; the least significant byte is always stored in the next (+ 1) odd-numbered register.

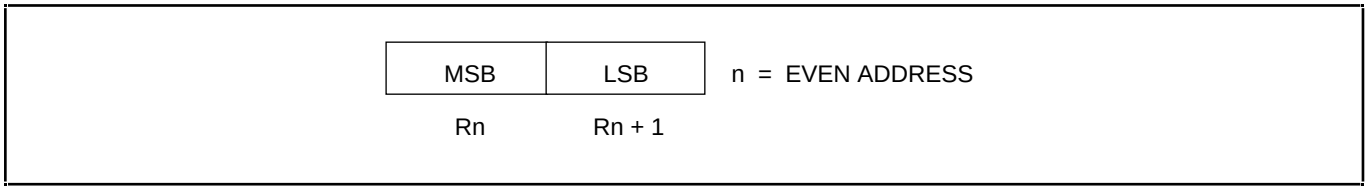


Figure 2–8. 16-Bit Register Pairs

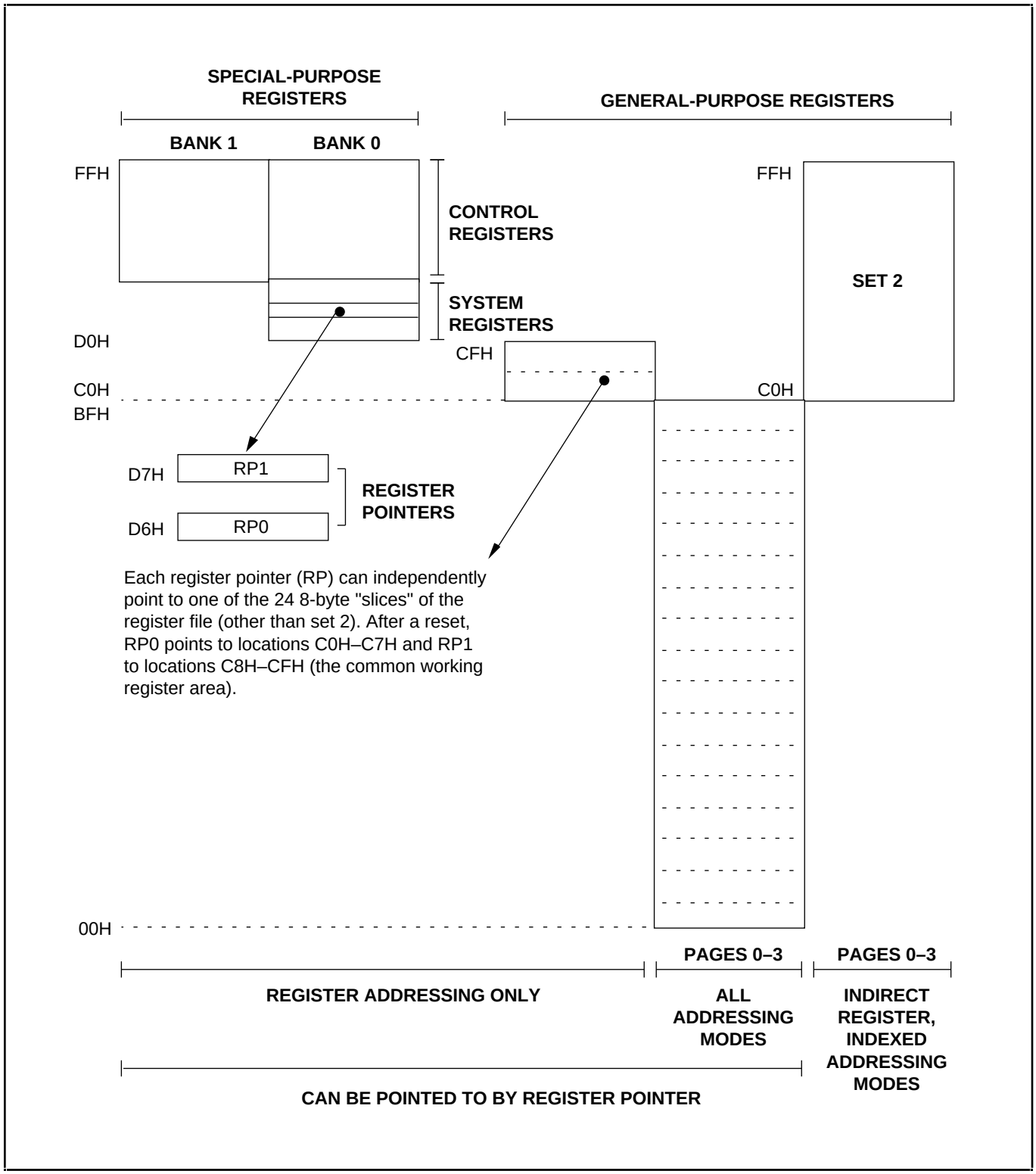


Figure 2–9. Register File Addressing

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 the *common area*. You can use common area registers as working registers for operations that address locations on different pages in the register file.

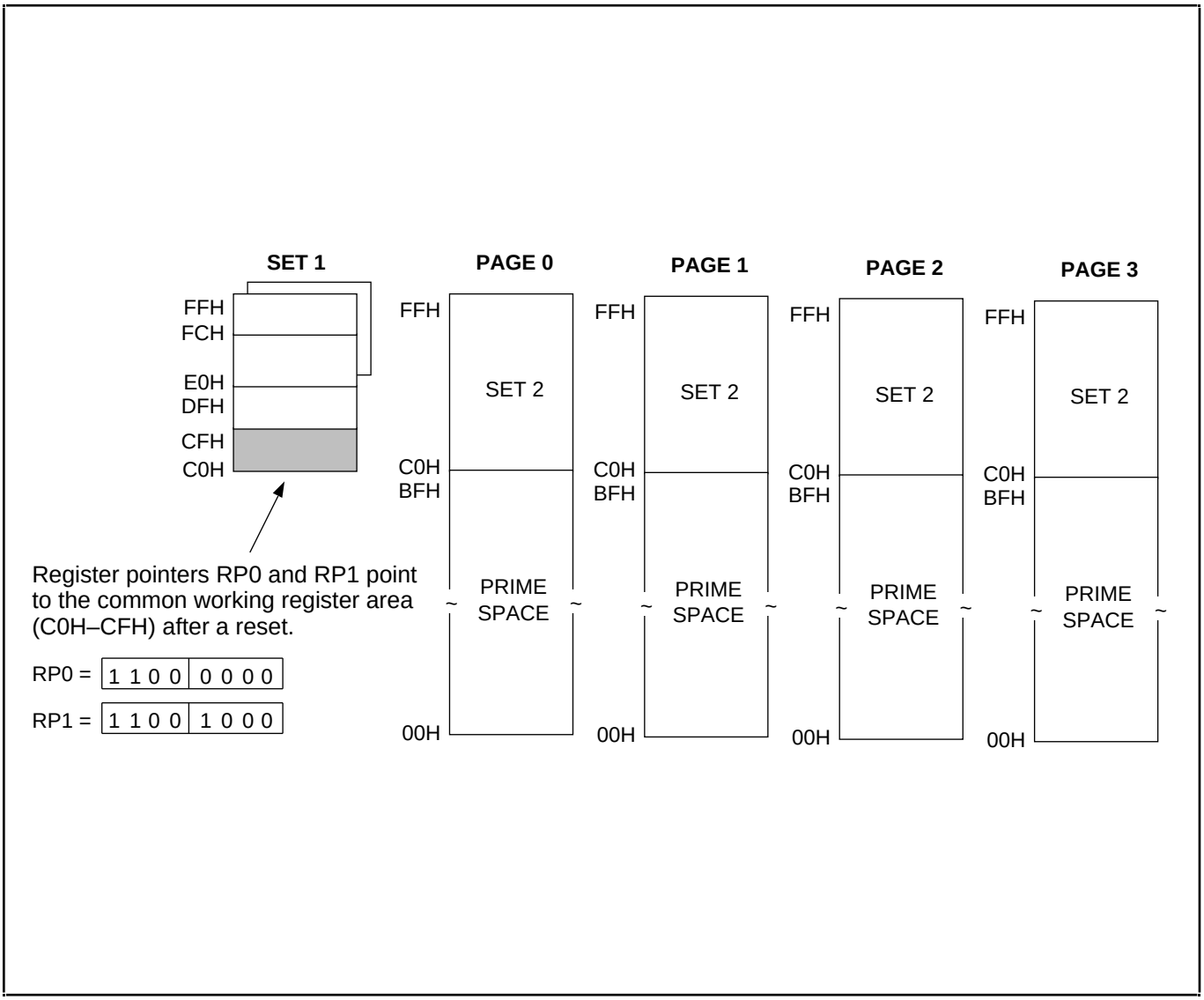


Figure 2–10. Common Working Register Area

☞ 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.

Example 1:

```
LD      0C2H,40H        ;  Invalid addressing mode!
```

Use working register addressing instead:

```
SRP     #0C0H
LD      R2,40H          ;  R2 (C2H) ← the value in location 40H
```

Example 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 enables 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 net effect 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).

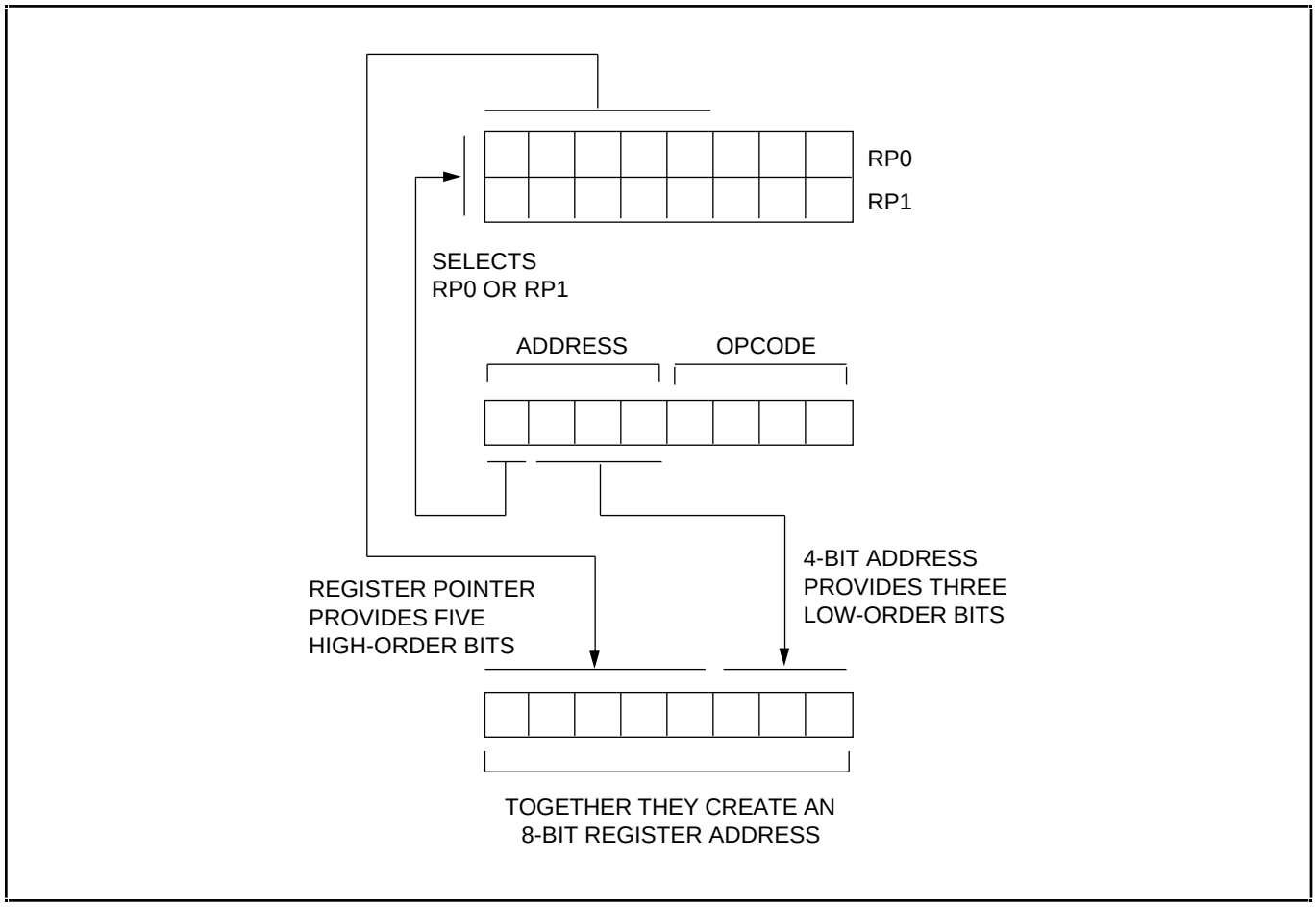


Figure 2–11. 4-Bit Working Register Addressing

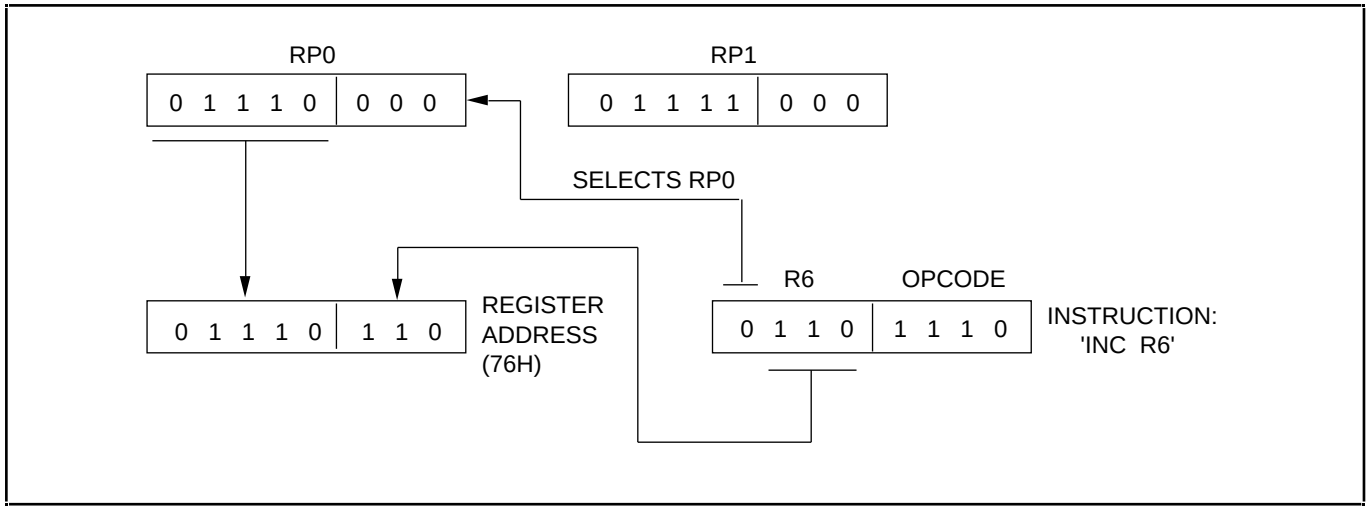


Figure 2–12. 4-Bit Working Register Addressing Example

8-Bit Working Register Addressing

You can also use 8-bit working register addressing to access registers in a selected working register area. In order 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, and 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. The fourth bit ("1") selects RP1 and the five high-order bits in RP1 (10100B) 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. Together, the five address bits from RP1 and the three address bits from the instruction comprise the complete register address, R163 (10100011B).

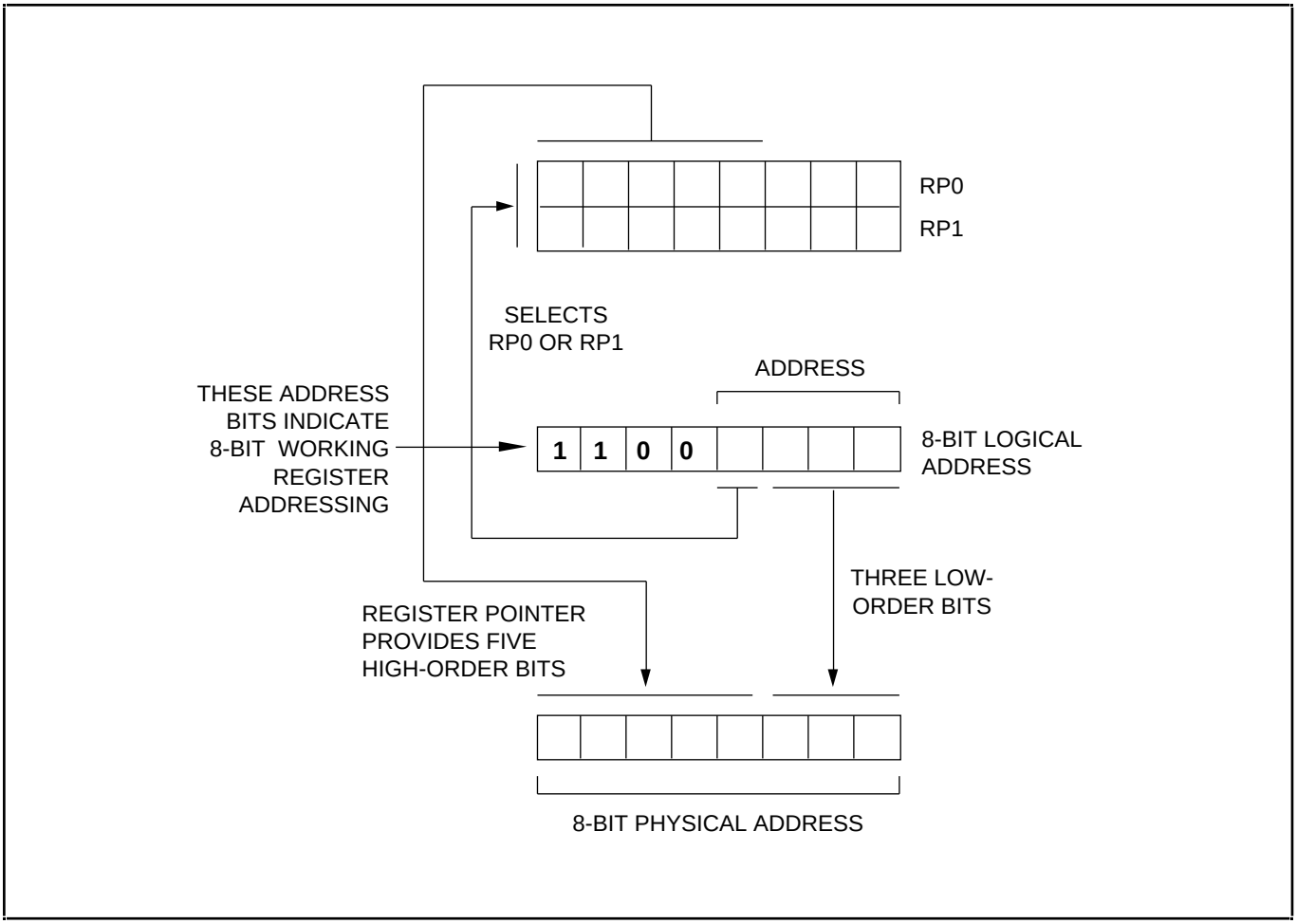


Figure 2–13. 8-Bit Working Register Addressing

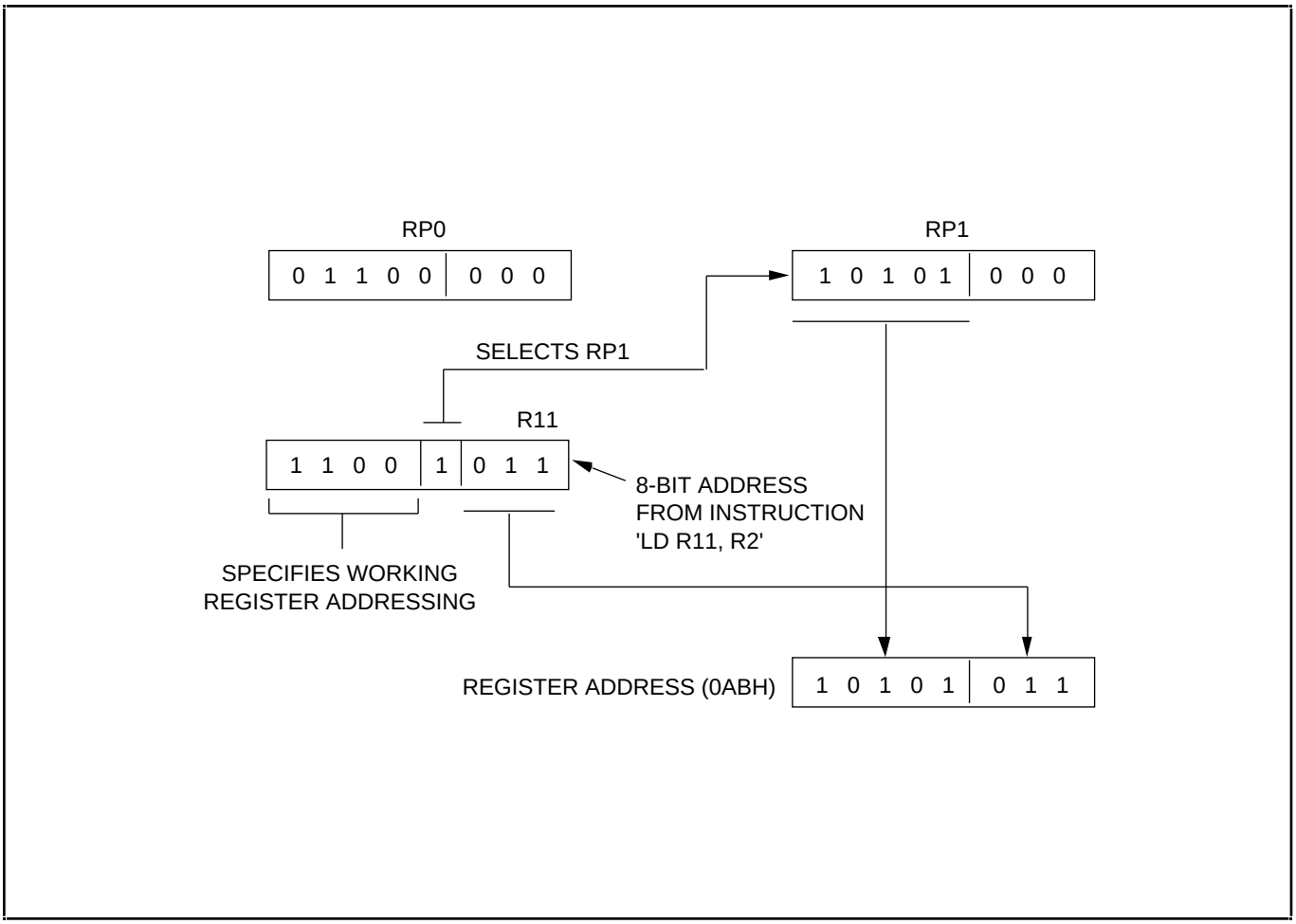


Figure 2–14. 8-Bit Working Register Addressing Example

System and User Stacks

KS88-series microcontrollers can be programmed to use system stack for subroutine calls, returns, interrupts, and to store data. The PUSH and POP instructions are used to control system stack operations.

The SAM8 architecture supports stack operations in the internal register file as well as in external data memory. To select an internal or external stack area, you manipulate bit 1 of the external memory timing register, EMT.1.

Stack Operations

Return addresses for procedure calls and 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 is always decremented *before* a push operation and incremented *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.

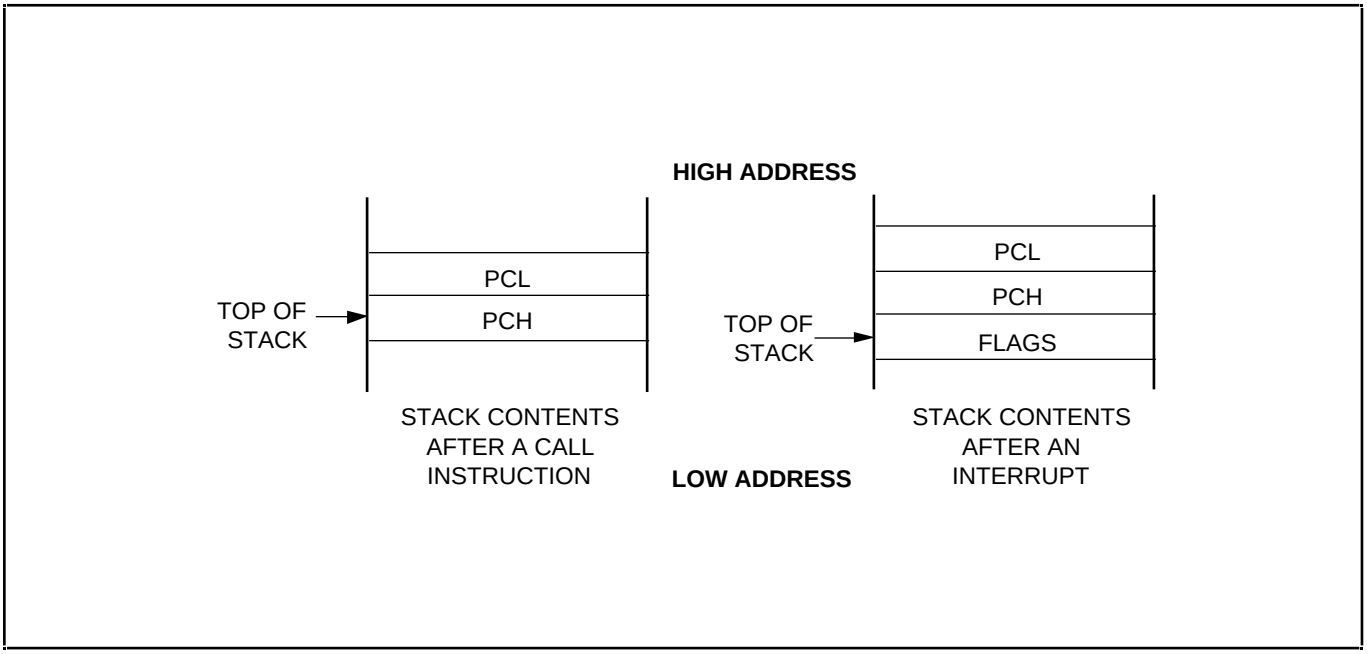


Figure 2–15. Stack Operations

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.

These instructions cannot address external memory locations. Only PUSH and POP instructions can be used for an externally defined stack.

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); the least significant byte, SP7–SP0, is stored in the SPL register (D9H). After a reset, the SP value is undetermined.

If only internal memory space is implemented, 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 needed). If external memory is implemented, both SPL and SPH must be initialized with a full 16-bit address.

When the SPL register contains the only stack pointer value (that is, when it points to a system stack in the register file), the SPH register can be used as a general-purpose data register.

However, if an overflow or underflow condition occurs as the result of incrementing or decrementing the stack address 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.

Stack operation page is in only *page 0*, regardless the processing page.

☞ 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 SAM8 instructions may be condition codes, immediate data, or a location in the register file, program memory, or data memory.

The SAM8 instruction set supports seven explicit addressing modes. Not all of these addressing modes are available for each instruction. The addressing modes and their symbols are as follows:

— 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, the operand is the content of a specified register or register pair (see Figure 3–1). Working register addressing differs from Register addressing because 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).

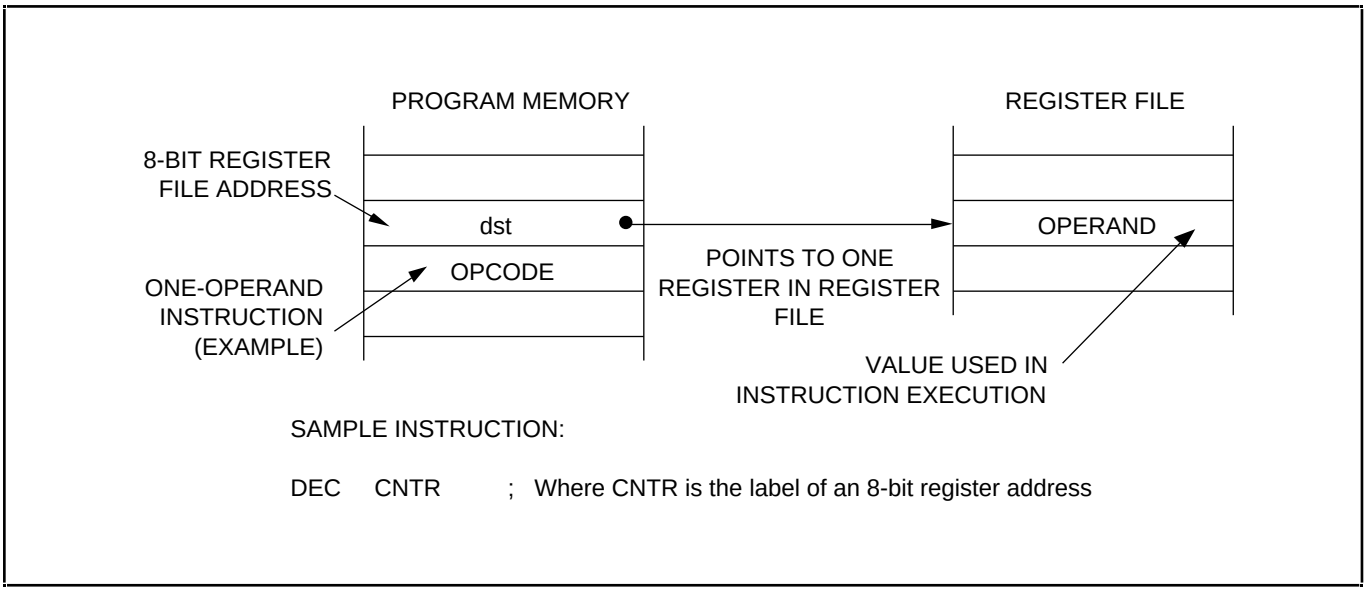


Figure 3–1. Register Addressing

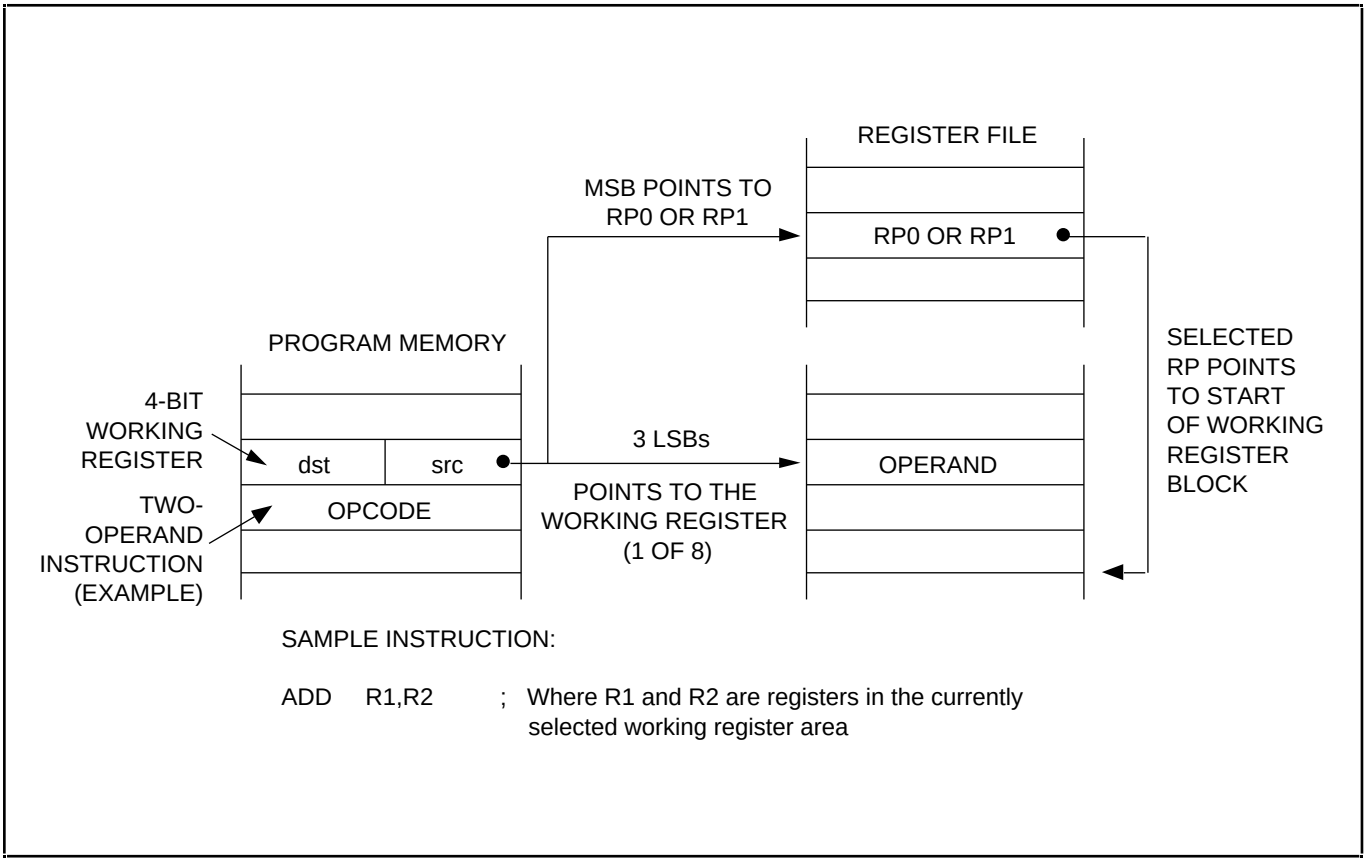


Figure 3–2. Working Register Addressing

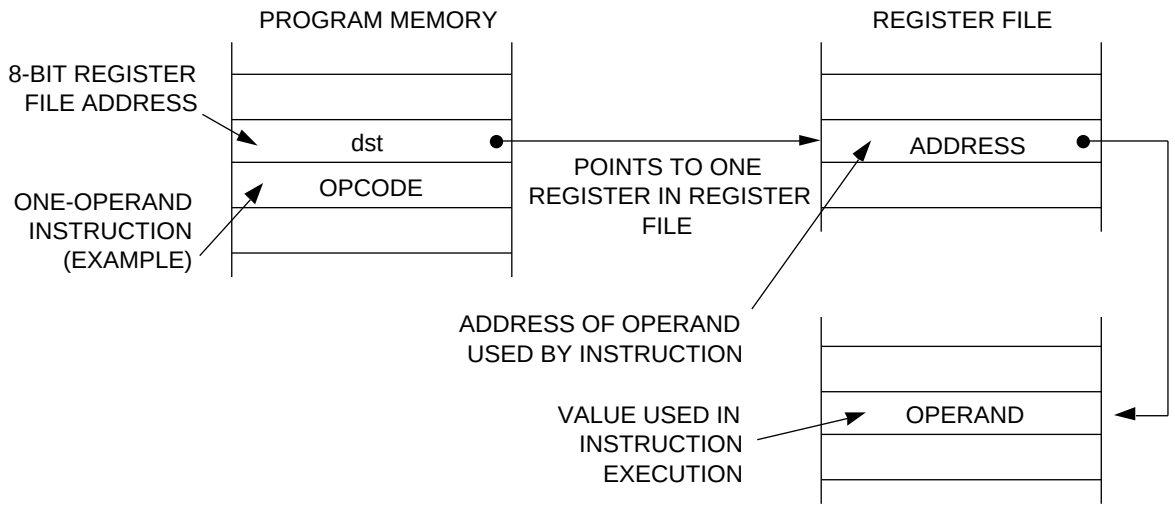
Indirect Register Addressing Mode (IR)

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.

You cannot, however, access locations C0H–FFH in set 1 using Indirect Register addressing mode.



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)

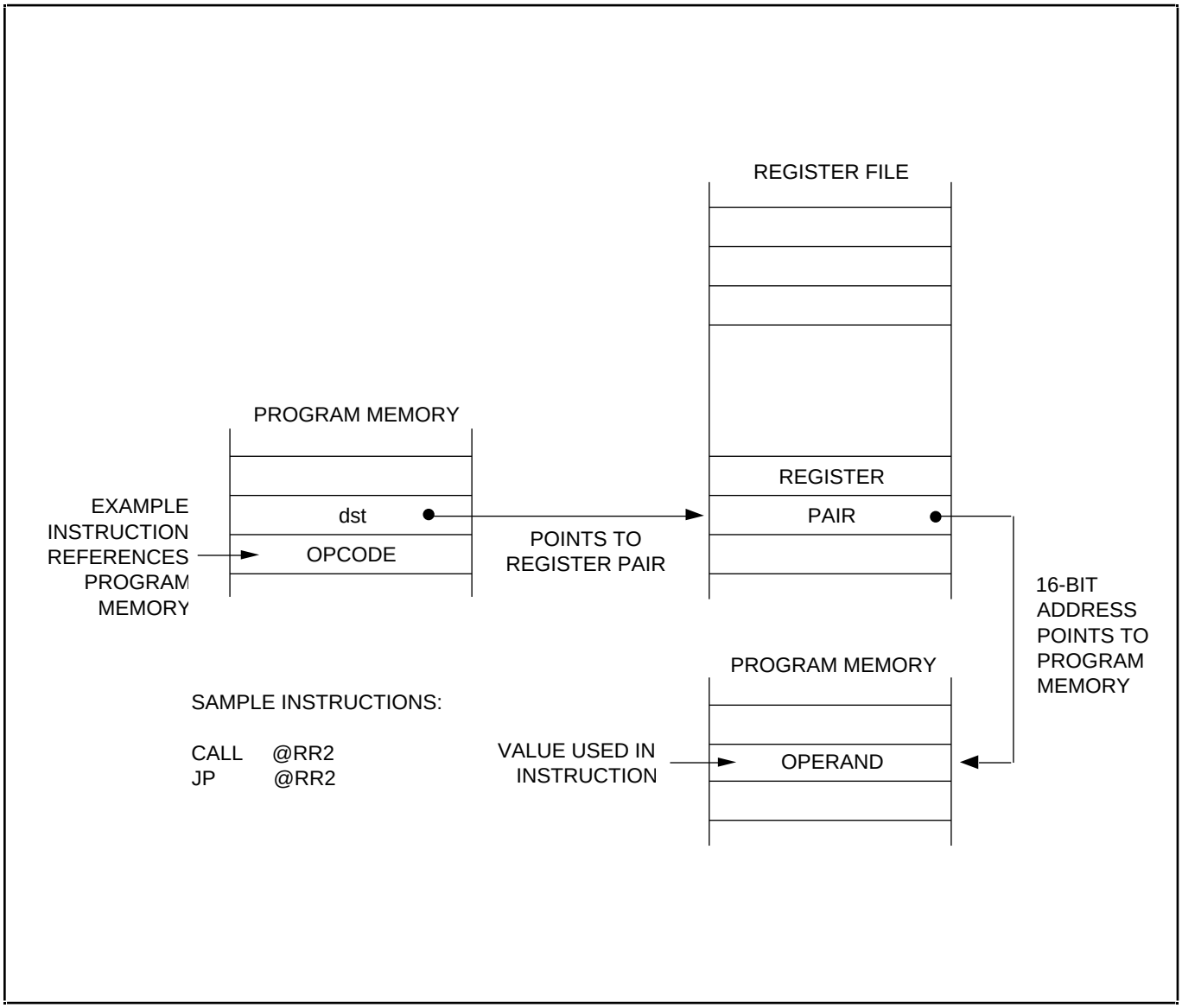


Figure 3–4. Indirect Register Addressing to Program Memory

Indirect Register Addressing Mode (Continued)

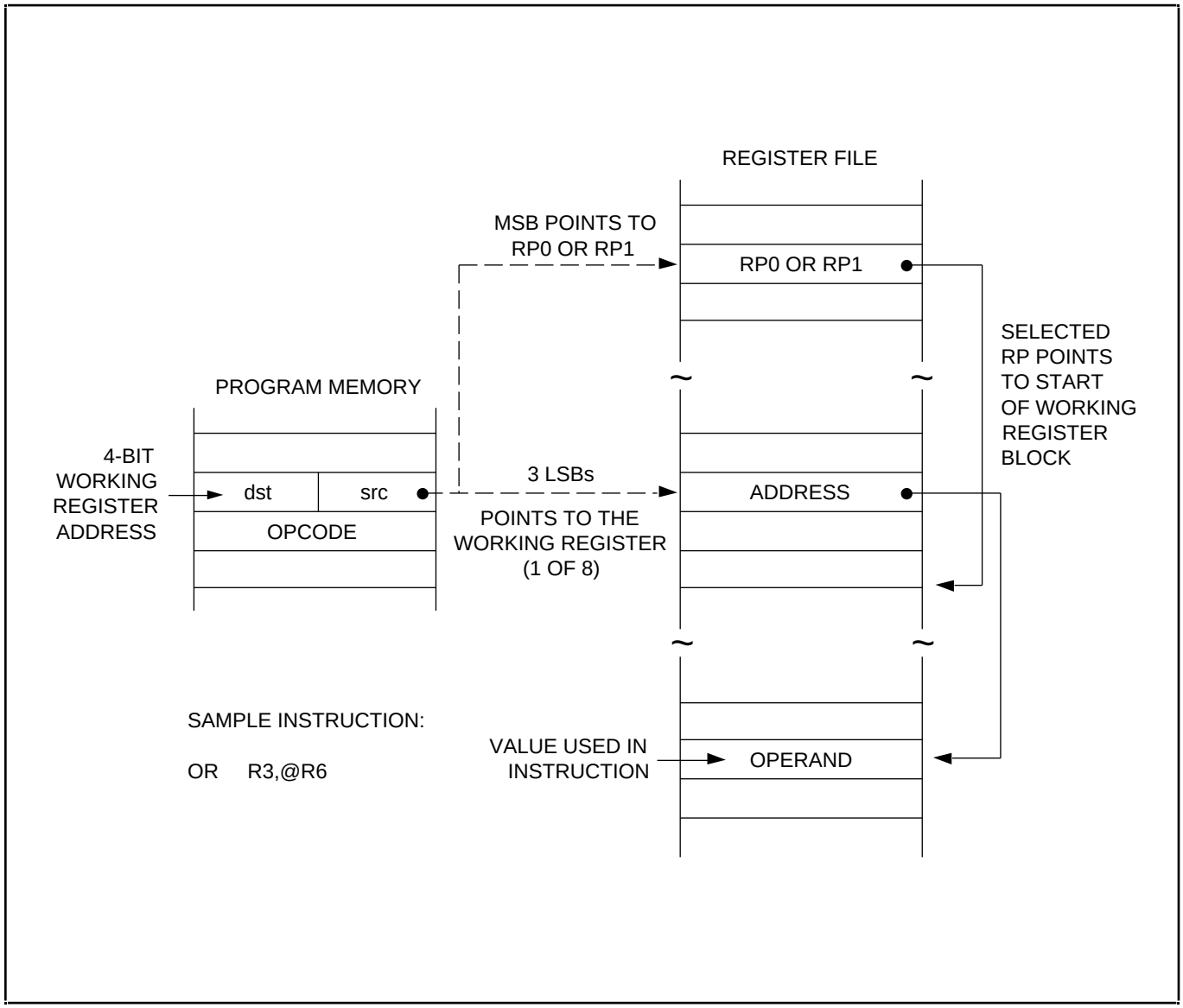
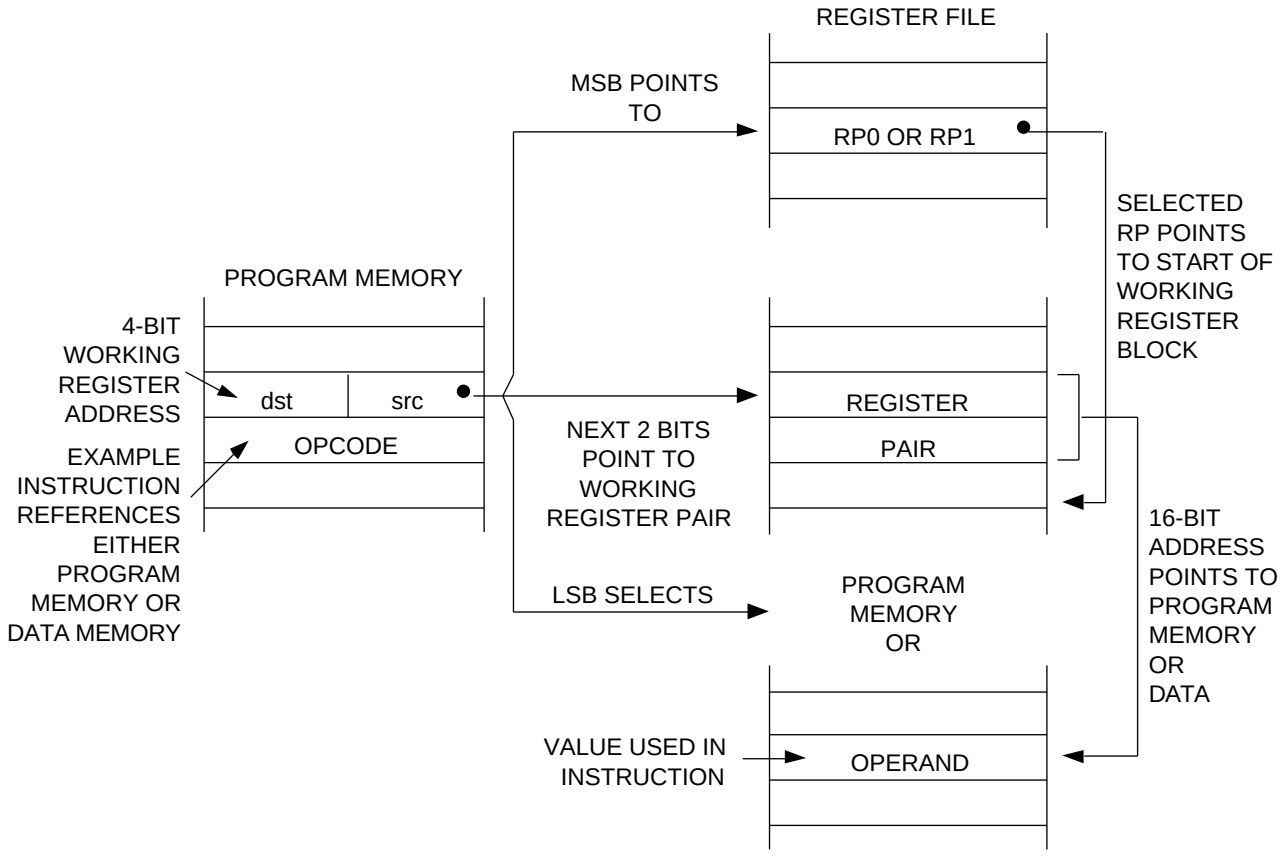


Figure 3–5. Indirect Working Register Addressing to Register File

Indirect Register Addressing Mode (Concluded)



SAMPLE INSTRUCTIONS:

```
LDC    R5,@RR6      ;  Program memory access
LDE    R3,@RR0      ;  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. You cannot, however, 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 the 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.

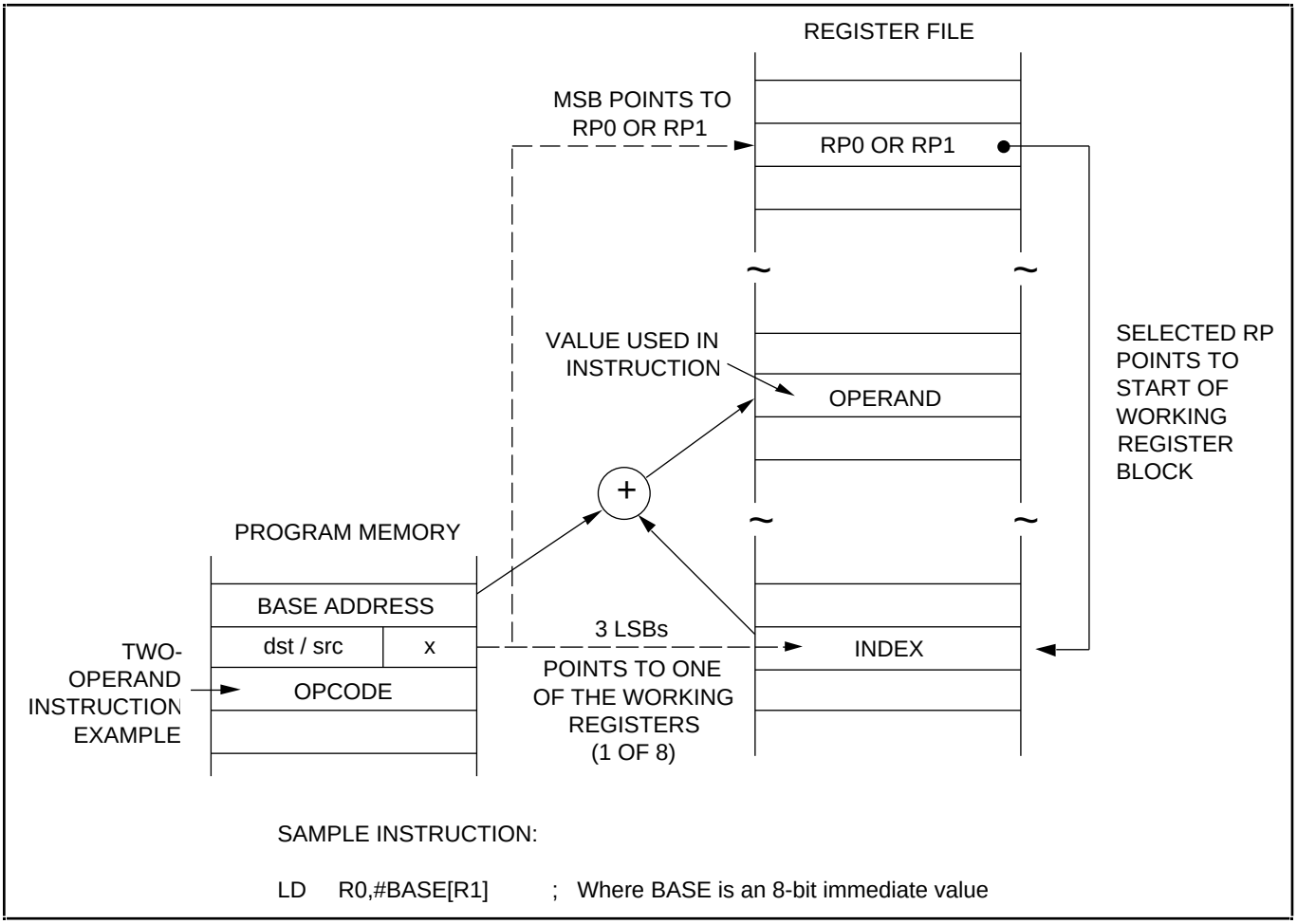
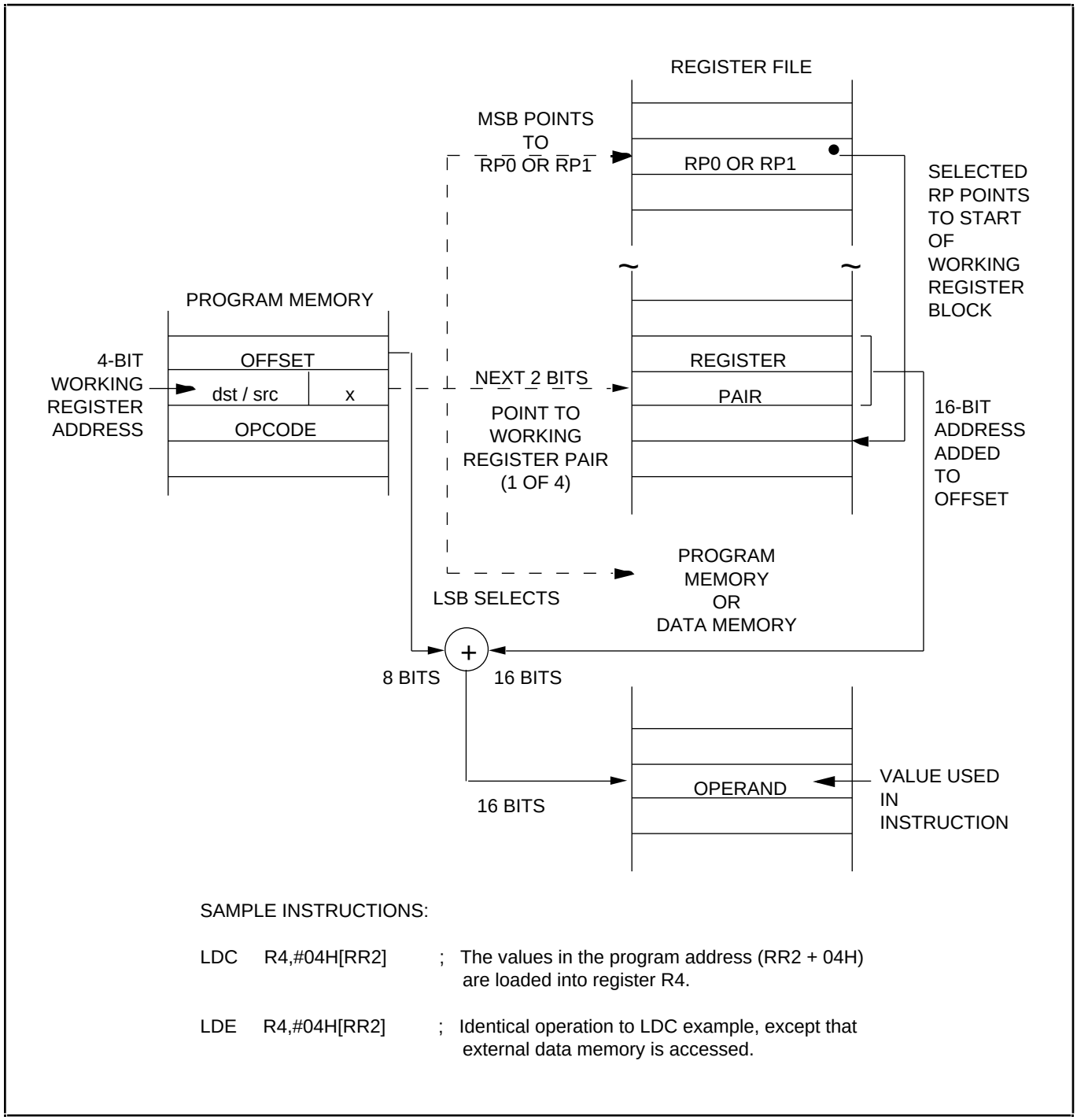


Figure 3–7. Indexed Addressing to Register File

Indexed Addressing Mode (Continued)

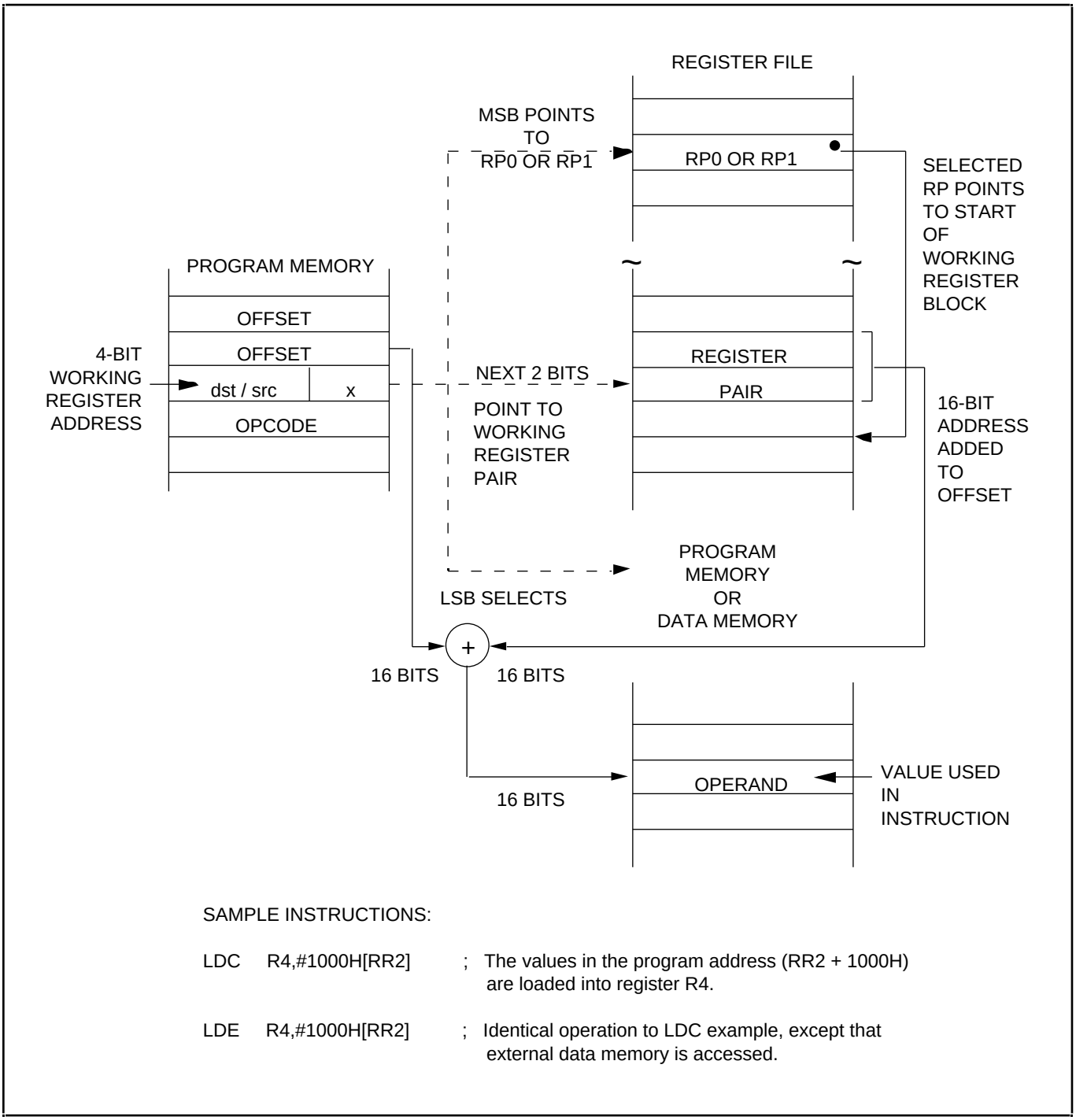


SAMPLE INSTRUCTIONS:

```
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 data memory is accessed.
```

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

Indexed Addressing Mode (Concluded)



SAMPLE INSTRUCTIONS:

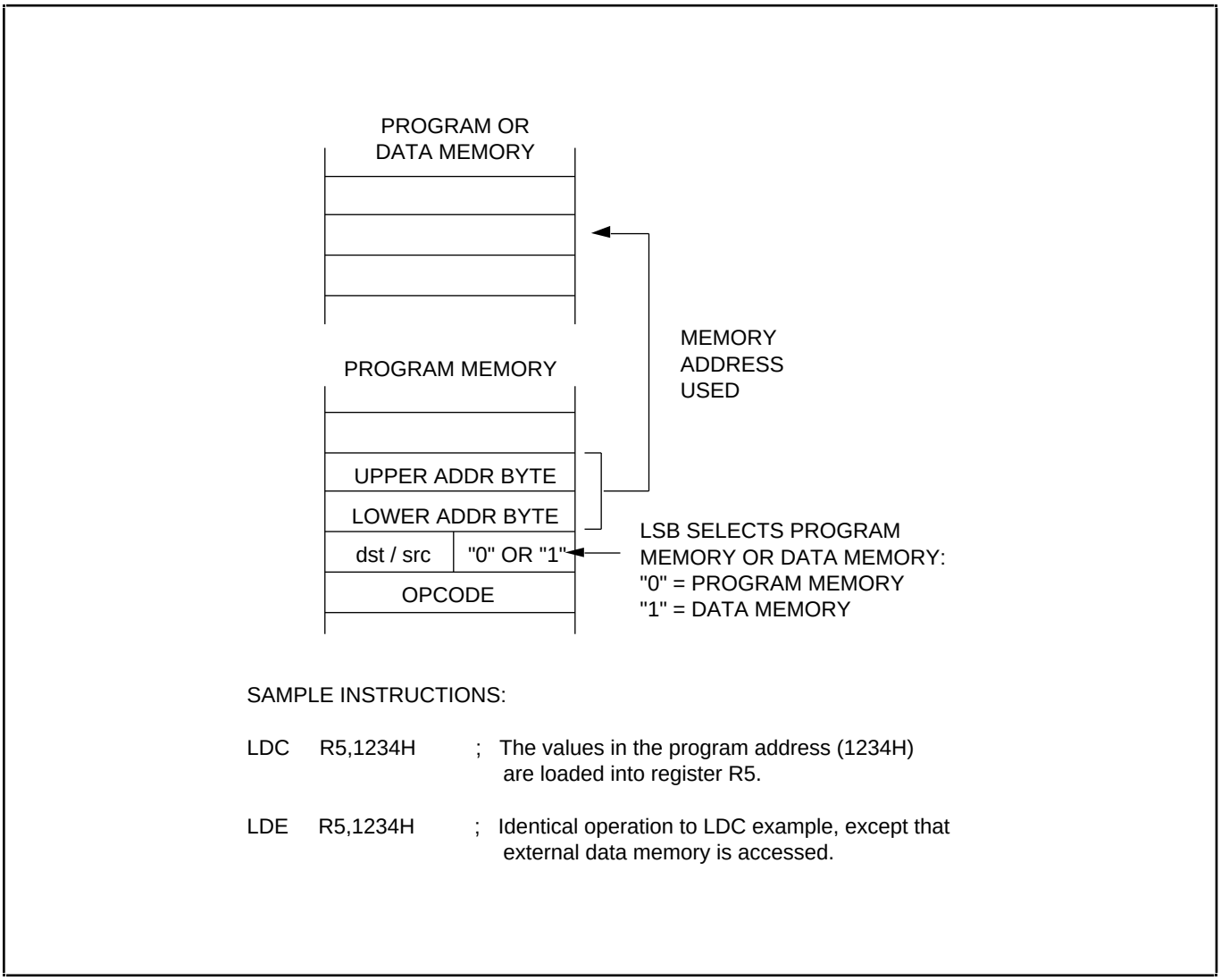
```
LDC     R4,#1000H[RR2]      ;  The values in the program address (RR2 + 1000H)
                               are loaded into register R4.

LDE     R4,#1000H[RR2]      ;  Identical operation to LDC example, except that
                               external data 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.

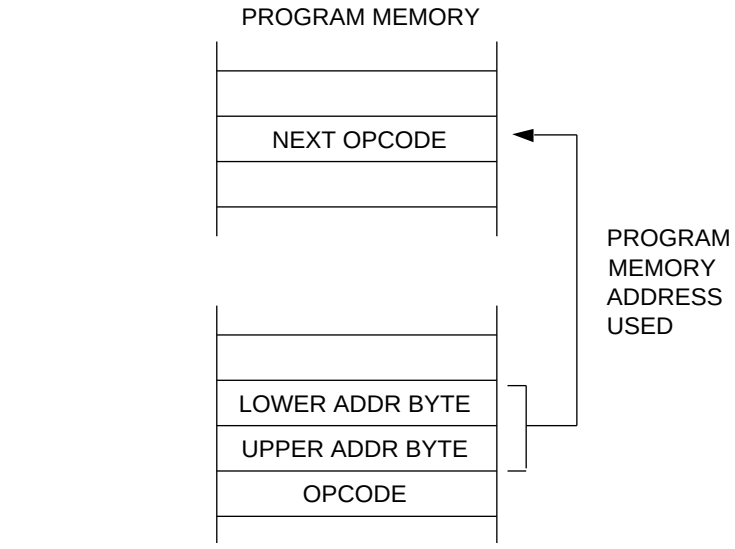


SAMPLE INSTRUCTIONS:

```
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 data memory is accessed.
```

Figure 3–10. Direct Addressing for Load Instructions

Direct Address Mode (Continued)



SAMPLE INSTRUCTIONS:

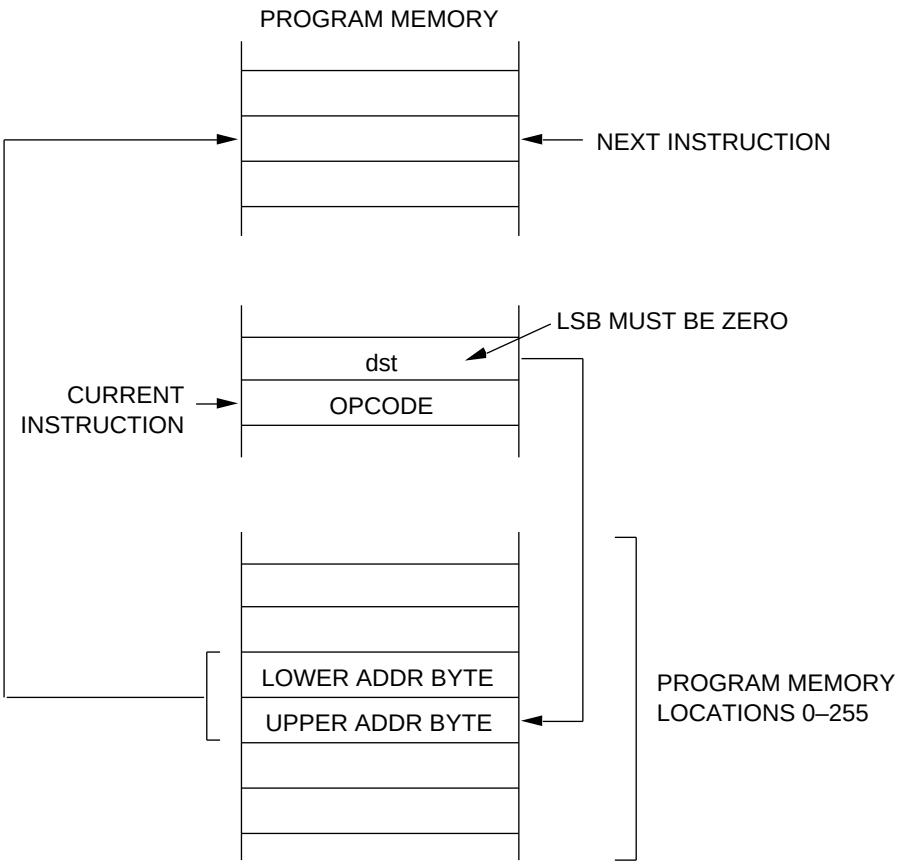
```
JP      C,JOB1      ;  Where JOB1 is a 16-bit immediate address
CALL    DISPLAY     ;  Where DISPLAY is a 16-bit immediate address
```

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 two's-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.

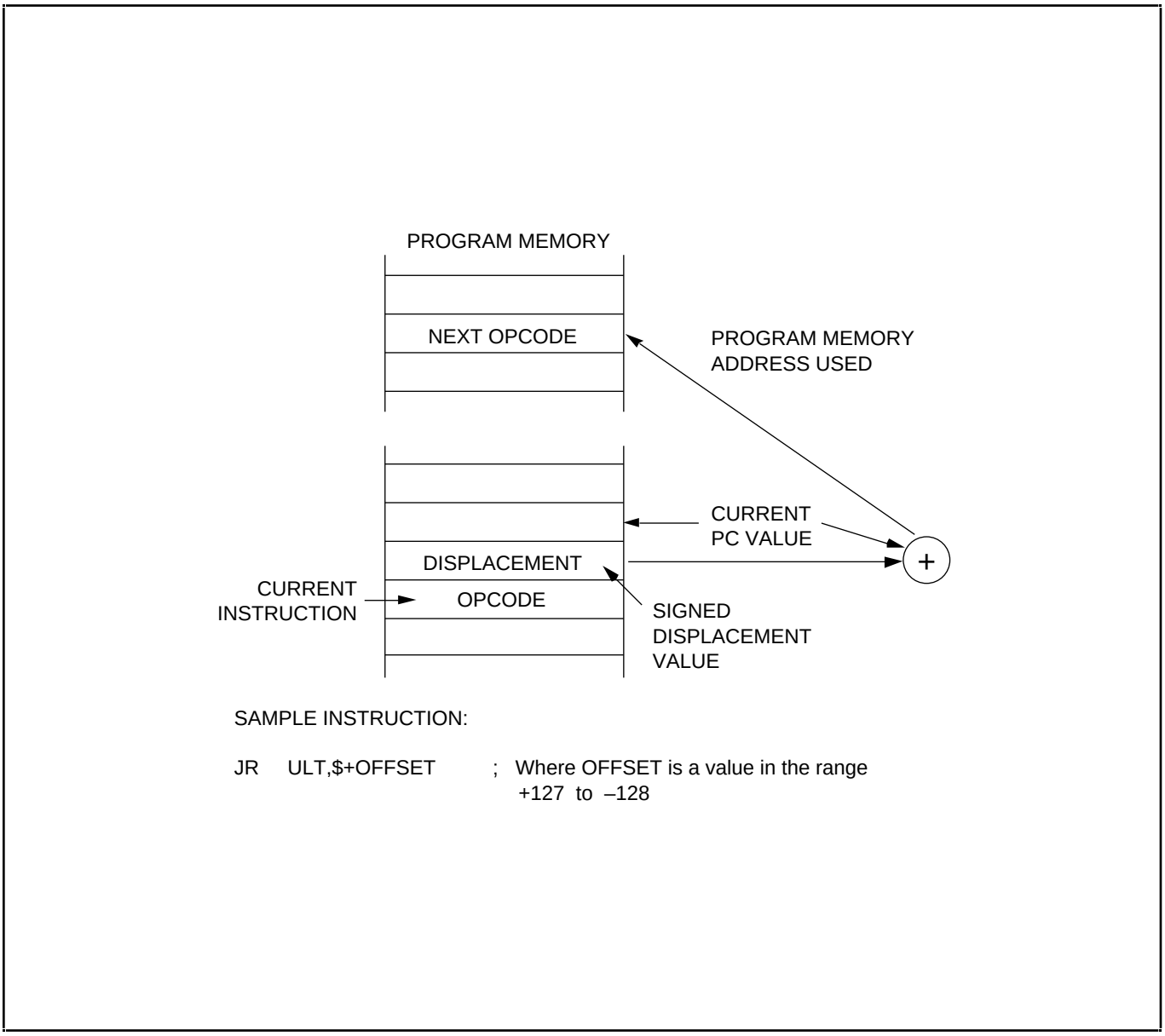


Figure 3–13. Relative Addressing

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.

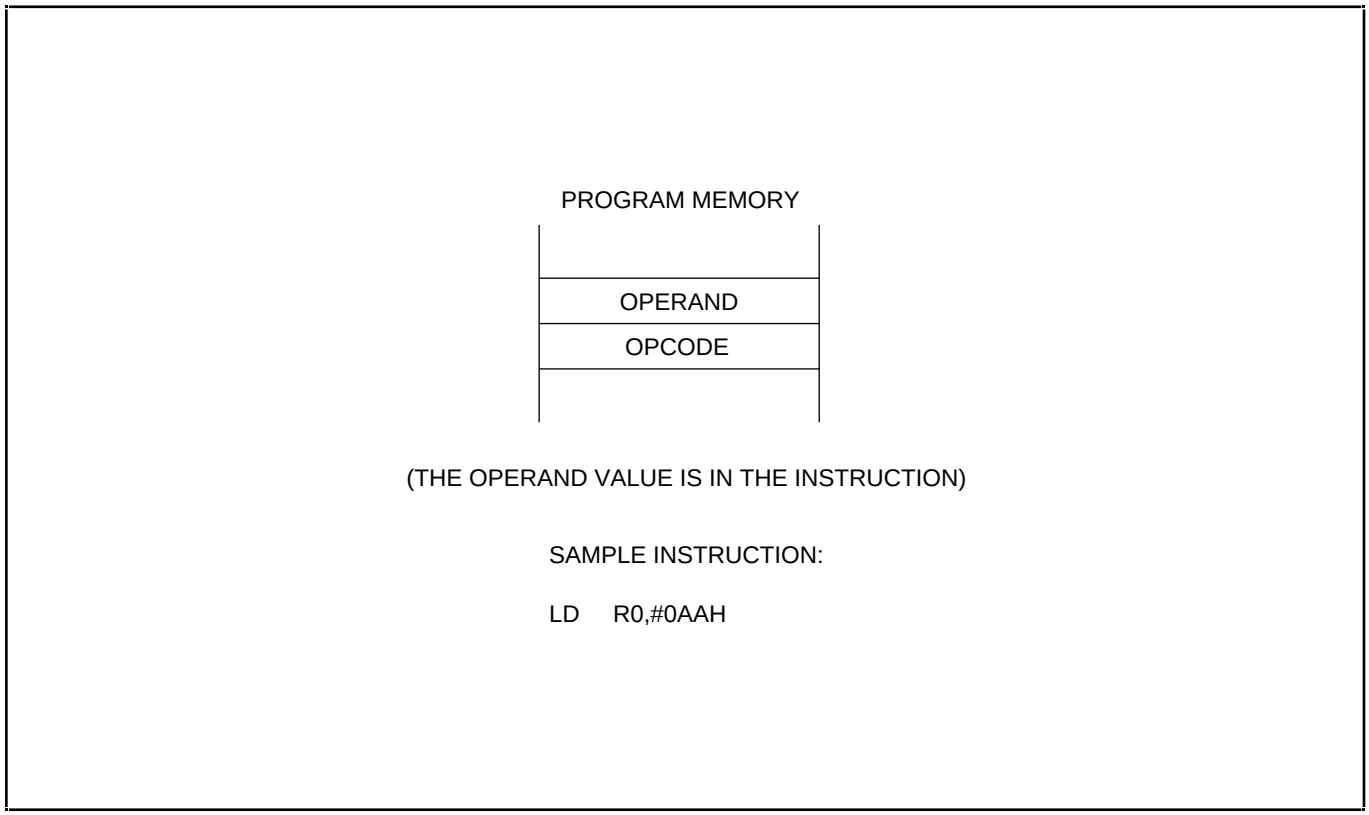


Figure 3–14. Immediate Addressing

4 Control Registers

OVERVIEW

In this section, detailed descriptions of the KS88C4400 control registers are presented in an easy-to-read format.

These descriptions will help familiarize you with the mapped locations in the register file. You can also use them as a quick-reference source when writing application programs.

System and peripheral registers are summarized in Tables 4–1, 4–2, and 4–3. 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 information about control registers is presented in the context of the various peripheral hardware descriptions in Part II of this manual.

Table 4–1. Set 1 Registers

Register Name	Mnemonic	Decimal	Hex	R/W
Timer C counter (high byte)	TCH	208	D0H	R/W
Timer C counter (low byte)	TCL	209	D1H	R/W
Timer D counter (high byte)	TDH	210	D2H	R/W
Timer D counter (low byte)	TDL	211	D3H	R/W
Port 4 interrupt pending pegister	P4PND	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
Locations E0H and E1H are not mapped.				
Port 2 data register	P2	226	E2H	R/W
Port 3 data register	P3	227	E3H	R/W
Port 4 data register	P4	228	E4H	R/W
Port 5 data register	P5	229	E5H	R/W
Port 6 data register	P6	230	E6H	R/W
Locations E7H and E8H are not mapped.				
UART shift register	SIO	233	E9H	R/W
UART control register	SIOCON	234	EAH	R/W
UART interrupt pending register	SIOPND	235	EBH	R/W
Timer A data register	TADATA	236	ECH	W
Timer B data register	TBDATA	237	EDH	W
Timer module 0 control register	T0CON	238	EEH	W (1)
Timer B control register	TBCON	239	EFH	R/W (2)

Table 4–2. Set 1, Bank 0 Registers (Continued)

Register Name	Mnemonic	Decimal	Hex	R/W
Locations F0H and F1H are not mapped.				
Port 2 control register	P2CON	242	F2H	R/W
Location F3H is not mapped.				
Port 3 control register (high byte)	P3CONH	244	F4H	R/W
Port 3 control register (low byte)	P3CONL	245	F5H	R/W
Port 4 control register (high byte)	P4CONH	246	F6H	R/W
Port 4 control register (low byte)	P4CONL	247	F7H	R/W
Port 5 control register	P5CON	248	F8H	R/W
Port 4 interrupt enable register	P4INT	249	F9H	R/W
Timer module 1 control register	T1CON	250	FAH	R/W
Timer module 1 mode register	T1MOD	251	FBH	R/W
Port 3 interrupt enable register	P3INT	252	FCH	R/W
Port 3 interrupt pending register	P3PND	253	FDH	R/W
External memory timing register	EMT	254	FEH	R/W
Interrupt priority register	IPR	255	FFH	R/W

NOTES:
1. T0CON.1 is read/write addressable; the other seven bits in this register are write-only.
2. The timer B operating mode selection bit, TBCON.0, is write-only.

Table 4–3. Set 1, Bank 1 Registers

Register Name	Mnemonic	Decimal	Hex	R/W
Locations E0H–F8H in set 1, bank 1, are not mapped.				
A/D converter input register	ADIN	249	F9H	R
A/D converter output register	ADOUT	250	FAH	R
A/D converter control register	ADCON	251	FBH	R/W (Note)
PWM module control register	PWMCON	252	FCH	R/W
PWM1 data register	PWM1	253	FDH	R/W
PWM0 data register	PWM0	254	FEH	R/W
PWM capture register	PWMCAP	255	FFH	R/W

NOTE: The A/D converter end-of-conversion bit, ADCON.3, is read-only.

☞ Programming Tip — Using Load Instructions for Read-Only and Write-Only Registers

To avoid programming errors, we recommend that you not use the instructions OR (Logical OR), AND (Logical AND), CP (Compare), and LDB (Load Bit) to access write-only registers. Use Load instructions instead (except for LDB). Here are some examples:

Example 1:

```
        OR      T0CON,#04H          ;  Invalid use of logical-OR instruction!
```

Use the LD instruction instead to manipulate the T0CON register:

```
        OR      ST0CON, #00000100B  ;  ST0CON is a shadow register for T0CON
        LD      T0CON,ST0CON        ;  Set bit 2 in the T0CON register
```

Example 2:

```
        CP      T0CON,#3CH          ;  Invalid use of the CP instruction!
        JP      EQ,AAA
        •
        •
        •
AAA     NOP
```

Use a shadow register instead to manipulate the T0CON register:

```
        CP      ST0CON,#3CH         ;  ST0CON is a shadow register for T0CON
        JP      EQ,AAA
        •
        •
        •
AAA     NOP
```

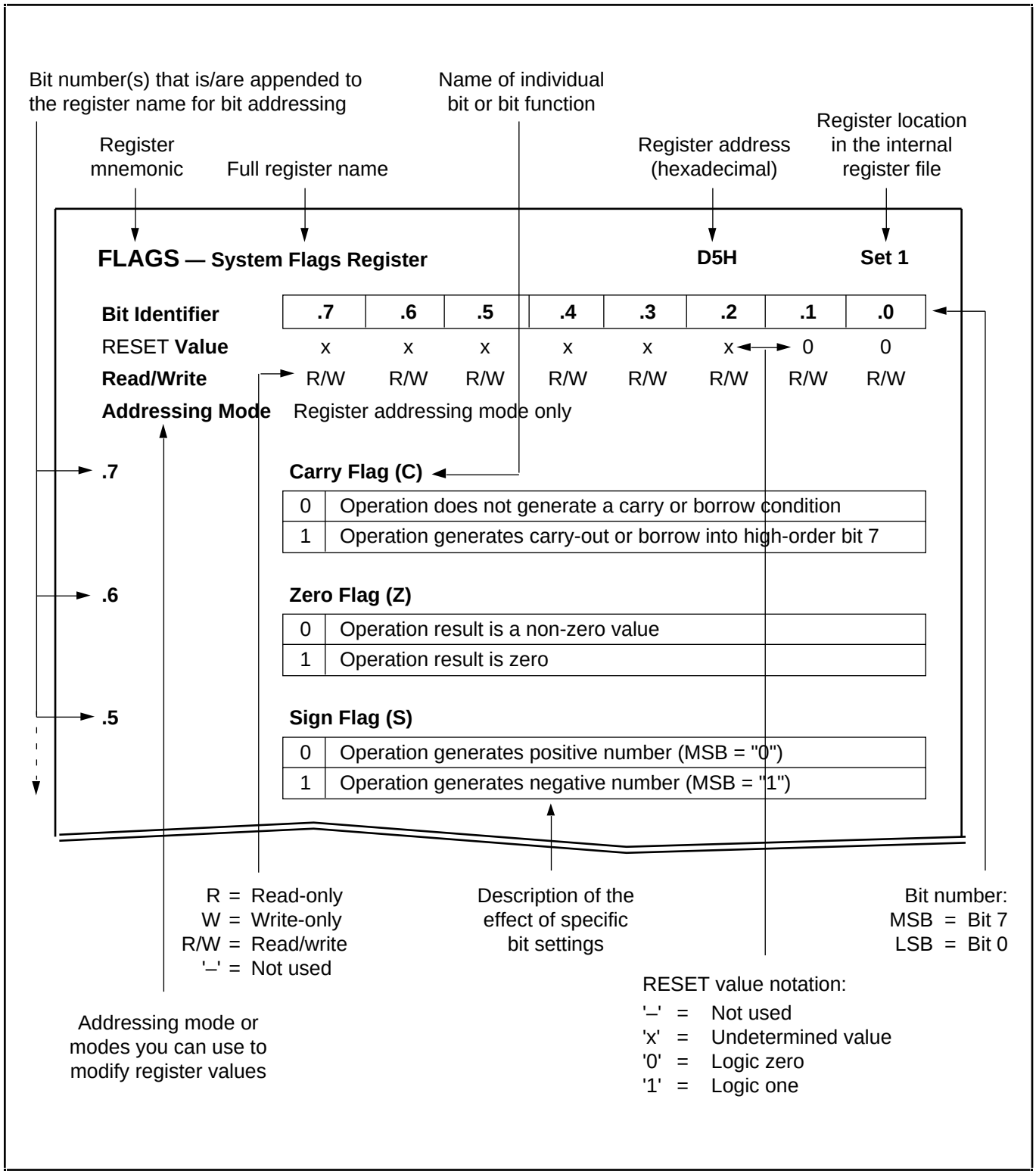


Figure 4–1. Register Description Format

ADCON — A/D Converter Control Register FBH Set 1, Bank 1

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

.7 **A/D Converter Test Mode Control Bit**

This bit is used for factory testing only. During normal operation, ADCON.7 should always remain cleared to "0".

.6 – .4 **A/D Converter Analog Input Pin Selection Bits**

0	0	0	ADC0 (P7.0)
0	0	1	ADC1 (P7.1)
0	1	0	ADC2 (P7.2)
0	1	1	ADC3 (P7.3)
1	0	0	ADC4 (P7.4)
1	0	1	ADC5 (P7.5)
1	1	0	ADC6 (P7.6)
1	1	1	ADC7 (P7.7)

.3 **End-of-Conversion Bit (Read-only)** (1, 2)

0	A/D conversion operation is in progress
1	A/D conversion operation is complete

.2 – .0 Not used for KS88C4400

NOTE: This bit is read-only. You can poll ADCON.3 to determine internally when an A/D conversion operation has been completed. A reset operation sets ADCON.3 to "1".

EMT — External Memory Timing Register FEH Set 1, Bank 0

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

.7 **External** WAIT **Input Function Enable Bit** (Note)

0	Disable WAIT input function for external device (normal operating mode)
1	Enable WAIT input function for external device

.6 **Slow Memory Timing Enable Bit**

0	Disable slow memory timing
1	Enable slow memory timing

.5 and .4 **Program Memory Automatic Wait Control Bits**

0	0	No wait (normal operation)
0	1	Wait one cycle
1	0	Wait two cycles
1	1	Wait three cycles

.3 and .2 **Data Memory Automatic Wait Control Bits**

0	0	No wait (normal operation)
0	1	Wait one cycle
1	0	Wait two cycles
1	1	Wait three cycles

.1 **Stack Area Selection Bit**

0	Select internal register file area
1	Select external data memory area

.0 Not used for KS88C4400

NOTE: Before you enable the external interface WAIT input function, you must first configure P3.7 as the WAIT signal input pin. To do this, bits 6 and 7 in the P3CONH register must be set to one of the input mode settings.

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/W	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	Cleared automatically during an interrupt return (IRET)
1	Automatically set to "1" during a fast interrupt service routine

.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; INT8–INT11

0	Disable IRQ7 interrupts
1	Enable IRQ7 interrupts

.6 Interrupt Level 6 (IRQ6) Enable Bit; INT5–INT7

0	Disableπ IRQ6 interrupts
1	Enable IRQ6 interrupts

.5 Interrupt Level 5 (IRQ5) Enable Bit; INT4

0	Disable IRQ5 interrupts
1	Enable IRQ5 interrupts

.4 Interrupt Level 4 (IRQ4) Enable Bit; INT0–INT3

0	Disable IRQ4 interrupts
1	Enable IRQ4 interrupts

.3 Interrupt Level 3 (IRQ3) Enable Bit; Serial Rx/Tx, Timers C and D

0	Disable IRQ3 interrupts
1	Enable IRQ3 interrupts

.2 Interrupt Level 2 (IRQ2) Enable Bit

Not used for KS88C4400

.1 Interrupt Level 1 (IRQ1) Enable Bit; PWM, Capture, and Timer A

0	Disable IRQ1 interrupts
1	Enable IRQ1 interrupts

.0 Interrupt Level 0 (IRQ0) Enable Bit; Timer B

0	Disable IRQ0 interrupts
1	Enable IRQ0 interrupts

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

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**

Not used for KS88C4400

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

0	IRQ0 > IRQ1
1	IRQ1 > IRQ0

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 Interrupt Level 7 (IRQ7) Request Pending Bit; INT8–INT11

0	No IRQ7 interrupt pending
1	IRQ7 interrupt is pending

.6 Interrupt Level 6 (IRQ6) Request Pending Bit; INT5– INT7

0	No IRQ6 interrupt pending
1	IRQ6 interrupt is pending

.5 Interrupt Level 5 (IRQ5) Request Pending Bit; INT4

0	No IRQ5 interrupt pending
1	IRQ5 interrupt is pending

.4 Interrupt Level 4 (IRQ4) Request Pending Bit; INT0–INT3

0	No IRQ4 interrupt pending
1	IRQ4 interrupt is pending

.3 Interrupt Level 3 (IRQ3) Request Pending Bit; Serial Rx/Tx, Timers C and D

0	No IRQ3 interrupt pending
1	IRQ3 interrupt is pending

.2 Interrupt Level 2 (IRQ2) Request Pending Bit

Not used for KS88C4400

.1 Interrupt Level 1 (IRQ1) Request Pending Bit; PWM, Capture, and Timer A

0	No IRQ1 interrupt pending
1	IRQ1 interrupt is pending

.0 Interrupt Level 0 (IRQ0) Request Pending Bit; Timer B

0	No IRQ0 interrupt pending
1	IRQ0 interrupt is pending

P2CON — Port 2 Control Register F2H Set 1, Bank 0

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

.7 and .6 **P2.7/ TB Configuration Control Bits**

0	x	Input mode
1	0	Push-pull output mode
1	1	Timer B output enabled

.5 and .4 **P2.6/ TA Configuration Control Bits**

0	x	Input mode
1	0	Push-pull output mode
1	1	Timer A output enabled

.3 – .0 Not used for KS88C4400

NOTE: 'x' means don't care.

P3CONH — Port 3 Control Register (High Byte) 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 and .6 **P3.7 /** WAIT **Configuration Bits**

0	x	Input mode, WAIT signal input enabled
1	0	Input mode, pull-up resistor active, WAIT signal input enabled
1	1	Push-pull output mode

.5 and .4 **P3.6 / CAP Configuration Bits**

0	x	Input mode, capture input for PWM (CAP pin) enabled
1	0	Input mode, pull-up active, capture input for PWM (CAP pin) enabled
1	1	Push-pull output mode

.3 and .2 **P3.5 Configuration Bits**

0	x	Input mode
1	0	Input mode, pull-up resistor active
1	1	Push-pull output mode

.1 and .0 **P3.4 Configuration Bits**

0	x	Input mode
1	0	Input mode, pull-up resistor active
1	1	Push-pull output mode

NOTES:
1. To configure P3.7 and P3.6 to their alternate input function (WAIT and CAP, respectively), either of the two input mode settings is valid.
2. To enable the WAIT or CAP input function, you must also make the appropriate settings in the EMT and PWMCON registers, respectively.
3. 'x' means don't care.

P3CONL — Port 3 Control Register (Low Byte) 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 and .6 **P3.3 /TDG /INT3 Configuration Bits**

0	0	Input mode, falling-edge interrupts, timer D gate input enabled
0	1	Input mode, rising-edge interrupts, timer D gate input enabled
1	0	Input mode, pull-up, falling-edge interrupts, timer D gate input enabled
1	1	Push-pull output mode

.5 and .4 **P3.2 / TCG /INT2 Configuration Bits**

0	0	Input mode, falling-edge interrupts, timer C gate input enabled
0	1	Input mode, rising-edge interrupts, timer C gate input enabled
1	0	Input mode, pull-up, falling-edge interrupts, timer C gate input enabled
1	1	Push-pull output mode

.3 and .2 **P3.1 / TDCK /INT1 Configuration Bits**

0	0	Input mode, falling-edge interrupts, timer D clock input enabled
0	1	Input mode, rising-edge interrupts, timer D clock input enabled
1	0	Input mode, pull-up, falling-edge interrupts, timer D clock input enabled
1	1	Push-pull output mode

.1 and .0 **P3.0 / TCCK /INT0 Configuration Bits**

0	0	Input mode, falling-edge interrupts, timer C clock input enabled
0	1	Input mode, rising-edge interrupts, timer C clock input enabled
1	0	Input mode, pull-up, falling-edge interrupts, timer C clock input enabled
1	1	Push-pull output mode

NOTES:
1. The alternate function for pins P3.3–P3.0 is configured when you select any one of the three input modes.
2. To enable the timer C and D gate input function, first configure P3.2 and P3.3 to input mode. Then, make the appropriate control bit settings in the T1MOD register.
3. To use pins P3.0 and P3.1 as timer C and D clock inputs, you first configure the pins in the P3CONL register. Then, to enable the clock input function (that is, to select the clock source), set the appropriate bits in T1MOD register.

P3INT — Port 3 Interrupt Enable Register FCH Set 1, Bank 0

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

.7 – .4

Not mapped for KS88C4116

.3 P3.3 External Interrupt (IRQ4) Enable Bit

0	Disable INT4 interrupt at P3.3
1	Enable INT4 interrupt at P3.3

.2 P3.2 External Interrupt (IRQ4) Enable Bit

0	Disable INT4 interrupt at P3.2
1	Enable INT4 interrupt at P3.2

.1 P3.1 External Interrupt (IRQ4) Enable Bit

0	Disable INT4 interrupt at P3.1
1	Enable INT4 interrupt at P3.1

.0 P3.0 External Interrupt (IRQ4) Enable Bit

0	Disable INT4 interrupt at P3.0
1	Enable INT4 interrupt at P3.0

NOTES:
1. External interrupts at the low-byte port 3 pins, P3.0–P3.3, are all interrupt level IRQ4.
2. The IRQ4 interrupts at P3.2 and P3.3 have the same priority level and vector address in the interrupt structure. In case of contention, the P3.2 interrupt is serviced first.

P3PND — Port 3 Interrupt Pending Register FDH Set 1, Bank 0

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

.7 – .4

Not mapped for KS88C4400

.3 P3.3 Interrupt (INT4) Pending Bit

0	No INT4 interrupt pending at P3.3 (when bit is read)
1	INT4 interrupt is pending at P3.3 (when bit is read)

.2 P3.2 Interrupt (INT4) Pending Bit

0	No INT4 interrupt pending at P3.2 (when bit is read)
1	INT4 interrupt is pending at P3.2 (when bit is read)

.1 P3.1 Interrupt (INT4) Pending Bit

0	No INT4 interrupt pending at P3.1 (when bit is read)
1	INT4 interrupt is pending at P3.1 (when bit is read)

.0 P3.0 Interrupt (INT4) Pending Bit

0	No INT4 interrupt pending at P3.0 (when bit is read)
1	INT4 interrupt is pending at P3.0 (when bit is read)

NOTES:
1. P3PND bits can be polled by application software to detect interrupt pending conditions.
2. To clear an interrupt pending condition, write a "1" to the P3PND register bit location; writing a "0" has no effect.
3. To avoid errors, we recommend using Load instructions (except for LDB) to manipulate the P3PND register.

P4CONH — Port 4 Control Register (High Byte) F6H 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 and .6 **P4.7 / INT11 Configuration Bits**

0	0	Input mode, interrupt on falling edges
0	1	Input mode, interrupt on rising edges
1	0	Input mode, interrupt on falling edges, pull-up resistor active
1	1	Push-pull output mode

.5 and .4 **P4.6 / INT10 Configuration Bits**

0	0	Input mode, interrupt on falling edges
0	1	Input mode, interrupt on rising edges
1	0	Input mode, interrupt on falling edges; pull-up resistor active
1	1	Push-pull output mode

.3 and .2 **P4.5 / INT9 Configuration Bits**

0	0	Input mode, interrupt on falling edges
0	1	Input mode, interrupt on rising edges
1	0	Input mode, interrupt on falling edges; pull-up resistor active
1	1	Push-pull output mode

.1 and .0 **P4.4 / INT8 Configuration Bits**

0	0	Input mode, interrupt on falling edges
0	1	Input mode, interrupt on rising edges
1	0	Input mode, interrupt on falling edges; pull-up resistor active
1	1	Push-pull output mode

P4CONL — Port 4 Control Register (Low Byte) F7H 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 and .6 **P4.3 / INT7 Configuration Bits**

0	0	Input mode, interrupt on falling edges
0	1	Input mode, interrupt on rising edges
1	0	Input mode, interrupt on falling edges, pull-up resistor active
1	1	Push-pull output mode

.5 and .4 **P4.2 / INT6 Configuration Bits**

0	0	Input mode, interrupt on falling edges
0	1	Input mode, interrupt on rising edges
1	0	Input mode, interrupt on falling edges, pull-up resistor active
1	1	Push-pull output mode

.3 and .2 **P4.1 / INT5 Configuration Bits**

0	0	Input mode, interrupt on falling edges
0	1	Input mode, interrupt on rising edges
1	0	Input mode, interrupt on falling edges, pull-up resistor active
1	1	Push-pull output mode

.1 and .0 **P4.0 / INT4 Configuration Bits**

0	0	Input mode, interrupt on falling edges
0	1	Input mode, interrupt on rising edges
1	0	Input mode, interrupt on falling edges, pull-up resistor active
1	1	Push-pull output mode

P4INT — Port 4 Interrupt Enable Register F9H 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 P4.7 / INT11 Interrupt Enable Bit

0	Disable INT11
1	Enable INT11

.6 P4.6 / INT10 Interrupt Enable Bit

0	Disable INT10
1	Enable INT10

.5 P4.5 / INT9 Interrupt Enable Bit

0	Disable INT9
1	Enable INT9

.4 P4.4 / INT8 Interrupt Enable Bit

0	Disable INT8
1	Enable INT8

.3 P4.3 / INT7 Interrupt Enable Bit

0	Disable INT7
1	Enable INT7

.2 P4.2 / INT6 Interrupt Enable Bit

0	Disable INT6
1	Enable INT6

.1 P4.1 / INT5 Interrupt Enable Bit

0	Disable INT5
1	Enable INT5

.0 P4.0 / INT4 Interrupt Enable Bit

0	Disable INT4
1	Enable INT4

P4PND — Port 4 Interrupt Pending Register D4H Set 1

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

.7 P4.7 / INT11 Interrupt Pending Bit

0	No interrupt pending (when bit is read)
1	Interrupt is pending (when bit is read)

.6 P4.6 / INT10 Interrupt Pending Bit

0	No interrupt pending (when bit is read)
1	Interrupt is pending (when bit is read)

.5 P4.5 / INT9 Interrupt Pending Bit

0	No interrupt pending (when bit is read)
1	Interrupt is pending (when bit is read)

.4 P4.4 / INT8 Interrupt Pending Bit

0	No interrupt pending (when bit is read)
1	Interrupt is pending (when bit is read)

.3 P4.3 / INT7 Interrupt Pending Bit

0	No interrupt pending (when bit is read)
1	Interrupt is pending (when bit is read)

.2 P4.2 / INT6 Interrupt Pending Bit

0	No interrupt pending (when bit is read)
1	Interrupt is pending (when bit is read)

.1 P4.1 / INT5 Interrupt Pending Bit

0	No interrupt pending (when bit is read)
1	Interrupt is pending (when bit is read)

.0 P4.0 / INT4 Interrupt Pending Bit

0	No interrupt pending (when bit is read)
1	Interrupt is pending (when bit is read)

NOTES:
1. To clear an interrupt pending condition, write a "1" to appropriate the P4PND bit.
2. To avoid errors, we recommend using Load instructions (except for LDB) to manipulate P4PND register values.

P5CON — Port 5 Control Register F8H 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 **Port 5 Upper Nibble (P5.4–P5.7) Configuration Bits**

x	0	x	0	Input mode
x	1	x	0	Input mode, pull-up resistor active
x	0	0	1	Push-pull output mode
x	0	1	1	N-channel, open-drain output mode
x	1	1	1	N-channel, open-drain output mode, pull-up resistor active

.3 – .0 **Port 5 Lower Nibble (P5.0 – P5.3) Configuration Bits**

x	0	x	0	Input mode
x	1	x	0	Input mode, pull-up resistor active
x	0	0	1	Push-pull output mode
x	0	1	1	N-channel, open-drain output mode
x	1	1	1	N-channel, open-drain output mode, pull-up resistor active

NOTE: 'x' means don't care.

PP — Register Page Pointer DFH Set 1

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 KS88C4400

.1 – .0 **Page Selection Bits (for KS88C4400 Register File Addressing)**

0	0	Page 0
0	1	Page 1
1	0	Page 2
1	1	Page 3

PWMCON — PWM Control Register FCH Set 1, Bank 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 and .6 2-Bit Prescaler Value for PWM Counter Input Clock

0	0	Divide input clock by one
0	1	Divide input clock by two
1	0	Divide input clock by three
1	1	Divide input clock by four

.5 PWM Counter Enable Bit

0	Stop PWM counter operation
1	Start (or resume) PWM counter operation

.4 PWM Counter Overflow Interrupt Enable Bit

0	Disable PWM counter overflow interrupt
1	Enable PWM counter overflow interrupt

.3 Capture Interrupt Enable Bit

0	Disable capture interrupt
1	Enable capture interrupt

.2 PWM Test Mode Enable Bit

This bit is used for factory testing only. During normal operation, PWMCON.2 should always remain cleared to "0".

.1 and .0 Data Capture Function Control Bits

0	0	Disable capture function
0	1	Capture on falling signal edge only
1	0	Capture on rising signal edge only
1	1	Capture on both rising and falling signal edges

NOTE: To avoid errors, we recommend using Load instructions (except for LDB) to modify PWMCON register values.

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 24 8-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 KS88C4400

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 24 8-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 KS88C4400

SIOCON — UART Control Register EAH 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 and .6 Mode and Baud Rate Selection Bits

0	0	Select SIO mode 0 (shift register, baud rate = CPU clock /6)
0	1	Select UART mode 1 (8-bit UART, variable baud rate)
1	0	Select UART mode 2 (9-bit UART, baud rate = CPU clock /16 or /32)
1	1	Select UART mode 3 (9-bit UART, variable baud rate)

.5 Multiprocessor Communication Enable Bit

0	Disable multiprocessor communication feature
1	Enable the multiprocessor communication feature in UART modes 2 and 3. If SIOCON.5 = "1", the receive interrupt bit (SIOCON.1) will *not* be set if the 9th data bit is "0". In UART mode 1, if SIOCON.5 = "1", the receive interrupt is only enabled when a valid stop bit is received. In SIO mode 0, SIOCON.5 should always be "0".

.4 Serial Data Receive Enable Bit

0	Disable serial data receive function
1	Enable serial data receive function

.3 Value of the 9th Bit To Be Transmitted in UART Mode 2 or 3

0	Transmit a "0" as the 9th data bit (valid for UART modes 2 or 3 only)
1	Transmit a "1" as the 9th data bit (valid for UART modes 2 or 3 only)

.2 Value of the 9th Bit That Was Received in UART Mode 2 or 3

In SIO modes 2 and 3, SIOCON.2 is the 9th data bit that was received (including the start bit). In mode 1, if SIOCON.5 = "0", SIOCON.2 is the Stop bit. In mode 0, SIOCON.2 is not used because a Start bit is not required.

.1 UART Receive Interrupt Enable Bit

0	Disable UART receive interrupt
1	Enable UART receive interrupt

.0 UART Transmit Interrupt Enable Bit

0	Disable UART transmit interrupt
1	Enable UART transmit interrupt

SIOPND — UART Interrupt Pending Register EBH Set 1, Bank 0

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 KS88C4400

.1 **UART Receive Interrupt Pending Flag**

0	No UART receive interrupt is pending (when bit is read)
1	UART receive interrupt is pending (when bit is read)

.0 **UART Transmit Interrupt Pending Flag**

0	No UART transmit interrupt is pending (when bit is read)
1	UART transmit interrupt is pending (when bit is read)

NOTES:
1. To clear an interrupt pending condition, you must write a "1" to the appropriate SIOPND bit location.
2. In order to avoid programming errors, we recommend that you use Load instructions only (except for LDB) to modify SIOPND register values.

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.

NOTE: If you only use the internal register file as stack area, SPH can serve as a general-purpose register. To avoid possible overflows or underflows of the SPL register by operations that increment or decrement the stack, we recommend that you initialize SPL with the value 'FFH' instead of '00H'. If you use external memory as stack area, the stack pointer requires a full 16-bit address.

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.

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 Tri-State External Interface Control Bit

0	Normal operation (disable tri-state operation)
1	Set external interface lines to high impedance (enable tri-state operation)

.6 and .5

Not used for KS88C4400

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

0	0	0	IRQ0 (timer B overflow)
0	0	1	IRQ1 (timer A overflow only)
0	1	0	IRQ2 (invalid selection; not used in KS88C4400)
0	1	1	IRQ3 (serial Rx/Tx, timer C and timer D overflow)
1	0	0	IRQ4 (INT0–INT3 external interrupts at P3.0–P3.3)
1	0	1	IRQ5 (INT4 external interrupt at P4.0)
1	1	0	IRQ6 (INT5–INT7 external interrupts at P4.1–P4.3)
1	1	1	IRQ7 (INT8–INT11 external interrupts at P4.4–P4.7)

.1 Fast Interrupt Enable Bit

0	Disable fast interrupt processing
1	Enable fast interrupt processing

.0 Global Interrupt Enable Bit (2)

0	Disable global interrupt processing
1	Enable global interrupt processing

NOTES:
1. Fast interrupt processing is only available for interrupt levels whose pending bits are cleared by software. Fast interrupts are therefore not available for the PWM counter overflow and capture data input interrupts (IRQ1).
2. Following a reset, you enable global interrupt processing by executing an EI instruction (not by writing a "1" to SYM.0).

T0CON — Timer 0 Control Register EEH 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 (1)	W	W	W	W	W	W	R/W	W
Addressing Mode	Register addressing mode only							

.7 – .4 4-Bit Prescaler Value for Timer A and B Clock Input

0	0	0	0	Divide clock input by 1 (non-divided)
0	0	0	1	Divide clock input by 2
•	•	•	•	(Divide clock input by 3–15)
1	1	1	1	Divide clock input by 16

.3 Timer A and B Clock Source Selection Bit

0	Select divided-by-1024 CPU clock
1	Select non-divided CPU clock

.2 Timer A Overflow Interrupt Enable Bit

0	Disable timer A interrupt requests
1	Enable timer A interrupt requests

.1 Timer A Overflow Interrupt Pending Flag (2)

0	No timer A interrupt is pending (when bit is read)
1	Timer A interrupt is pending (when bit is read)

.0 Timer A Operating Mode Selection Bit

0	Interval timer mode
1	Pulse width modulation mode (3)

NOTES:
1. To avoid programming errors, we recommend that you use Load instructions only (except for LDB) to modify T0CON register values.
2. T0CON.1 is the only readable bit in the T0CON register and can therefore be polled to detect timer A interrupt pending conditions. To clear a timer A interrupt pending condition, you must write a "1" to T0CON.1. Writing a "0" has no effect.
3. For PWM output mode, you must also enable the timer A output pin (P2.6) by setting P2CON bits 5 and 4 to '11B'.

T1CON — Timer Module 1 Control Register FAH 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 only							

.7 **Double Baud Rate Selection Bit** (1)

0	Normal baud rate for timer D UART feature
1	Double baud rate for timer D UART feature

.6 Not used for KS88C4400

.5 **Timer D Overflow Interrupt Pending Flag** (2)

0	No timer D interrupt is pending (when bit is read)
1	Timer D interrupt is pending (when bit is read)

.4 **Timer C Overflow Interrupt Pending Flag** (2)

0	No timer C interrupt is pending (when bit is read)
1	Timer C interrupt is pending (when bit is read)

.3 **Timer D Overflow Interrupt Enable Bit**

0	Disable timer D interrupt
1	Enable timer D interrupt

.2 **Timer C Overflow Interrupt Enable Bit**

0	Disable timer C interrupt
1	Enable timer C interrupt

.1 **Timer D Run Enable Bit**

0	Disable timer D (stop)
1	Enable timer D (run)

.0 **Timer C Run Enable Bit**

0	Disable timer C (stop)
1	Enable timer C (run)

NOTES:
1. Timer D overflows can be used as a baud rate generator for the UART module. To configure this feature, the timer D interrupt must first be disabled. Please refer to the hardware descriptions in Part II of this manual for more information.
2. To clear a timer C or timer D interrupt pending condition, you must write a "1" to the appropriate pending flag.
3. To avoid programming errors, we recommend using only Load instructions (except for LDB) to manipulate T1CON register values.

T1MOD — Timer Module 1 Mode 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 Timer D Gate Function Enable Bit (1)

0	Disable timer D gate function
1	Enable timer D gate function

.6 Timer D Clock Source Selection Bit (2)

0	Divided-by-six CPU clock (for timer operation)
1	External clock source (for event counter operation)

.5 and .4 Timer D Operating Mode Selection Bits

0	0	Cascaded 13-bit timer/counter
0	1	16-bit timer/counter
1	0	8-bit auto-reload timer/counter
1	1	Disable timer/counter D

.3 Timer C Gate Function Enable Bit (1)

0	Disable timer C gate function
1	Enable timer C gate function

.2 Timer C Clock Source Selection Bit (2, 3)

0	Divided-by-six CPU clock (for timer operation)
1	External clock source (for event counter operation)

.1 and .0 Timer C Operating Mode Selection Bits

0	0	Cascaded 13-bit timer/counter
0	1	16-bit timer/counter
1	0	8-bit auto-reload timer/counter
1	1	Two 8-bit timer/counters

NOTES:
1. Before the timer C or D gate function is enabled, you must first configure the appropriate input pins in the P4CONL control register: P4.0 (bit-pair 1/0) for the timer C gate and P4.1 (bit-pair 3/2) for the timer D gate.
2. Before the timer C or D clock source selection is enabled, you must first configure the appropriate pins in the P3CONL control register: P3.0 (bit-pair 1/0) for the timer C pin TCCK and P3.1 (bit-pair 3/2) for the timer D pin TDCK.
3. The CPU clock frequency is equal to the oscillator clock frequency.

TBCON — Timer B Control Register EFH Set 1, Bank 0

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

.7 – .3

Not used for KS88C4400

.2 **Timer B Overflow Interrupt Enable Bit**

0	Disable timer B interrupt
1	Enable timer B interrupt

.1 **Timer B Overflow Interrupt Pending Flag** (2)

0	No timer B interrupt is pending (when bit is read)
1	Timer B interrupt is pending (when bit is read)

.0 **Timer B Operating Mode Selection Bit**

0	Interval timer mode
1	Pulse width modulation mode (3)

NOTES:
1. To avoid programming errors, we recommend that you use Load instructions only (except for LDB) to modify TBINT register values.
2. TBINT.1 is the only readable bit in this register. You can therefore poll TBINT.1 to detect timer B interrupt pending conditions. To clear a timer B pending condition, you must write a "1" to TBINT.1. Writing a "0" has no effect.
3. For PWM output mode, you must also enable the timer B output pin (P2.7) by setting P2CON bits 6 and 7 to '11B'.

NOTES

5 Interrupt Structure

OVERVIEW

The SAM8 interrupt structure has three basic components: levels, vectors, and sources. The CPU recognizes 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. Each vector can have one or more sources.

Levels

Levels provide the highest-level method of 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 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 interrupt level numbers 0 through 7 do not necessarily indicate the relative priority of the levels. They are simply identifiers for the interrupt levels that are recognized by the CPU (IRQ0–IRQ7).

The relative priority of different interrupt levels is determined by settings in the interrupt priority register, IPR. The interrupt group and subgroup logic that is controlled by IPR settings lets you define more complex priority relationships.

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 KS88-series devices will always be much smaller.) If an interrupt level has more than one vector address, the relative vector priorities are set in hardware.

Sources

A source is any peripheral that generates an interrupt. A source can be an external pin or a counter overflow, for example. Each vector can have several interrupt sources. When a service routine starts, the respective pending bit is either cleared automatically by hardware or "manually" by the application software.

The characteristics of the source's pending mechanism determine which method is used to clear its corresponding pending bit.

INTERRUPT TYPES

The three components of the SAM8 interrupt structure described above — 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):

Interrupt Types (Continued)

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 KS88C4400 microcontroller, only interrupt types 1 and 3 are implemented. In interrupt level IRQ4, two sources (P3.2 and P3.3 external interrupt) do share the same vector (ECH). However, IRQ4 has the basic type 3 interrupt structure.

KS88C4400 INTERRUPT STRUCTURE

The KS88C4400 microcontroller supports up to 20 interrupt sources. Each interrupt source has a corresponding interrupt vector address.

Nineteen different vector addresses are allocated to these 20 interrupt sources. (The reason why there are not 20 vectors is that the P3.3 and P3.2 external interrupt sources in level IRQ4 share the same vector address, ECH.)

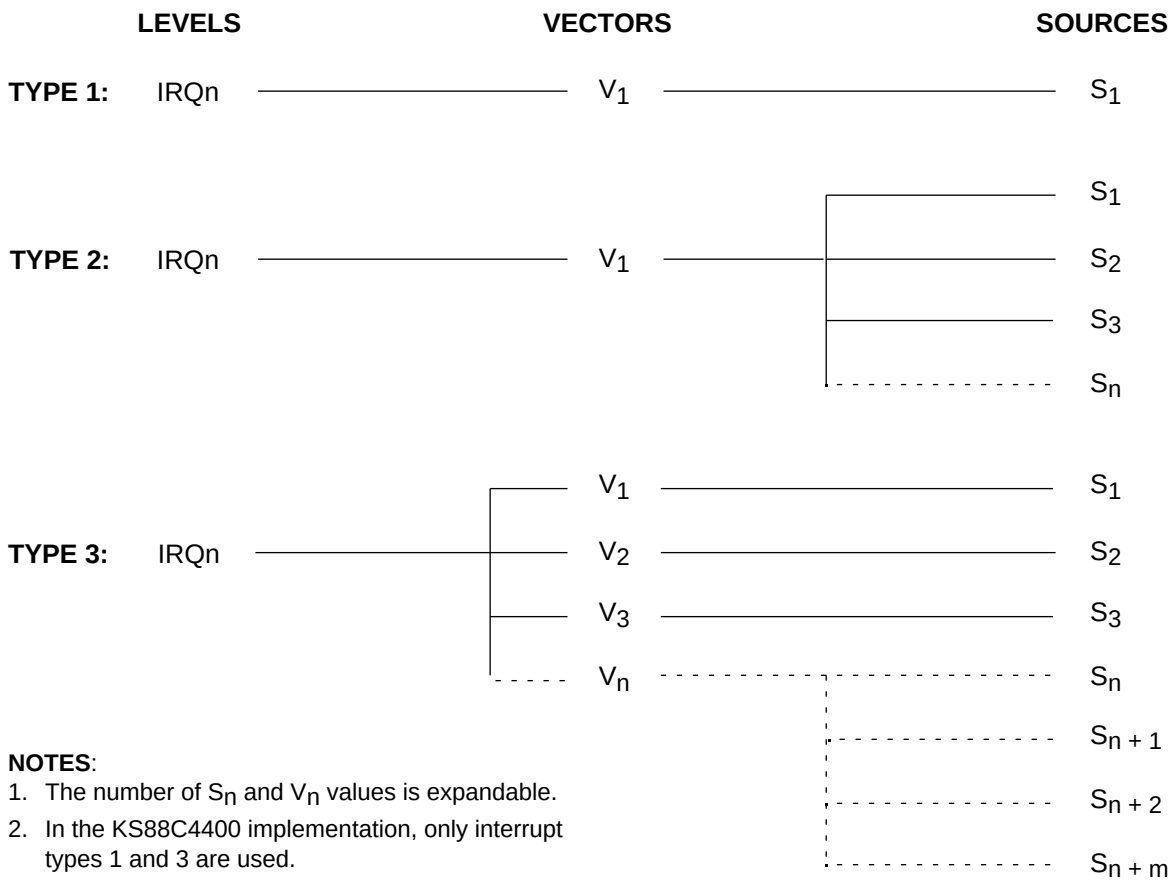
Only seven levels are used in the KS88C4400's interrupt structure: IRQ0, IRQ1, and IRQ3–IRQ7. Interrupt level IRQ2 is not implemented. The device-specific interrupt structure is shown in Figure 5–2.

When multiple interrupt levels are active in a instruction simultaneously, 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 set in hardware.)

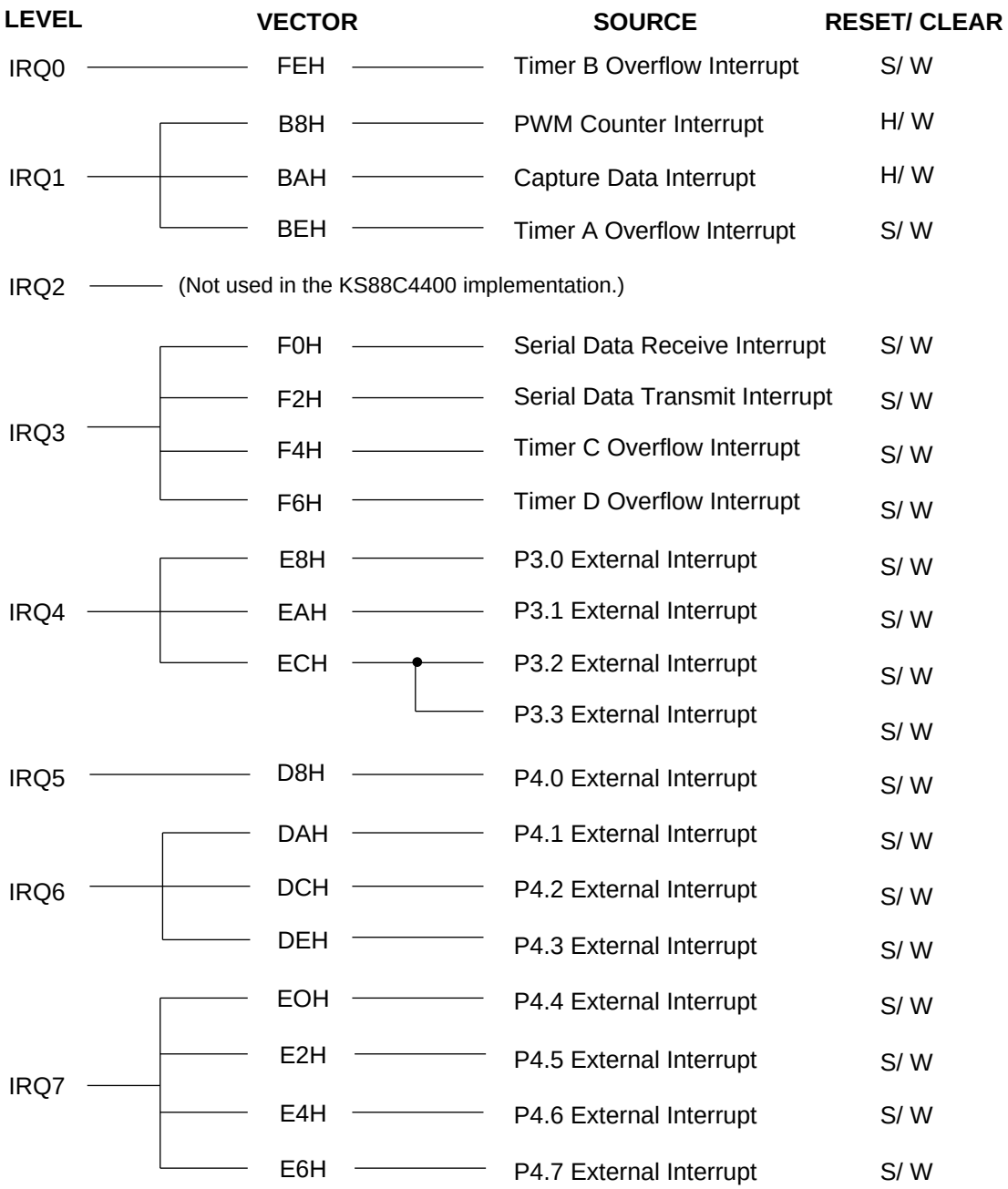
When an interrupt request is granted, an interrupt machine cycle is entered. This disables all subsequent interrupts, saves the program counter and status flags, and branches to the program memory vector location reserved for that interrupt.

This memory location, together with the next memory byte, constitutes the 16-bit address of the interrupt service routine for that particular interrupt request.



NOTES:
1. The number of S_n and V_n values is expandable.
2. In the KS88C4400 implementation, only interrupt types 1 and 3 are used.

Figure 5–1. KS88-Series Interrupt Types



NOTES:

1. Within a given interrupt level, the interrupt with the lower vector address has the higher priority. For example, E0H has higher priority than E2H within IRQ7. These priorities are set in hardware.
2. The P3.3 and P3.2 external interrupts in IRQ4 share the same vector: ECH.
3. External interrupts may be triggered by a rising or falling edge, based on the corresponding control register setting.

Figure 5–2. KS88C4400 Interrupt Structure

Interrupt Vector Addresses

Interrupt vector addresses for the KS88C4400 are stored in the first 256 bytes of the external program memory (ROM). Vectors for all interrupt levels are stored in the vector address area, 0H–FFH.

Unused locations in this range can be used as normal program memory. When writing an application program, you should be careful not to overwrite the address data stored in this area.

The program reset address in the external program memory is 0020H. When a reset occurs, the 16-bit address stored in this location (and in 0021H) is fetched and the corresponding instruction is executed.

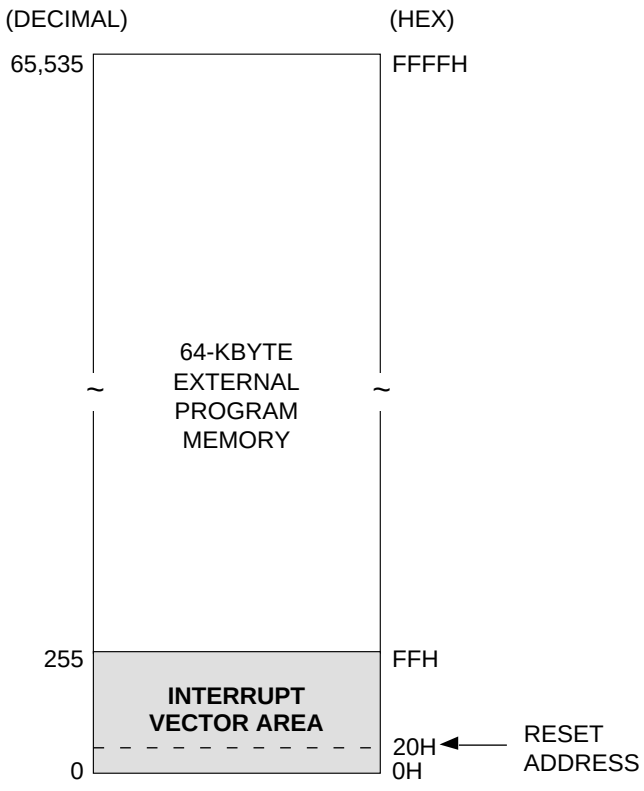


Figure 5–3. Vector Address Area in External Program Memory (ROM)

Table 5–1. KS88C4400 Interrupt Vectors

Vector Address		Interrupt Source	Request		Reset/Clear	
Decimal Value	Hex Value		Interrupt Level	Priority in Level	H/W	S/W
254	FEH	Timer B overflow	IRQ0	–		√
246	F6H	Timer D overflow	IRQ3	3		√
244	F4H	Timer C overflow		2		
242	F2H	Serial data transmit		1		
240	F0H	Serial data receive		0		
236	ECH	P3.3 external interrupt	IRQ4	2		√
(also 236)	(also ECH)	P3.2 external interrupt		2		
234	EAH	P3.1 external interrupt		1		
232	E8H	P3.0 external interrupt		0		
230	E6H	P4.7 external interrupt	IRQ7	3		√
228	E4H	P4.6 external interrupt		2		
226	E2H	P4.5 external interrupt		1		
224	E0H	P4.4 external interrupt		0		
222	DEH	P4.3 external interrupt	IRQ6	2		√
220	DCH	P4.2 external interrupt		1		
218	DAH	P4.1 external interrupt		0		
216	D8H	P4.0 external interrupt	IRQ5	–		√
190	BEH	Timer A overflow	IRQ1	2		√
186	BAH	Capture data input		1	√	
184	B8H	PWM counter overflow		0	√	

NOTES:
1. Interrupt priorities are identified in inverse order: '0' is highest priority, '1' is the next highest, and so on.
2. If two or more interrupts within the same level contend, the interrupt with the lowest vector address has priority over one with a higher vector address. The priorities within a level are preset at the factory. For example, in level IRQ3, the highest priority interrupt (0) is the serial data receive interrupt, vector F0H; the lowest priority interrupt (3) within the same level is the timer D interrupt, vector F6H.
3. The P3.3 and P3.2 interrupt sources share the same interrupt vector: ECH (decimal 236). You can identify P3.2 & P3.3 interrupt source using P3PND.2 & P3PND.3.

Enable/Disable Interrupt Instructions (EI, DI)

The Enable Interrupts (EI) instruction globally enables the interrupt structure. All interrupts are serviced as they occur, and according to established priorities. The system initialization routine that is executed following a reset must always contain an EI instruction.

During 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. Although you can manipulate SYM.0 directly to enable or disable interrupts, we recommend that you use the EI and DI instructions instead.

SYSTEM-LEVEL INTERRUPT CONTROL REGISTERS

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

— An interrupt level is enabled or disabled (masked) by bit settings in the interrupt mask register (IMR).

— Relative priorities of interrupt levels are controlled by the interrupt priority register (IPR).

— The interrupt request register (IRQ) contains interrupt pending flags for each level.

— The system mode register (SYM) dynamically enables or disables global interrupt processing. SYM settings also enable fast interrupts and control the external interface, if implemented.

Table 5–2. Interrupt Control Register Overview

Control Register	ID	R/W	Function Description
System mode register	SYM	R/W	Dynamic global interrupt processing enable and disable, fast interrupt processing, and external interface control.
Interrupt mask register	IMR	R/W	Bit settings in the IMR register enable or disable interrupt processing for each of the seven interrupt levels, IRQ0, IRQ1, and IRQ3–IRQ7. (IRQ2 is not implemented.)
Interrupt priority register	IPR	R/W	Controls the relative processing priorities of the interrupt levels. The eight levels are organized into three groups: A, B, and C. Group A is IRQ0 and IRQ1, group B is IRQ3 and IRQ4, and group C is IRQ5–IRQ7. (IRQ2 is not implemented.)
Interrupt request register	IRQ	R	This register contains a request pending bit for each of the seven interrupt levels that are used in the KS88C4400 interrupt structure, IRQ0, IRQ1, and IRQ3–IRQ7. (IRQ2 is not implemented.)

Interrupt Processing Control Points

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

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

— Interrupt level enable and disable settings (IMR register)

— Interrupt level priority settings (IPR register)

— Interrupt source enable and disable settings in the corresponding peripheral control register(s)

NOTE

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

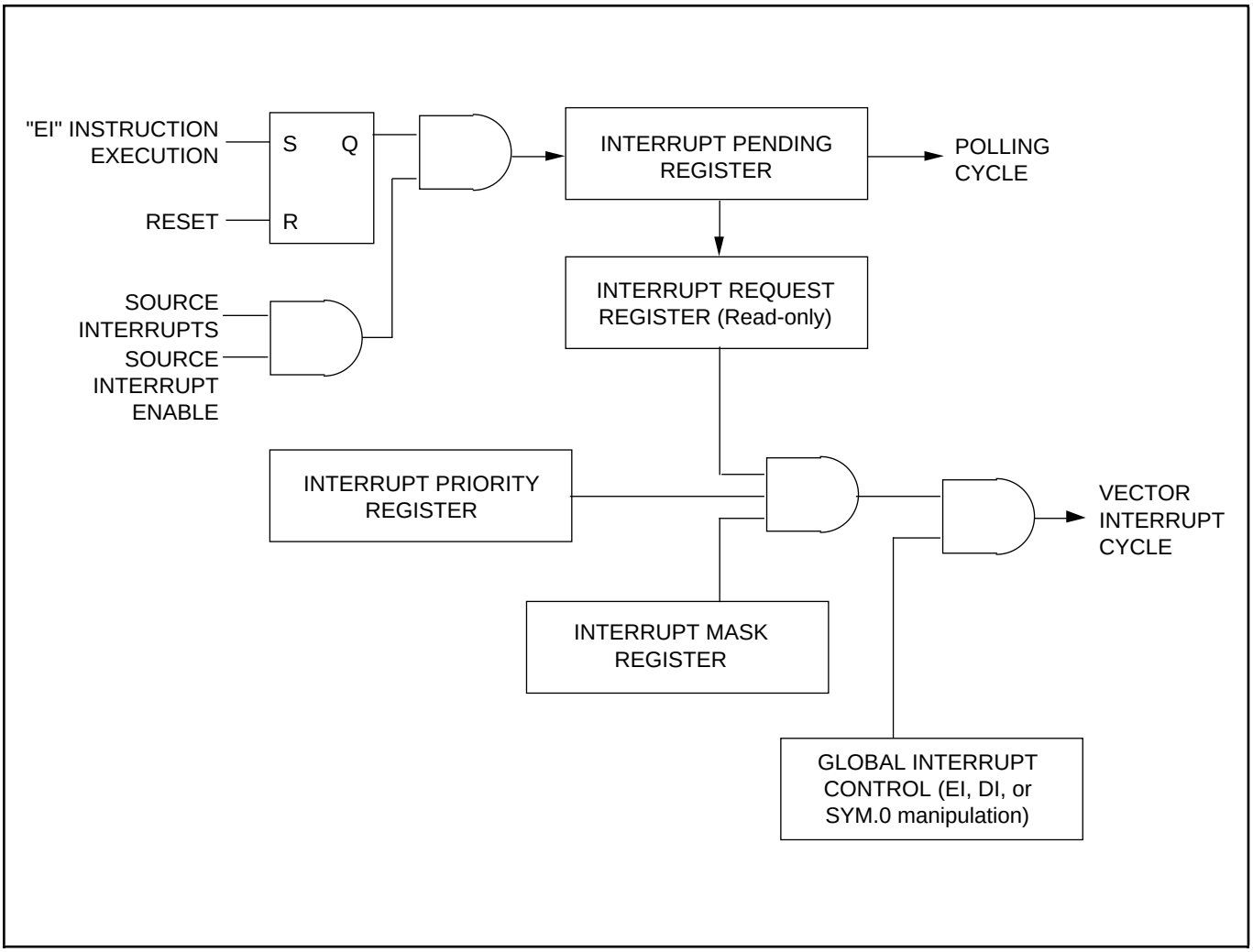


Figure 5–4. Interrupt Function Diagram

Peripheral Interrupt Control Registers

For each interrupt source there is a corresponding peripheral control register (or registers) to control the interrupts generated by that peripheral. These registers and their locations are listed in Table 5–3.

Table 5–3. Peripheral Interrupt Source Overview

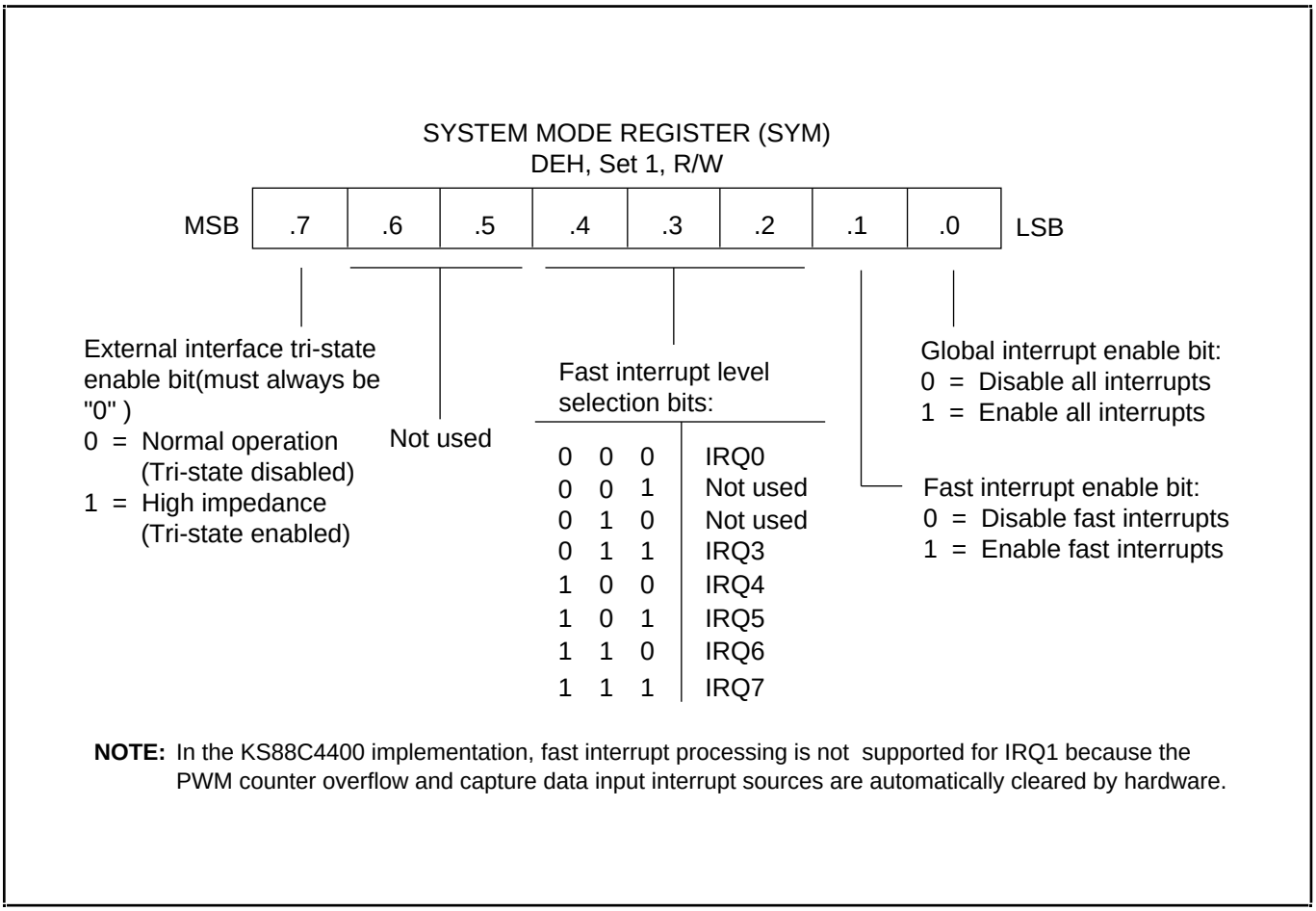
Peripheral Source	Interrupt Level	Control Register(s)	Register Location
Timer B overflow	IRQ0	TBCON	EFH (set 1, bank 0)
Timer D overflow	IRQ3	T1CON	FAH (set 1, bank 0)
Timer C overflow		T1MOD	FBH (set 1, bank 0)
Serial data transmit interrupt		SIOCON	EAH (set 1, bank 0)
Serial data receive interrupt		SIOPND	EBH (set 1, bank 0)
P3.3 external interrupt	IRQ4	P3CONL	F5H (set 1, bank 0)
P3.2 external interrupt		P3INT	FCH (set 1, bank 0)
P3.1 external interrupt		P3PND	FDH (set 1, bank 0)
P3.0 external interrupt			
P4.7 external interrupt	IRQ7	P4CONH	F6H (set 1, bank 0)
P4.6 external interrupt		P4INT	F9H (set 1, bank 0)
P4.5 external interrupt		P4PND	D4H (set 1)
P4.4 external interrupt			
P4.3 external interrupt	IRQ6	P4CONL	F7H (set 1, bank 0)
P4.2 external interrupt		P4INT, P4PND	Same as IRQ7
P4.1 external interrupt			
P4.0 external interrupt	IRQ5	P4CONL, P4INT, P4PND	Same as IRQ6
Timer A overflow	IRQ1	T0CON	EEH (set 1, bank 0)
Capture data input	IRQ1	PWMCON	FCH (set 1, bank 1)
PWM counter overflow			

System Mode Register (SYM)

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

SYM.0 is the enable and disable bit for global interrupt processing. SYM.1–SYM.4 control fast interrupt processing: SYM.1 is the enable bit; SYM.2–SYM.4 are the fast interrupt level selection bits. SYM.7 is the enable bit for the external memory interface's tri-state function. A reset clears SYM.0, SYM.1, and SYM.7 to "0"; the other bit values are undetermined.

The instructions EI and DI enable and disable global interrupt processing, respectively, by modifying SYM.0. An Enable Interrupt (EI) instruction must be included in the initialization routine that follows a reset operation in order to enable interrupt processing. Although you can manipulate SYM.0 directly to enable and disable interrupts during normal operation, we recommend that you use the EI and DI instructions for this purpose.



NOTE: In the KS88C4400 implementation, fast interrupt processing is not supported for IRQ1 because the PWM counter overflow and capture data input interrupt sources are automatically cleared by hardware.

Figure 5–5. System Mode Register (SYM)

Interrupt Mask Register (IMR)

The interrupt mask register (IMR) is used to enable or disable interrupt processing for the seven interrupt levels that are used in the KS88C4400 interrupt structure, IRQ0, IRQ1, and IRQ3–IRQ7. After a reset, all IMR register values are undetermined.

Each IMR bit corresponds to a specific interrupt level: bit 0 to IRQ0, bit 1 to IRQ1, 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. Before you write the IMR register, you should disable global interrupt processing by executing a DI instruction.

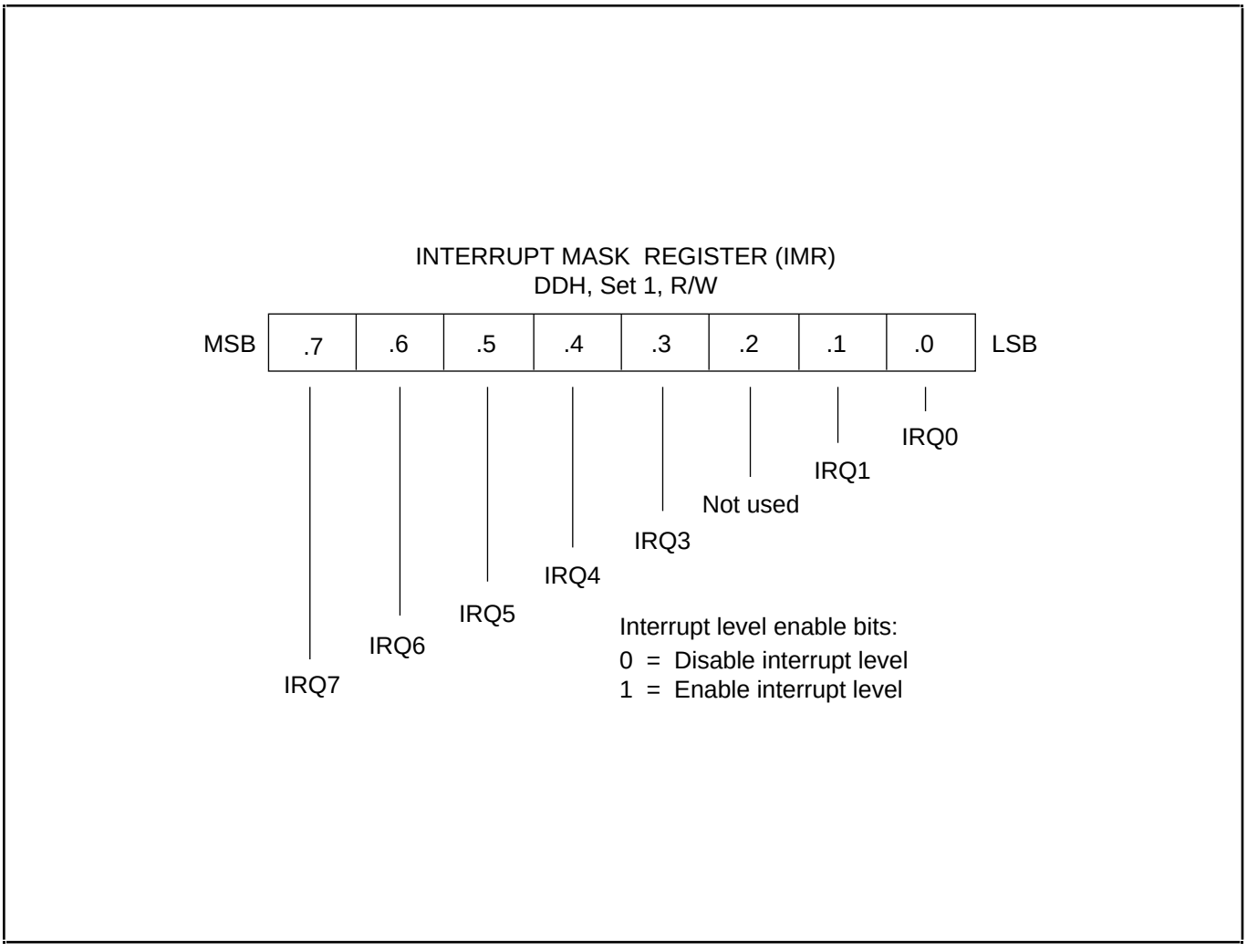


Figure 5–6. Interrupt Mask Register (IMR)

Interrupt Priority Register (IPR)

The interrupt priority register, IPR, is used to set the relative priorities of the seven interrupt levels that are used in the KS88C4400 interrupt structure (IRQ0, IRQ1, and IRQ3–7). The IPR register is mapped to register location FFH in set 1, bank 0.

After a reset, all IPR register values are undetermined. If more than one interrupt source is active following the reset, the source with the highest priority level is serviced first. If both sources belong to the same interrupt level, the source with the lowest vector address is usually assigned the highest priority. (The priority is set in hardware.)

In order to define the relative priorities of interrupt levels, they are organized into groups and subgroups by the interrupt logic. Three interrupt groups are defined for the IPR logic (these groups and subgroups are used only for IPR register priority definitions):

Group A IRQ0 and IRQ1

Group B IRQ3 and IRQ4

Group C IRQ5, IRQ6, and IRQ7

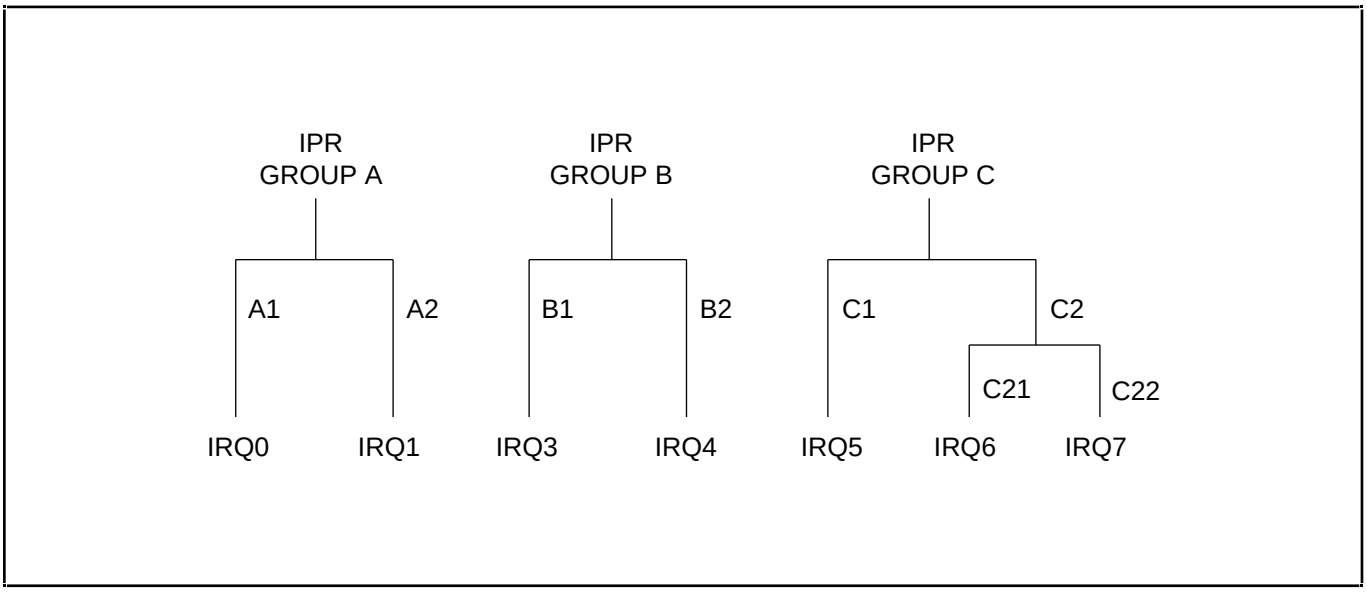


Figure 5–7. Interrupt Request Priority Groups

Bits 7, 4, and 1 of the IPR register control the relative priority of interrupt groups A, B, and C. For example, the setting '001B' would select the group relationship B > C > A, and '101B' would select C > B > A.

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

— IPR.0 controls the relative priority setting of group A interrupts (levels IRQ0 and IRQ1).

— IPR.2 is not used in the KS88C4400 implementation.

— IPR.3 controls the relative priorities of group B interrupts (levels IRQ3 and IRQ4)..

— IPR.5 controls the relative priorities of group C interrupts (levels IRQ5, IRQ6, and IRQ7).

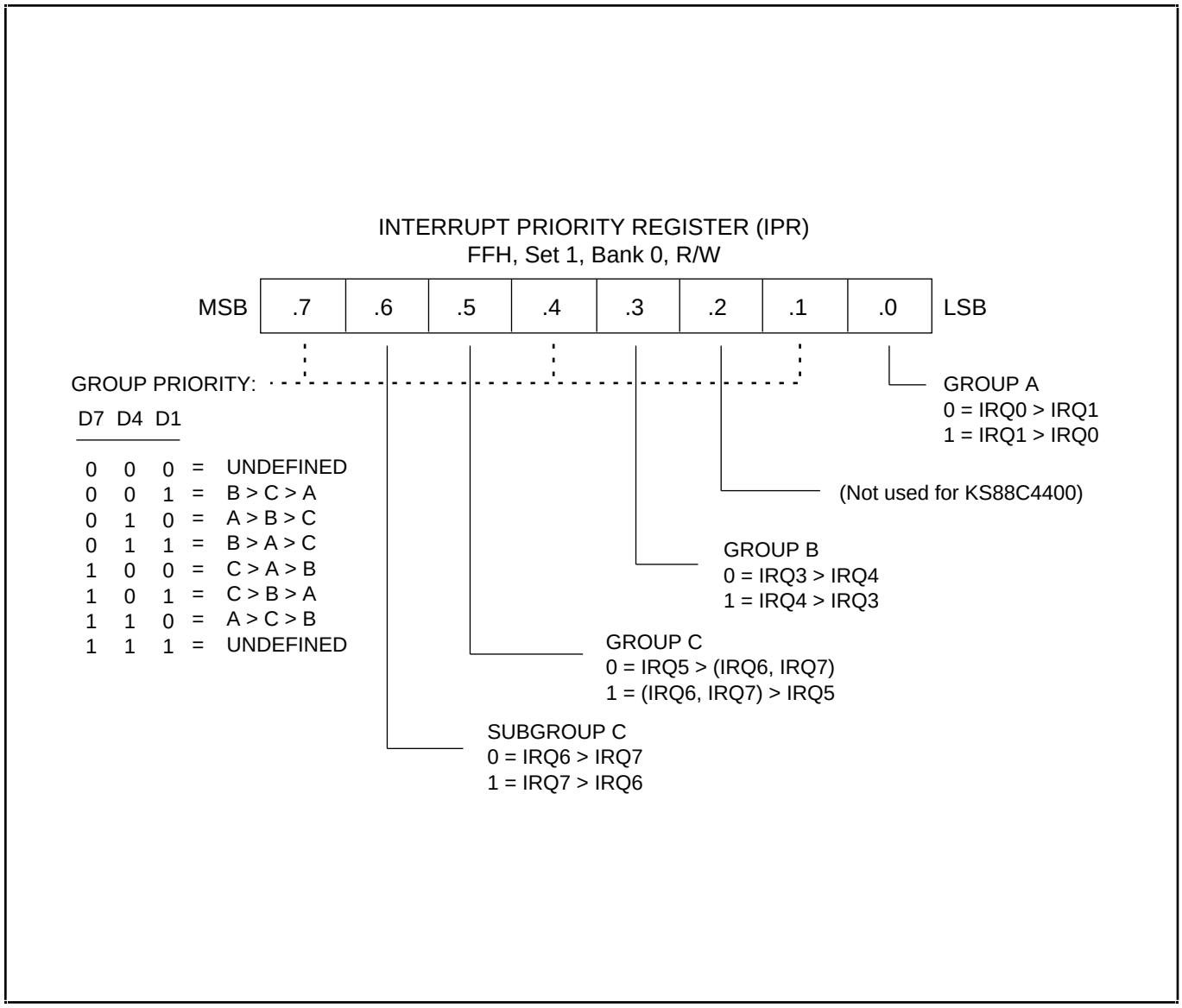


Figure 5–8. Interrupt Priority Register (IPR)

Interrupt Request Register (IRQ)

Bit values in the interrupt request register, IRQ, are polled to determine interrupt request status for the seven interrupt levels that are used in the KS88C4400 interrupt structure, IRQ0, IRQ1, and IRQ3–IRQ7.

Each bit corresponds to the interrupt level of the same number: bit 0 to IRQ0, bit 1 to IRQ1, and so on. A "0" indicates that no interrupt is currently requested and a "1" indicates that an interrupt is currently being requested for that level.

The IRQ register is mapped to register location DCH in set 1. 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, the IRQ register is cleared to '00H'.

IRQ register values can continue to be polled even if a DI instruction has been executed. If an interrupt occurs while the interrupt structure is disabled, it will not be serviced by the CPU.

The interrupt request will, however, be detected by the IRQ polling procedure. When an EI instruction is executed to enable interrupt processing, you can use this feature to determine which system events occurred while the interrupt structure was disabled.

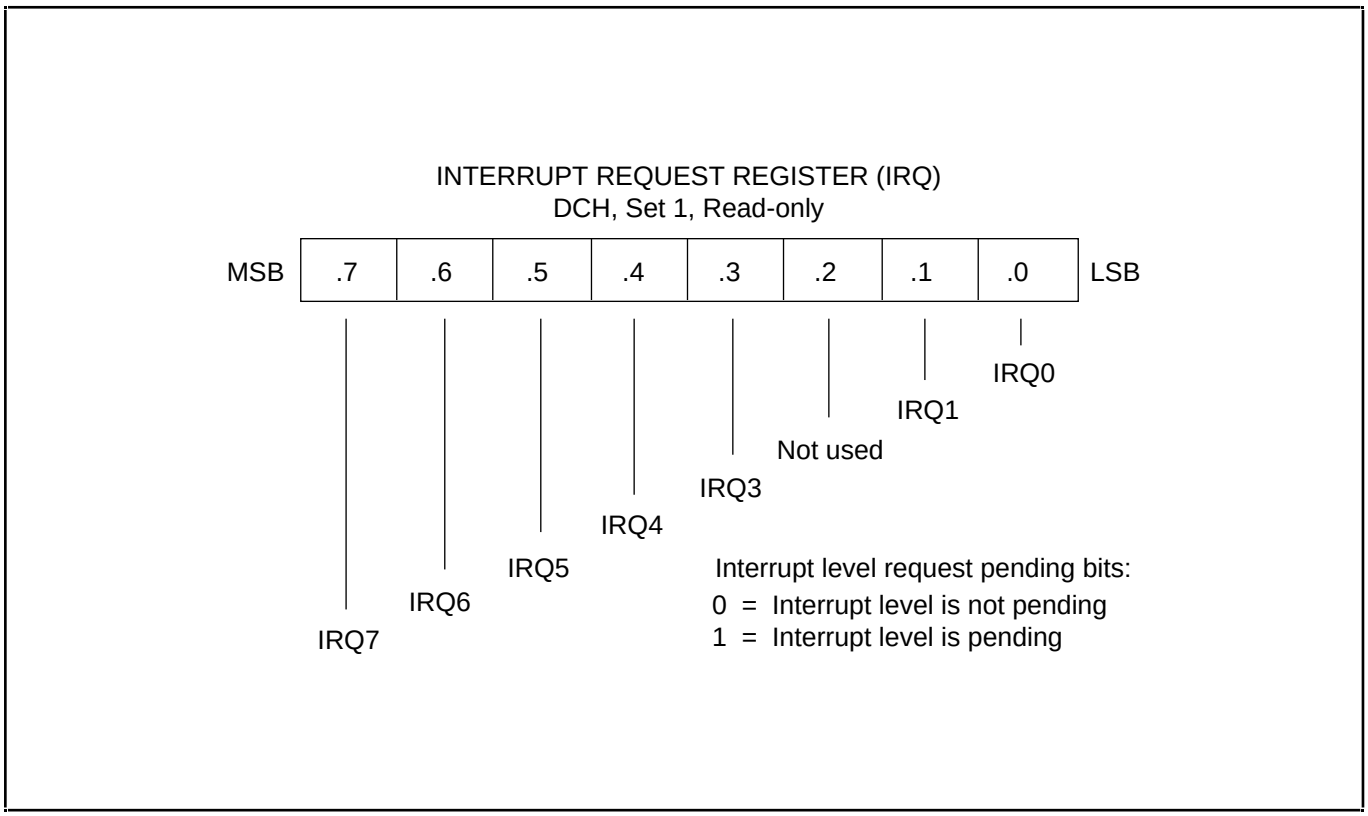


Figure 5–9. Interrupt Request Register (IRQ)

Interrupt Pending Function Types

Overview

There are two types of interrupt pending bits. One type is automatically cleared by hardware after the interrupt service routine is acknowledged and executed. The other type must be cleared by the application program's interrupt service routine.

Each interrupt level has a corresponding interrupt request bit in the IRQ register that the CPU polls for interrupt requests.

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 is detected.

It then issues an IRQ pulse to tell the CPU that an interrupt is waiting to be serviced. The CPU acknowledges the interrupt source, 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 software.

In the KS88C4400 interrupt structure, only the PWM counter overflow interrupt (IRQ1, vector B8H) and the capture data input interrupt (IRQ1, vector BAH) are cleared automatically by hardware.

Pending Bits Cleared by the Service Routine

The second type of pending bit must 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, you must write a logic "1" to the pending bit location in the corresponding mode or control register.

NOTE

If you execute an EI instruction immediately after clearing the pending bit, the EI will execute faster than the pending bit clear operation.

☞ PROGRAMMING TIP — Using Load Instructions to Manipulate Interrupt Pending Registers

Use only load (LD) instructions, except for LDB, to manipulate interrupt pending registers. For example, you cannot use an AND, OR, or LDB instruction to selectively clear bit 1 in the port 4 interrupt pending register, P4PND. The following examples show invalid use of logic instructions:

```
AND     P4PND,#0FDH     ;  Invalid use of logical AND instruction!
OR      P4PND,#02H      ;  Invalid use of logical OR instruction!
LDB     P4PND.1,R0      ;  Invalid use of LDB (Load Bit) instruction!
```

Use a LD instruction instead to manipulate pending register bits:

```
LD      P4PND,#02H      ;  Only bit 1 will be reset. (Writing a "1" clears the
                        ;  pending bit; writing a "0" has no effect.)
```

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 flag is cleared to "0" (either by hardware or by software).
7. The CPU continues polling for interrupt requests.

INTERRUPT SERVICE ROUTINES

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

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

If all of 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 and status flags to stack.
3. Branch to the interrupt vector to fetch the service routine's address.
4. Pass control to the interrupt service routine.

When the interrupt service routine is completed, an Interrupt Return instruction (IRET) occurs. The IRET restores the PC and status flags and sets SYM.0 to "1", allowing the CPU to process the next interrupt request.

Generating Interrupt Vector Addresses

The interrupt vector area in the ROM contains the addresses of the interrupt service routine that corresponds to each level in the interrupt structure. Vectored interrupt processing follows this sequence:

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

NOTE

A 16-bit vector address always begins at an even-numbered ROM location from 00H–FFH.

NESTING OF VECTORED INTERRUPTS

You can 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 to enable the higher priority interrupt only.
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 from the stack (POP IMR).
5. Execute an IRET.

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

INSTRUCTION POINTER (IP)

The instruction pointer (IP) is used by all KS88-series microcontrollers to control optional high-speed interrupt processing called *fast interrupts*. The IP consists of register pair DAH and DBH. The IP register names are IPH (high byte, IP15–IP8) and IPL (low byte, IP7–IP0).

Fast Interrupt Processing

The feature called *fast interrupt processing* lets designated interrupts be completed in approximately six clock cycles instead of the usual 22 clock cycles. Bit 1 of the system mode register, SYM.1, enables fast interrupt processing while SYM.2–SYM.4 are used to select a specific interrupt level for fast processing.

Two other system registers support fast interrupts:

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

— A dedicated register, FLAGS', saves the contents of the FLAGS register when a fast interrupt occurs.

NOTE

For the KS88C4400 microcontroller, the service routine for any interrupt level whose pending condition is cleared by software can be designated as a fast interrupt. Interrupts in levels IRQ1 and IRQ3 cannot be serviced as fast interrupts. Although the timer A overflow interrupt, which is level IRQ1, is cleared by software, the other two IRQ1 interrupts, the PWM counter interrupt and the capture data input interrupt, are automatically cleared by hardware. For this reason, none of the interrupts in this level can be selected as fast interrupts.

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.
2. Load the level number into the fast interrupt select field.
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 dedicated FLAGS' 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 into the FLAGS register.
7. The fast interrupt status bit in FLAGS is cleared automatically.

Programming Guidelines

Remember that the only way to enable or disable a fast interrupt is to set or clear the fast interrupt enable bit in the SYM register (SYM.1), respectively. Executing an EI or DI instruction affects only normal interrupt processing.

Also, if you use fast interrupts, remember to load the IP with a new start address when the fast interrupt service routine ends.

☞ Programming Tip — Setting Up the Interrupt Control Structure

This example shows you how to enable interrupts for select interrupt sources, disable interrupt for other sources, and to set interrupt priorities for the KS88C4400. The sample program does the following:

— Enable interrupts for port 4 pins P4.4–P4.7, and for timer A and timer C overflows

— Disable timer B, timer D, UART, P3.0–P3.3, P4.0–P4.3, PWM, and capture input interrupts

— Set interrupt priorities as P4.4–P4.7 > timer A > timer C

```
START
•
•
•
DI                          ;  Disable interrupt processing
LD      TBINT,#02H          ;  Disable timer B overflow interrupt
LD      IMR,#8AH            ;  Select levels IRQ1, IRQ3, and IRQ7
LD      IPR,#82H            ;  Priorities are IRQ7 > IRQ1 > IRQ3
LD      TADATA,#0FH         ;
LD      T0CON,#86H          ;  Enable timer A overflow interrupt
LD      TCH,#0H             ;
LD      TCL,#0H             ;
LD      T1CON,#35H          ;  Select normal baud rate, enable timer C interrupt
LD      T1MOD,#31H          ;  Disable timer D, set timer C to 16-bit timer mode
LD      P4CONH,#55H         ;  Select input mode with rsing edge interrupts
LD      P4PND,#0FFH         ;  Reset all port 4 interrupt pending values
LD      P4INT,#0F0H         ;  Enable external interrupts at P4.4–P4.7
                            ;  (Other interrupts are disabled automatically by a reset)
•
•
•
EI                          ;  Enable interrupts
```

Assuming interrupt sources and priorities have been set by the above instruction sequence, it would be possible to select interrupt level 1, 3, or 7 for fast interrupt processing. The following instructions enable fast interrupts for level 7 (IRQ7):

```
DI                          ;  Disable interrupts
LDW     IPH,#3000H          ;  Load the service routine address for IRQ7
LD      SYM,#1EH            ;  Enable fast interrupt processing
EI                          ;  Enable interrupts
```

note

6 SAM8 Instruction Set

OVERVIEW

The SAM8 instruction set is designed to support the large register file of KS88-series microcontrollers. It includes a full complement of 8-bit arithmetic and logic operations, including multiply and divide.

There are 79 instructions. No special I/O instructions are necessary because I/O control and data registers are mapped directly into the register file. Decimal adjustment is 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 complete the powerful data manipulation capabilities of the SAM8 instruction set.

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 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.4 – FLAGS.7, can be tested and used with conditional jump instructions; two others FLAGS.2 and FLAGS.3 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 being addressed.

FLAGS is located in the system control register area of set 1 (D5H). FLAGS bits can be read or written directly by program instructions.

However, the FLAGS register cannot be specified as the destination of a specific instruction that normally affects the flag bits. If this occurs, the result of the operation will be unspecified.

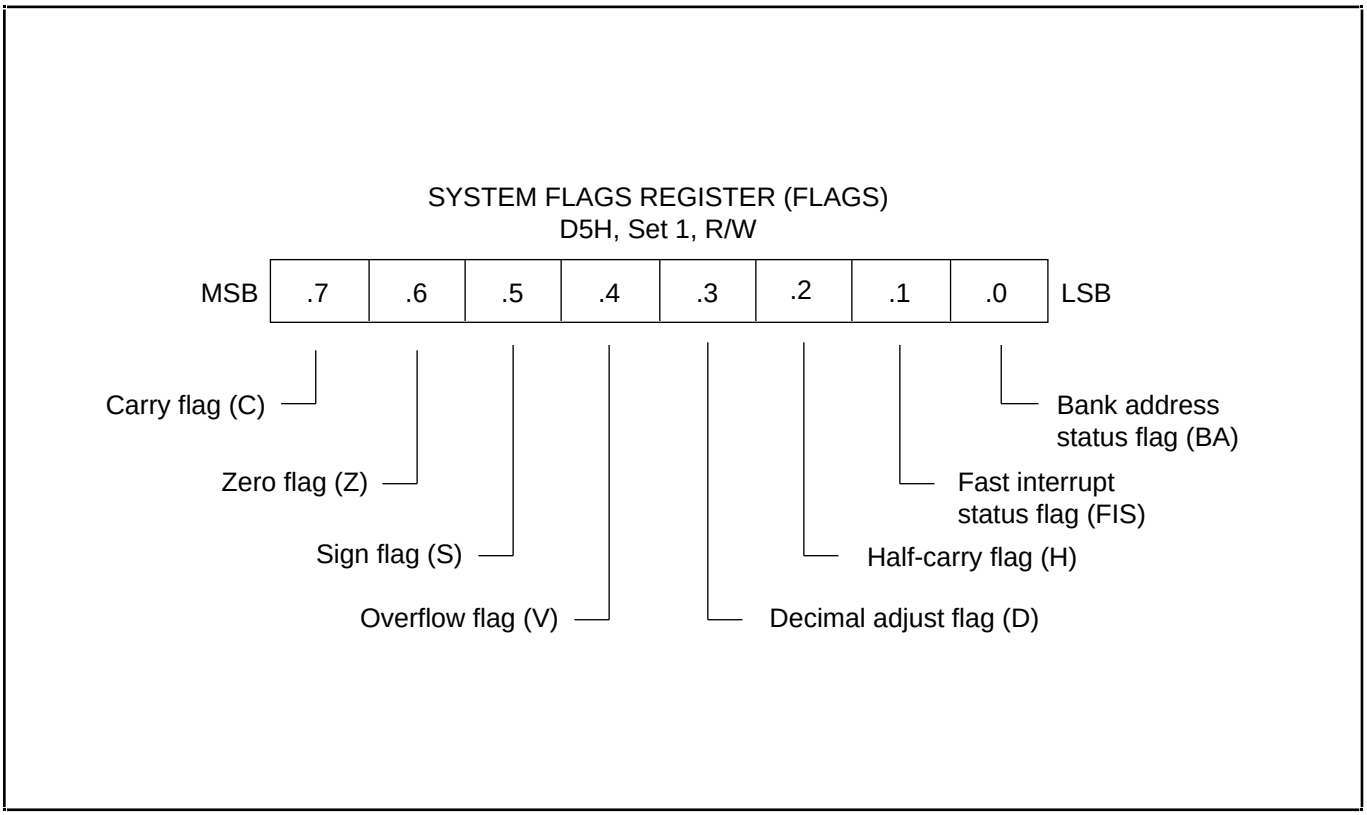


Figure 6–1. System Flags Register (FLAGS)

Flag Descriptions

Bank Address Flag (FLAGS.0, BA)

The BA flag indicates which register bank is 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.

Fast Interrupt Status Flag (FLAGS.1, FIS)

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.

Half-Carry Flag (FLAGS.2, H)

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.

Decimal Adjust Flag (FLAGS.3, D)

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.

Overflow Flag (FLAGS.4, V)

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.

Sign Flag (FLAGS.5, S)

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.

Zero Flag (FLAGS.6, Z)

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.

Carry Flag (FLAGS.7, C)

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.

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 *	C	Carry	C = 1
1111 *	NC	No carry	C = 0
0110 *	Z	Zero	Z = 1
1110 *	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 *	EQ	Equal	Z = 1
1110 *	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 *	UGE	Unsigned greater than or equal	C = 0
0111 *	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. Asterisks (*) indicate 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 (Hex)	Addr Mode dst	src
opc | dst | src	2	6	12	r	r
			13	r	lr
opc | src | dst	3	10	14	R	R
			15	R	IR
opc | dst | src	3	10	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 (Hex)	Addr Mode dst	Addr Mode src
opc | dst | src	2	6	02	r	r
			03	r	lr
opc | src | dst	3	10	04	R	R
			05	R	IR
opc | dst | src	3	10	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 (Hex)	Addr Mode dst	Addr Mode src
opc	dst | src		2	6	52	r	r
					53	r	lr
opc	src	dst	3	10	54	R	R
					55	R	IR
opc	dst	src	3	10	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 (Hex)	Addr Mode dst	Addr Mode src
opc | dst | b | 0 | src	3	10	67	r0	Rb
opc | src | b | 1 | dst	3	10	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:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	dst | b | 0	src	3	10	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 (Hex)	Addr Mode dst
opc | dst | b | 0	2	8	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:

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc	dst | b | 0	2	8	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	8	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 (Hex)	Addr Mode dst	Addr Mode src
opc	dst | b | 0	src	3	10	07	r0	Rb
opc	src | b | 1	dst	3	10	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:

(Note 1)

opc	src | b | 0	dst

Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
3	16/18 (2)	37	RA	rb

NOTES:
1. 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.
2. Execution time is 18 cycles if the jump is taken or 16 cycles if it is not taken.

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:

(Note 1)			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	src | b | 1	dst	3	16/18 (2)	37	RA	rb

NOTES:
1. 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.
2. Execution time is 18 cycles if the jump is taken or 16 cycles if it is not taken.

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 (Hex)	Addr Mode dst	Addr Mode src
opc	dst | b | 0	src	3	10	27	r0	Rb
opc	src | b | 1	dst	3	10	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	18	F6	DA
opc | dst	2	18	F4	IRR
opc | dst	2	20	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	6	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 dst
opc | dst	2	6	B0	R
			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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc dst	2	6	60	R
			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 (Hex)	Addr Mode dst	src
opc | dst | src	2	6	A2	r	r
			A3	r	lr
opc | src | dst	3	10	A4	R	R
			A5	R	IR
opc | dst | src	3	10	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 (Hex)	Addr Mode dst	src
opc | src | dst | RA	3	16/18	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 (Hex)	Addr Mode dst	Addr Mode src
opc | src | dst | RA	3	16/18	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:

CPIJNE R1,@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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst	2	6	40	R
			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 (Hex)	Addr Mode dst
opc | dst	2	6	00	R
			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	10	80	RR
			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.

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	6	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.

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	28/12 *	94	RR	R
	28/12 *	95	RR	IR
	28/12 *	96	RR	IM

* Execution takes 12 cycles if the divide-by-zero is attempted; otherwise it takes 28 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.

In addition to, the working registers that you use to execute a DJNZ instruction must be located either in page 0 (00H–BFH) or in set 1 (C0H–CFH).

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
r | opc | dst	2	12 (jump taken)	rA	RA
		10 (no jump)	r = 0 to F	

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

```
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	6	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

Operation:

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

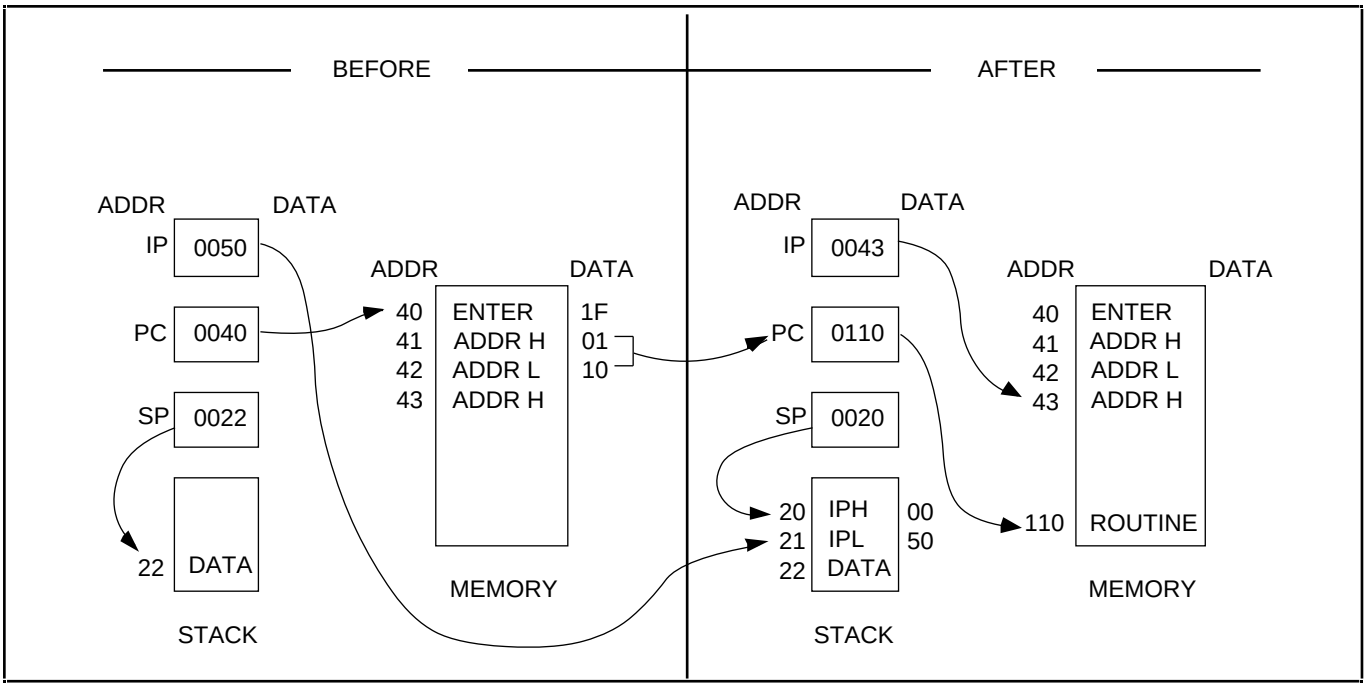
This instruction is useful when implementing threaded-code languages. The contents of the instruction pointer are pushed to the stack. The program counter (PC) value is then written to the instruction pointer. The program memory word that is pointed to by the instruction pointer is loaded into the PC,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

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	20	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

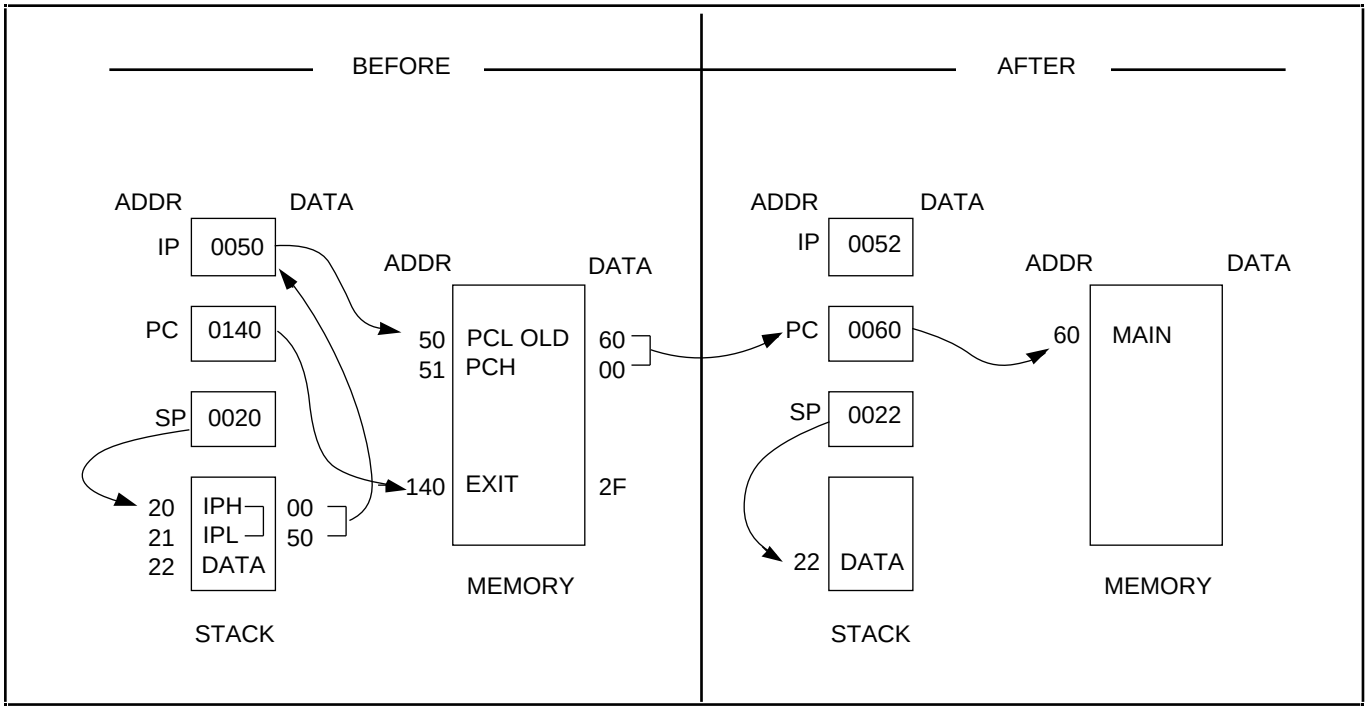
This instruction is useful when implementing threaded-code languages. The stack value is popped and loaded into the instruction pointer. The program memory word that is pointed to by the instruction pointer is then loaded into the program counter, and the instruction pointer is incremented by two.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	22	2F

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



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	3	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 (Hex)	Addr Mode dst
dst | opc	1	6	rE r = 0 to F	r
opc | dst	2	6	20 21	R 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 (Hex)	Addr Mode dst
opc dst	2	10	A0	RR
			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.

IRET

Interrupt Return

IRET IRET (Normal) IRET (Fast)

Operation:

IRET (Normal)	IRET (Fast)
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.

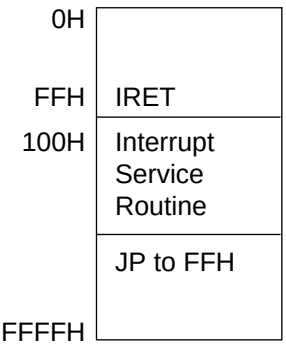
Flags: All flags are restored to their original settings (that is, the settings before the interrupt occurred).

Format:

IRET (Normal)	**Bytes**	**Cycles**	**Opcode (Hex)**
opc	1	16	BF

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 that 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 proceeded 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)

(2)		Bytes	Cycles	Opcode (Hex)	Addr Mode dst
cc | opc	dst	3	10/12 (3)	ccD cc = 0 to F	DA
opc	dst	2	10	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.
3. For a conditional jump, execution time is 12 cycles if the jump is taken or 10 cycles if it is not taken.

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

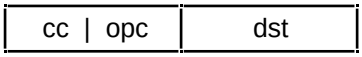
If the condition specified by the condition code (cc) is true, the relative address is added to the program counter and control passes to the statement whose address is now in the program counter; otherwise, the instruction following the JR instruction is executed. (See list of condition codes).

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)



Bytes	Cycles	Opcode (Hex)	Addr Mode dst
2	10/12 (2)	ccB	RA
		cc = 0 to F	

NOTES:
1. In the first byte of the two-byte instruction format, the condition code and the opcode are each four bits.
2. Instruction execution time is 12 cycles if the jump is taken or 10 cycles if it is not taken.

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 (Hex)	Addr Mode dst	src
dst | opc / src	2	6	rC	r	IM
		6	r8	r	R
src | opc / dst	2	6	r9 (r = 0 to F)	R	r
opc / dst | src	2	6	C7	r	lr
		6	D7	lr	r
opc / src / dst	3	10	E4	R	R
		10	E5	R	IR
opc / dst / src	3	10	E6	R	IM
		10	D6	IR	IM
opc / src / dst	3	10	F5	IR	R
opc / dst | src / x	3	10	87	r	x [r]
opc / src | dst / x	3	10	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 (Hex)	Addr Mode dst	Addr Mode src
opc | dst | b | 0 | src	3	10	47	r0	Rb
opc | src | b | 1 | dst	3	10	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 (Hex)	Addr Mode dst	Addr Mode src
1.	opc	dst | src			2	12	C3	r	Irr
2.	opc	src | dst			2	12	D3	Irr	r
3.	opc	dst | src	XS		3	18	E7	r	XS [rr]
4.	opc	src | dst	XS		3	18	F7	XS [rr]	r
5.	opc	dst | src	XL_L	XL_H	4	20	A7	r	XL [rr]
6.	opc	src | dst	XL_L	XL_H	4	20	B7	XL [rr]	r
7.	opc	dst | 0000	DA_L	DA_H	4	20	A7	r	DA
8.	opc	src | 0000	DA_L	DA_H	4	20	B7	DA	r
9.	opc	dst | 0001	DA_L	DA_H	4	20	A7	r	DA
10.	opc	src | 0001	DA_L	DA_H	4	20	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 (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 *   @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 *   #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 *   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
```

* 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 (Hex)	Addr Mode dst	src
opc | dst | src	2	16	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 (Hex)	Addr Mode dst	src
opc	dst | src	2	16	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 (Hex)	Addr Mode dst	src
opc	src | dst	2	16	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	16	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	Addr Mode src
opc | src | dst	3	10	C4	RR	RR
		10	C5	RR	IR
opc | dst | src	4	12	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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | src | dst	3	24	84	RR	R
		24	85	RR	IR
		24	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

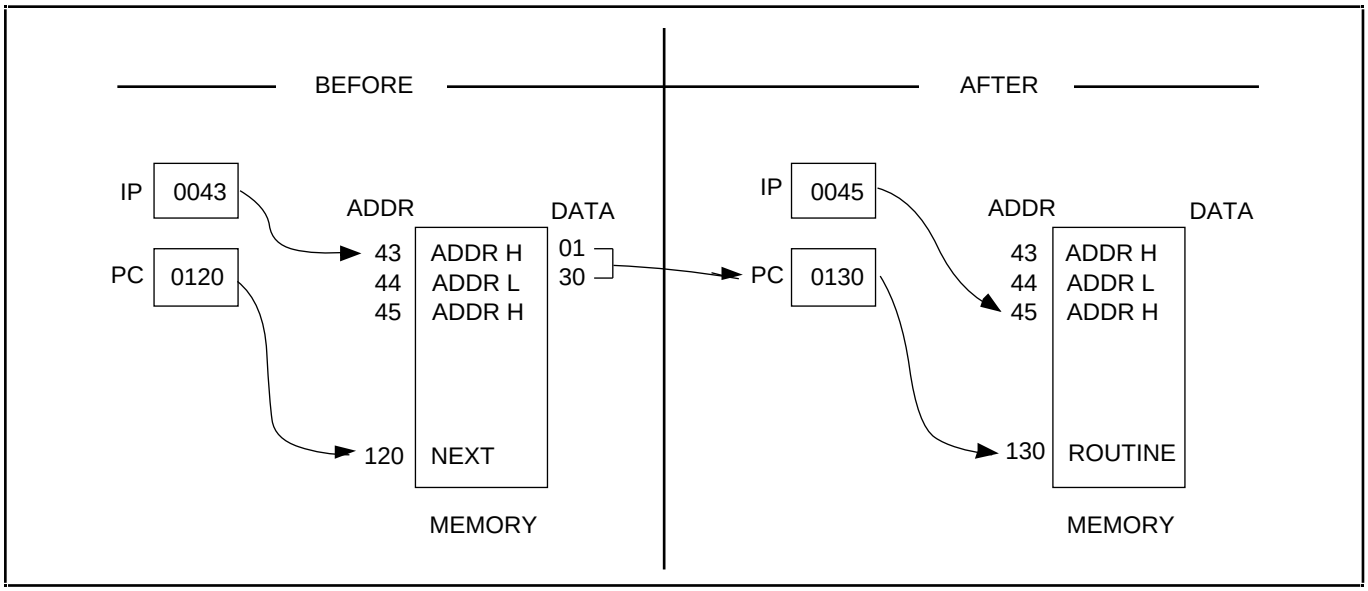
The NEXT instruction is useful when implementing threaded-code languages. The program memory word that is pointed to by the instruction pointer is loaded into the program counter. The instruction pointer is then incremented by two.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	14	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	6	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	Addr Mode src
opc	dst | src		2	6	42	r	r
				6	43	r	lr
opc	src	dst	3	10	44	R	R
				10	45	R	IR
opc	dst	src	3	10	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 dst
opc | dst	2	10	50	R
		10	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 (Hex)	Addr Mode dst	Addr Mode src
opc	src	dst	3	10	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 (Hex)	Addr Mode dst	Addr Mode src
opc | src | dst	3	10	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	10 (internal clock)	70	R
		12 (external clock)		
		12 (internal clock)		
		14 (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 (Hex)	Addr Mode dst	Addr Mode src
opc dst src	3	10	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 (Hex)	Addr Mode dst	src
opc	dst	src	3	10	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	6	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	14	AF

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

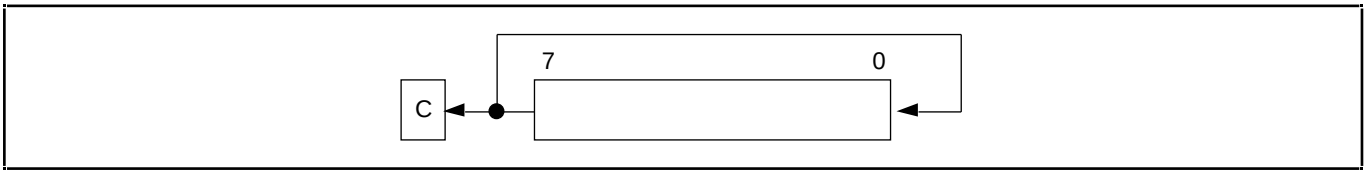
RL dst

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 dst
opc | dst	2	6	90	R
		6	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

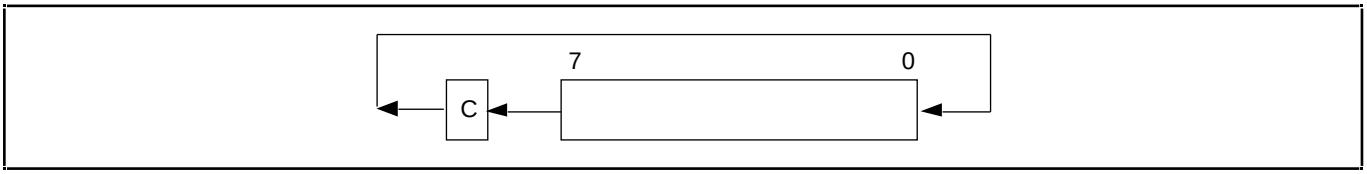
RLC dst

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	6	10	R
		6	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

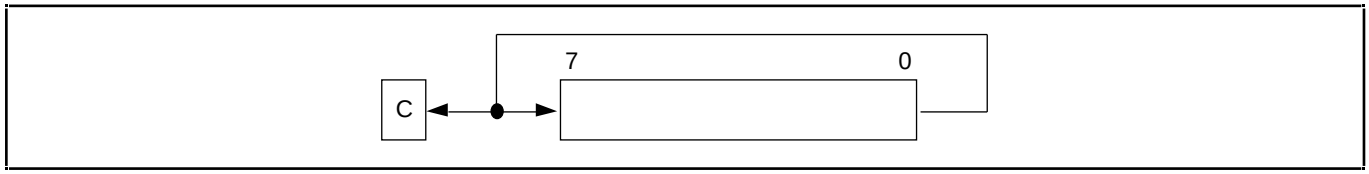
RR dst

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 dst
opc dst	2	6	E0	R
		6	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

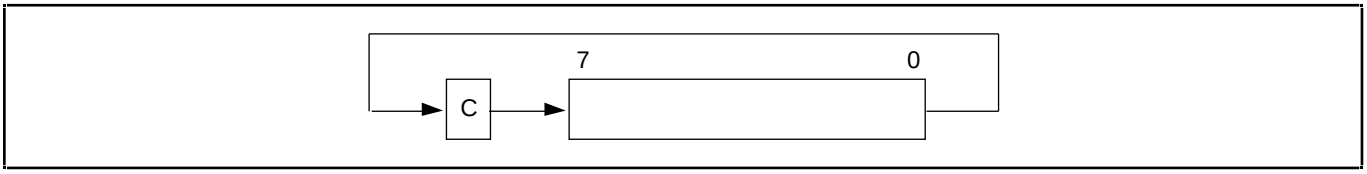
RRC dst

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	6	C0	R
		6	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	6	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 KS88-series microcontrollers.)

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	6	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	6	32	r	r
				6	33	r	lr
opc	src	dst	3	10	34	R	R
				10	35	R	IR
opc	dst	src	3	10	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 = 93V = "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	6	DF

Example: The statement

SCF

sets the carry flag to logic one.

SRA

Shift Right Arithmetic

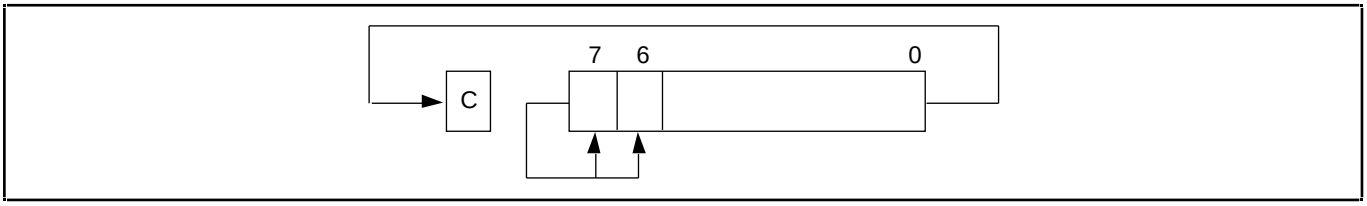
SRA dst

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	6	D0	R
		6	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 (Hex)	Addr Mode src
opc	src	2	6	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 only by an external reset operation. 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 (Hex)	Addr Mode dst	src
opc	1	3	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 (Hex)	Addr Mode dst	Addr Mode src
opc | dst | src	2	6	22	r	r
		6	23	r	lr
opc | src | dst	3	10	24	R	R
		10	25	R	IR
opc | dst | src	3	10	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.

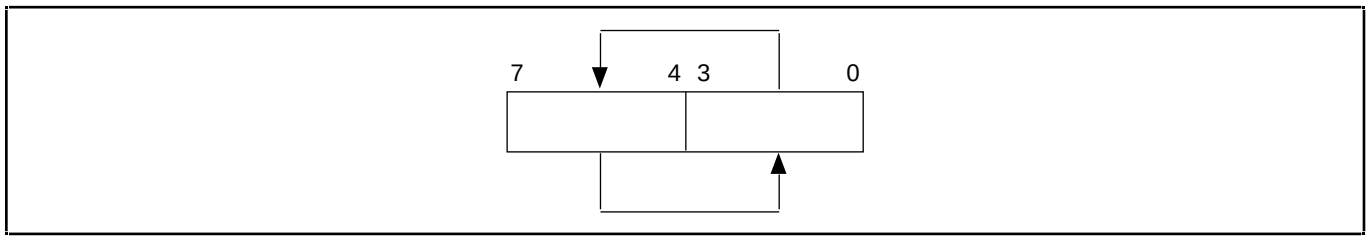
SWAP

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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst	2	8	F0	R
		8	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	Addr Mode src
opc | dst | src	2	6	62	r	r
		6	63	r	lr
opc | src | dst	3	10	64	R	R
		10	65	R	IR
opc | dst | src	3	10	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	6	72	r	r
		6	73	r	lr
opc | src | dst	3	10	74	R	R
		10	75	R	IR
opc | dst | src	3	10	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:

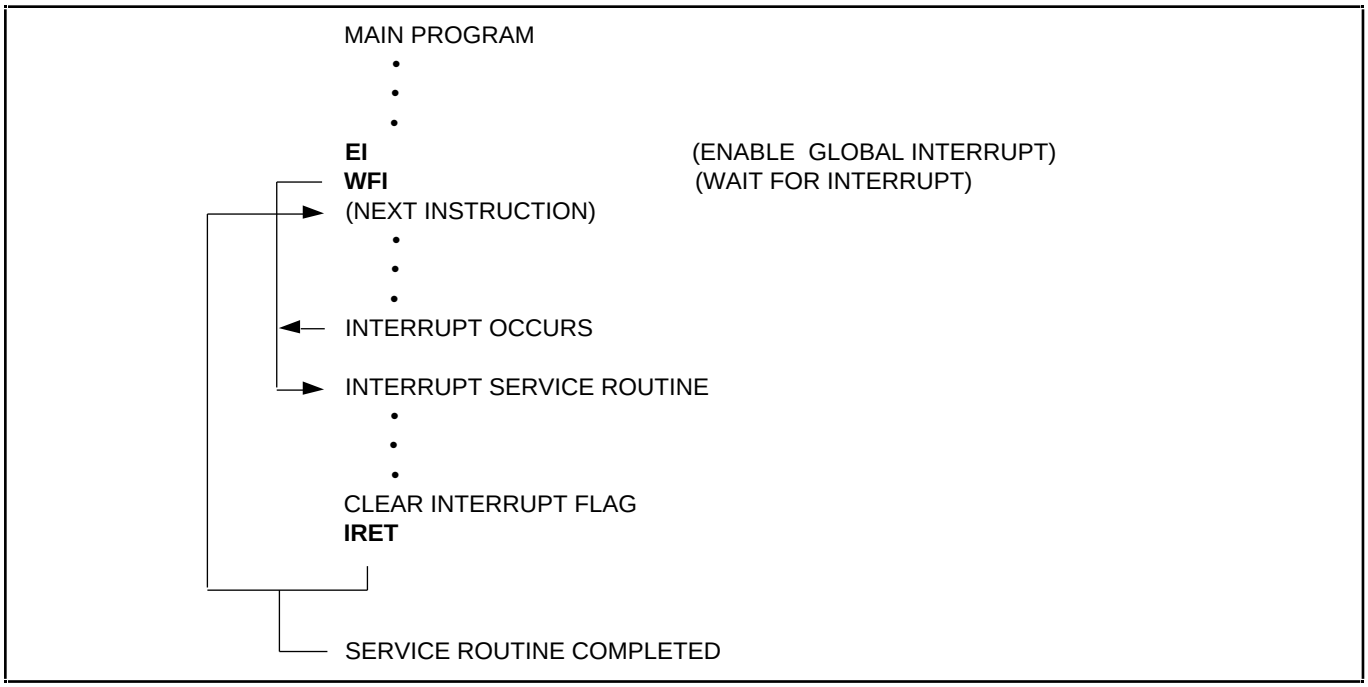
The CPU is effectively halted until an interrupt occurs, except that DMA transfers can still take place during this wait state. The WFI status can be released by an internal interrupt, including a fast interrupt .

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	6n	3F
		(n = 1, 2, 3, …)	

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	Addr Mode src
opc | dst | src	2	6	B2	r	r
		6	B3	r	lr
opc | src | dst	3	10	B4	R	R
		10	B5	R	IR
opc | dst | src	3	10	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

Oscillator Circuits

RESET **and Power-Down**

I/O Ports

Timer Module 0

Timer Module 1

Serial Port (UART)

PWM and Capture

A/D Converter

External Interface

Electrical Data

Mechanical Data

Development Tools

7 Oscillator Circuits

OVERVIEW

The KS88C4400 microcontroller's single external crystal oscillation source provides a maximum 18 MHz clock for the CPU. The X_{IN} and X_{OUT} pins connect the external oscillation source to the on-chip clock circuit.

Although you can use an external ceramic resonator as an oscillation source for the CPU clock, it is not recommended because the highest possible clock resolution is required for optimal performance.

MAIN OSCILLATOR CIRCUIT

The main oscillator circuit generates the CPU clock signal. To increase processing speed and to reduce noise levels, non-divided logic has been implemented for the main clock circuit. Non-divided clock operation is fast, but it requires a very high resolution waveform (square signal edges) in order for the CPU to efficiently process logic operations.

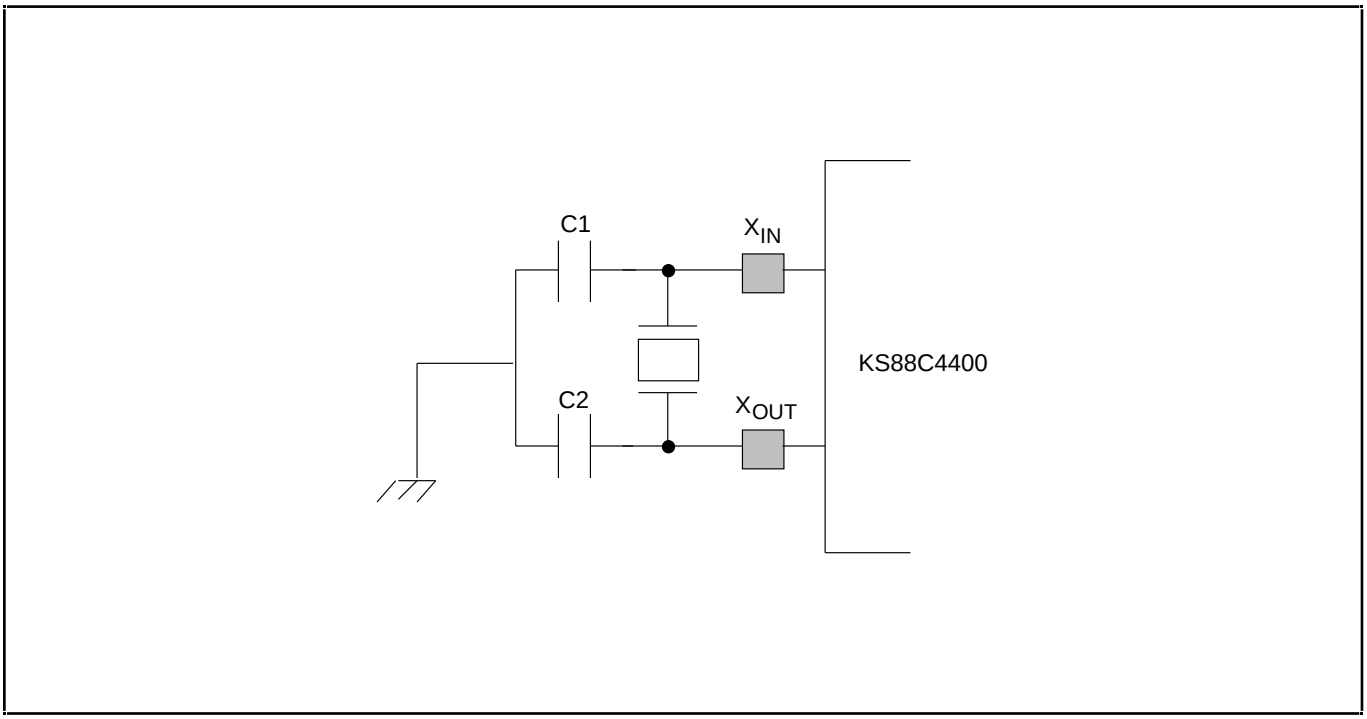


Figure 7–1. Main Oscillator Circuit (With External Crystal or Resonator)

Clock Status During Power-Down Modes

The two power-down modes, Stop mode and Idle mode, affect system clock oscillation as follows: During Stop mode, the main oscillator is stopped, but contents of the internal register file and special function registers are retained. Stop mode is ended, and clock oscillation re-started, by a reset operation.

In Idle mode, the internal clock signal is gated to the CPU, but not to the interrupts, timers, and serial port functions. CPU status is preserved, including stack pointer, program counter, and flags, and data contained in the internal register file is retained. Idle mode can be terminated either by an interrupt or by RESET.

8 RESET and Power-Down

RESET

A reset overrides all other operating conditions and puts the KS88C4400 into a known state. A reset is initiated by holding the signal at the RESET pin to low level for at least 22 CPU clocks. The RESET signal is input through a Schmitt trigger circuit and is then synchronized with the CPU clock. The following events occur during a reset operation:

— All interrupts are disabled.

— Ports 2–5 are set to input mode.

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

— The program counter is loaded with the ROM's reset address, 0020H.

— Eight clocks after RESET returns high, the instruction is fetched from ROM location 0020H and executed.

For power-up, the RESET input must be held low for about 20 milliseconds after the power supply comes within tolerance. This allows enough time for internal CPU clock oscillation to stabilize.

KS88C4400 Reset Operation (ROM-Less Mode)

The KS88C4400 microcontroller can be configured as a ROM-less device by applying a constant 5 V current to the EA pin. Assuming the EA pin is held at high level (5 V input) prior to a RESET, the reset operation initiates ROM-less operating mode and the external interface is automatically configured by firmware, as follows:

— Port A, AD and C pins are automatically configured for external interface operation.

Figure 8–1 shows the timing of the address and data strobes when a RESET occurs during ROM-less operation.

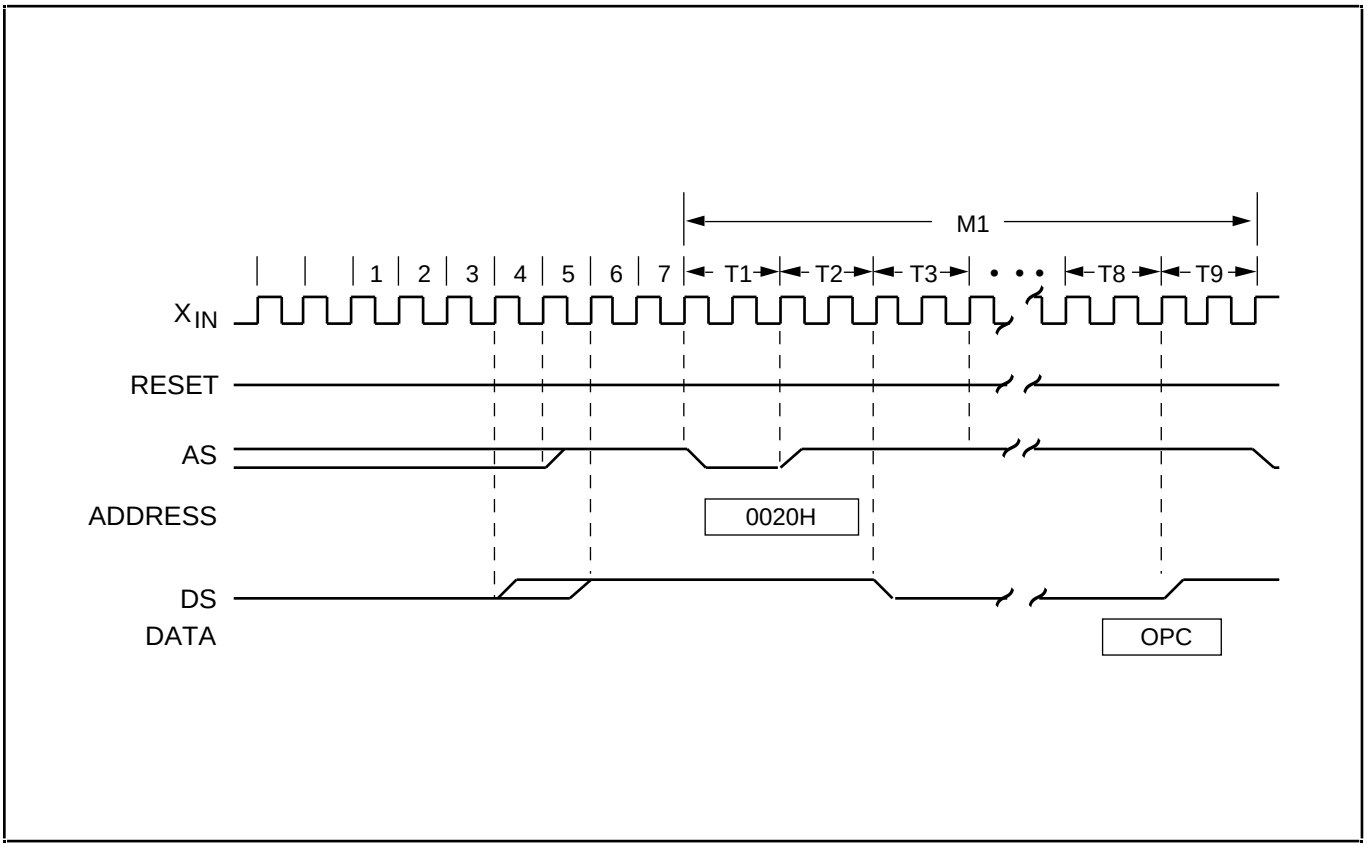


Figure 8–1. RESET **Timing for External Interface (ROM-less Mode)**

Register Initialization Values Following RESET

Table 8–1 lists the values for KS88C4400 CPU and system registers, peripheral control registers, and peripheral data registers following a reset operation in normal operating mode.

Table 8–2 shows the hardware reset values for external interface control registers when the KS88C4400 is configured for ROM-less operation (that is, when 5 V is applied at the EA pin). The following notation is used to represent specific reset values:

— A "1" or a "0" shows the reset bit value as logic one or logic zero.

— 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.

Table 8–1. Set 1 Register Values After RESET

Register Name	Mnemonic	Address		Bit Values After RESET							
		Dec	Hex	7	6	5	4	3	2	1	0
Timer C Data Register (High Byte)	TCH	R208	D0H	x	x	x	x	x	x	x	x
Timer C Data Register (Low Byte)	TCL	R209	D1H	x	x	x	x	x	x	x	x
Timer D Data Register (High Byte)	TDH	R210	D2H	x	x	x	x	x	x	x	x
Timer D Data Register (Low Byte)	TDL	R211	D3H	x	x	x	x	x	x	x	x
Port 4 Interrupt Pending Register	P4PND	R212	D4H	0	0	0	0	0	0	0	0
System Flags Register	FLAGS	R213	D5H	x	x	x	x	x	x	0	0
Register Pointer 0	RP0	R214	D6H	1	1	0	0	0	0	0	0
Register Pointer 1	RP1	R215	D7H	1	1	0	0	1	0	0	0
Stack Pointer (High Byte)	SPH	R216	D8H	x	x	x	x	x	x	x	x
Stack Pointer (Low Byte)	SPL	R217	D9H	x	x	x	x	x	x	x	x
Instruction Pointer (High Byte)	IPH	R218	DAH	x	x	x	x	x	x	x	x
Instruction Pointer (Low Byte)	IPL	R219	DBH	x	x	x	x	x	x	x	x
Interrupt Request Register	IRQ	R220	DCH	0	0	0	0	0	0	0	0
Interrupt Mask Register	IMR	R221	DDH	x	x	x	x	x	x	x	x
System Mode Register	SYM	R222	DEH	0	–	–	x	x	x	0	0
Register Page Pointer	PP	R223	DFH	–	–	–	–	–	–	0	0

Table 8–2. Set 1, Bank 0 Register Values After RESET

Register Name	Mnemonic	Address		Bit Values After RESET							
		Dec	Hex	7	6	5	4	3	2	1	0
Port 2 Data Register	P2	R226	E2H	0	0	–	–	–	–	–	–
Port 3 Data Register	P3	R227	E3H	0	0	0	0	0	0	0	0
Port 4 Data Register	P4	R228	E4H	0	0	0	0	0	0	0	0
Port 5 Data Register	P5	R229	E5H	0	0	0	0	0	0	0	0
Port 6 Data Register	P6	R230	E6H	1	1	1	1	1	1	1	1
Serial I/O Shift Register	SIO	R233	E9H	x	x	x	x	x	x	x	x
Serial I/O Control Register	SIOCON	R234	EAH	0	0	0	0	0	0	0	0
Serial I/O Interrupt Pending Reigister	SIOPND	R235	EBH	–	–	–	–	–	–	0	0
Timer A Data Register	TADATA	R236	ECH	0	0	0	0	0	0	0	0
Timer B Data Register	TBDATA	R237	EDH	0	0	0	0	0	0	0	0
Timer Module 0 Control Register	T0CON	R238	EEH	0	0	0	0	0	0	0	0
Timrt B Control Register	TBCON	R239	EFH	–	–	–	–	–	0	0	0
Port 2 Control Register	P2CON	R242	F2H	0	0	0	0	–	–	–	–
Port 3 Control Register (High Byte)	P3CONH	R244	F4H	0	0	0	0	0	0	0	0
Port 2 Control Register (Low Byte)	P3CONL	R245	F5H	0	0	0	0	0	0	0	0
Port 4 Control Register (High Byte)	P4CONH	R246	F6H	0	0	0	0	0	0	0	0
Port 4 Control Register (Low Byte)	P4CONL	R247	F7H	0	0	0	0	0	0	0	0
Port 5 Control Register	P5CON	R248	F8H	0	0	0	0	0	0	0	0
Port 4 Interrupt Enable Register	P4INT	R249	F9H	0	0	0	0	0	0	0	0
Timer Module 1 Control Register	T1CON	R250	FAH	0	0	0	0	0	0	0	0
Timer Module 1 Mode Register	T1MOD	R251	FBH	0	0	0	0	0	0	0	0
Port 3 Interrupt Enable Register	P3INT	R252	FCH	0	0	0	0	0	0	0	0
Port 3 Interrupt Pending Register	P3PND	R253	FDH	0	0	0	0	0	0	0	0
External Memory Timing Register	EMT	R254	FEH	0	1	1	1	1	1	0	–
Interrupt Priority Register	IPR	R255	FFH	x	x	x	x	x	x	x	x

Table 8–3. Set 1, Bank 1 Register Values After RESET

Register Name	Mnemonic	Address		Bit Values After RESET							
		Dec	Hex	7	6	5	4	3	2	1	0
A/D Converter Input Register	ADIN	R249	F9H	0	0	0	0	0	0	0	0
A/D Converter Output Register	ADOUT	R250	FAH	x	x	x	x	x	x	x	x
A/D Converter Control Register	ADCON	R251	FBH	0	0	0	0	1	–	–	–
PWM Module Control Register	PWMCON	R252	FCH	0	0	0	0	0	0	0	0
PWM1 Data Register	PWM1	R253	FDH	0	0	0	0	0	0	0	0
PWM0 Data Register	PWM0	R254	FEH	0	0	0	0	0	0	0	0
PWM Capture Register	PWMCAP	R255	FFH	0	0	0	0	0	0	0	0

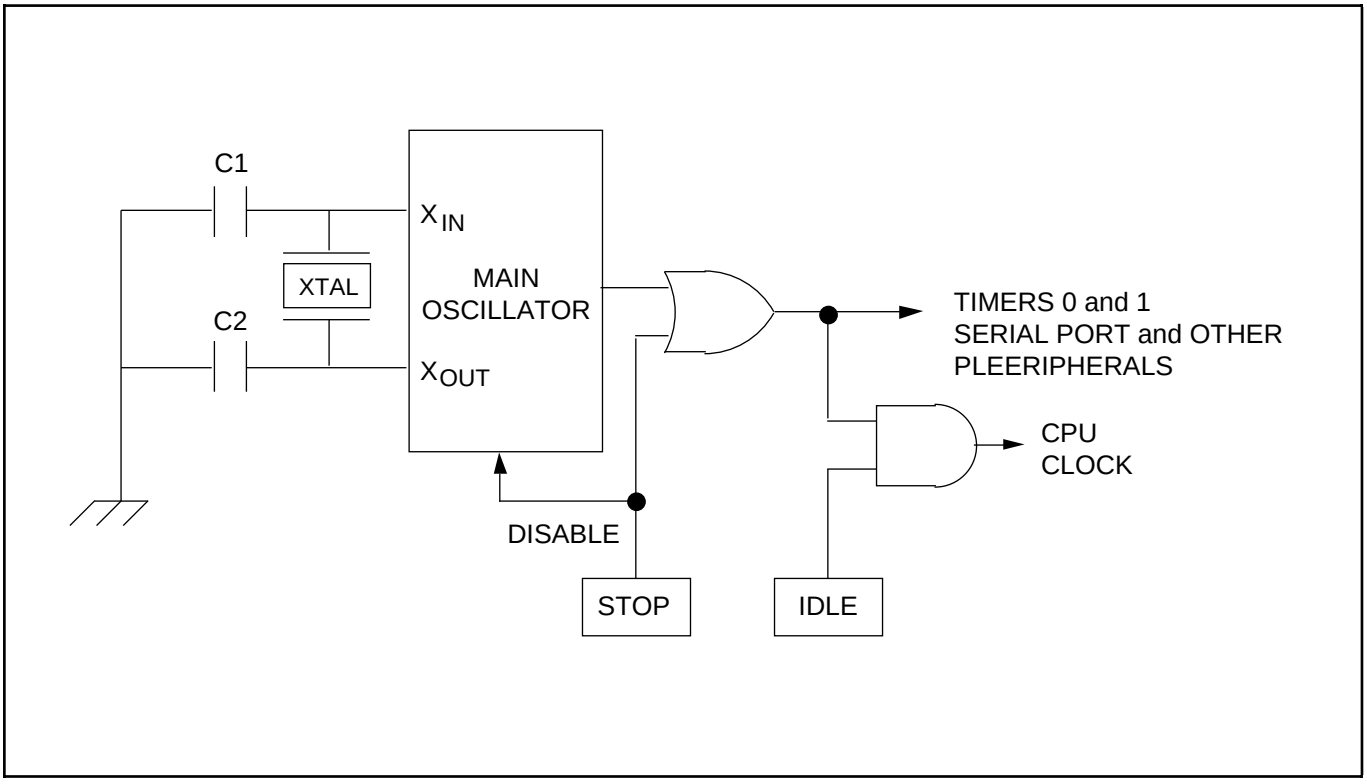


Figure 8–2. Stop and Idle Mode Function Diagram

Idle Mode

The instruction IDLE (opcode 6FH) invokes Idle mode. In Idle mode, the CPU "sleeps" while select peripherals remain active: For the KS88C4400, these peripherals are the PWM module, serial I/O port, timers A and B, timers C and D and external interrupt logic.

The CPU does this by holding the internal clock signal (CLK) high while toggling the Idle mode clock (CLKID) high and low. During Idle mode, the contents of system registers, control registers, and data registers are retained. Port pins retain the mode (input or output) they had at the time Idle mode was entered.

There are two ways to end Idle mode once it has been started:

1. Activate any enabled interrupt, causing a hardware release of Idle mode. The interrupt will then be serviced. Following the IRET from the service routine, the instruction immediately following the one which originally initiated Idle mode will be executed.

   If you use an interrupt request, the interrupt settings in the IMR register and the interrupt enable/disable bit value should be correct so that the CPU can respond correctly to the interrupt and "wake up." An enabled externally-generated interrupt, the timer B non-maskable interrupt, or a fast interrupt (for valid interrupt levels only) may also be used to end Idle mode.
2. Execute an external reset. Since the clock oscillator is still running, the reset operation is completed by hardware. If interrupts are masked, a hardware reset is the only way to end Idle mode.

STOP MODE

The instruction STOP (opcode 7FH) invokes Stop mode. In Stop mode, both the CPU and its peripherals are "put to sleep": the on-chip main oscillator is stopped and both the internal clock signal (CLK) and the power-down clock signal (CLKID) are held high.

The supply current is reduced to less than 50 µA. When the clock freezes, all system functions stop; data stored in the internal register file and in peripheral control and data registers are retained.

The only way to exit Stop mode is by executing a hardware reset. RESET must be held low for at least 22 clock cycles. The reset operation resets all system and peripheral registers to their default values; data in the register file remains unchanged. When RESET is released and goes high, the processor restarts the program from ROM vector address 0020H.

☞ Programming Tip — Sample KS88C4400 Initialization Routine

The following sample program is an example of how to make the initial program settings for the KS88C4400 address space, interrupt vectors, and peripheral functions. The program comments guide you through the necessary steps:

```
;       << Base Number Setting >>

            DECIMAL

;       << User Equation Define >>

            INCLUDE     C: EQU.TBL

;       Reference label vector area: 000H–00FFH

            ORG         0000H

;       << Reset Vector >>

            ORG         0020H
            JP          t,INITIAL

;           0023H–00B7H: Reserved

;       << Interrupt Vector Addresses >>

            ORG         00B8H

VECTOR      PWMOV_int                       ;   IRQ1
VECTOR      PWMCAP_int                      ;   IRQ1
  ;         00BCH–00BDH: Reserved
            ORG         00BEH
VECTOR      TA_int                          ;   IRQ1

  ;         00C0H–00D7H: Reserved

            ORG         00D8H

VECTOR      EXT40_int                       ;   IRQ5
VECTOR      EXT41_int                       ;   IRQ6
VECTOR      EXT42_int                       ;   IRQ6
VECTOR      EXT43_int                       ;   IRQ6
VECTOR      EXT44_int                       ;   IRQ7
VECTOR      EXT45_int                       ;   IRQ7
VECTOR      EXT46_int                       ;   IRQ7
VECTOR      EXT47_int                       ;   IRQ7
```

(Continued on next page)

☞ Programming Tip — Sample KS88C4400 Initialization Routine (Continued)

```
                ORG         00E8H

VECTOR          EXT30_int                       ;  IRQ4
VECTOR          EXT31_int                       ;  IRQ4
VECTOR          EXT32_33_int                    ;  IRQ4

                ORG         00F0H

VECTOR          SIOINT_R                        ;  IRQ3
VECTOR          SIOINT_T                        ;  IRQ3
VECTOR          TC_int                          ;  IRQ3
VECTOR          TD_int                          ;  IRQ3

;               00F8H–00FDH: Reserved

                ORG         00FEH

                VECTOR      TB_int              ;  IRQ0
```

; << System and Peripheral Initialization >>

```
                ORG         0100H

INITIAL:        DI
                LD          PP,#00              ;  Clear page pointer register
                SB0                             ;  Select bank 0

    ;       < System register setting >

                LD          SYM,#00000000B      ;  Fast, global interrupt disable
                LD          EMT,#00000000B      ;  No wait / internal stack area select
                LD          SPL,#00H            ;  Stack pointer low byte to zero
```

(Continued on next page)

☞ **Programming Tip — Sample KS88C4400 Initialization Routine (Continued)**

```
;       < Port 2 setting >

        LD      P2CON,#01100000B    ;  P2.6 output mode
                                    ;  P2.7 input mode

;       < Port 3 setting >

        LD      P3CONL,#10101010B   ;  P3.0–P3.7 input mode with pull-up
        LD      P3CONH,#10101010B
        LD      P3INT,#00H          ;  Disable all port 3 interrupts

;       < Port 4 setting >

        LD      P4CONL,#00000000B   ;  P4.0–P4.3 input mode
        LD      P4CONH,#11111111B   ;  P4.4–P4.7 output, push-pull
        LD      P4INT,#00H          ;  All P4 interrupts disabled

;       < Port 5 setting >

        LD      P5CON,#11H          ;  Output, push-pull

;       < Port 6 is always n-channel, open-drain, output mode >

;       < Timer A >

        LD      TADATA,#17H         ;  CPU clock divided by 18

        LD      T0CON,#00000110B    ;  If CPU clock = 11.0592 MHz
                                    ;  CPU clock / 1024 /18  →  1.66 ms
                                    ;  Interval mode
                                    ;  Interrupt enable

;       < Timer B >

        LD      TBINT,#02H          ;  Disable timer B interrupt

;       < Timer C >                 ;  16-bit free-running timer, no interrupt
```

(Continued on next page)

☞ Programming Tip — Sample KS88C4400 Initialization Routine (Continued)

```
;       < Timer D >                             ;  SIO baud rate generator
                                                ;  9600 BPS if CPU clock = 11.0592 MHz
                LD      TDL,#0FAH               ;  Start value
                LD      TDH,#0FAH               ;  Auto-reload value
                                                ;  FA = 9600 BPS
                                                ;  F4 = 4800 BPS

                LD      T1CON,#00110011B        ;  Normal baud rate
                                                ;  Timer C, D pending bit clear
                                                ;  Timer C, D interrupt disable
                                                ;  Timer C, D run enable

                LD      T1MOD,#00100000B        ;  Timer C, D gate function disable
                                                ;  Timer C and D clock is CPU clock /6
                                                ;  Timer C  →  16-bit timer mode
                                                ;  Timer D  →  Auto-reload mode

;       < SIO(UART) >                           ;  Mode 1, 8-bit UART, variable baud rate
                LD      SIOCON,#01010010B       ;  Multi-bit clear
                                                ;  Receive enable
                                                ;  Tx 9th bit = "0"
                                                ;  Rx 9th bit = "0"
                                                ;  Receive interrupt enable
                                                ;  Transmit interrupt disable

                LD      SIOPND,#03              ;  Pending bit clear

;       << Register Initialization >>

                SRP     #0C0H

;       < Clear all data registers 00H–0FFH in page 0 >

                LD      R0,#0FFH
RAMCLR:         CLR     @R0
                DJNZ    R0,RAMCLR

;       < Initialize other registers >

                •
                •
                •
                EI                              ;  Must be executed in this position
                                                ;  before external interrupt is executed
```

(Continued on next page)

☞ **Programming Tip — Sample KS88C4400 Initialization Routine (Continued)**

```
;       << Main Loop >>

MAIN:   NOP                             ;  Start main loop
        •
        •
        •
        CALL    KEY_SCAN
        •
        •
        •
        CALL    LED_DISPLAY
        •
        •
        •
        CALL    JOB
        •
        •
        •
        JR      t,MAIN


;       < Subroutine 1 >

KEY_SCAN: NOP
        •
        •
        •
        RET

;       < Subroutine 2 >

LED_DISPLAY: NOP
        •
        •
        •
        RET

;       < Subroutine 3 >

JOB:    NOP
        •
        •
        •
        RET
```

(Continued on next page)

☞ **Programming Tip — Sample KS88C4400 Initialization Routine (Continued)**

```
;       << Interrupt Service Routine >>

TA_int:     PUSH    RP0
            PUSH    RP1
            SRP     #TA_REG             ;  TA_REG = 30H
            •
            •
            •
            LD      T0CON,#00000110B    ;  Pending bit clear

            POP     RP1
            POP     RP0
            IRET

;       << Other Interrupt Vectors >>

EXT30_int:
EXT31_int:
EXT32_33_int:

            LD      P3PND,#0FH
            IRET

EXT40_int:
EXT41_int:
EXT42_int:
EXT43_int:
EXT44_int:
EXT45_int:
EXT46_int:
EXT47_int:

            LD      P4PND,#0FFH
            IRET

SIOint_R:
SIOint_T:

            LD      SIOPND,#03H
            IRET

TC_int:
TD_int:

            LD      T1CON,#00110011B
            IRET
```

(Continued on next page)

☞ **Programming Tip — Sample KS88C4400 Initialization Routine (Continued)**

```
TB_int:

        LD      TBINT,#02H
        IRET

        END
```

note

9 I/O Ports

OVERVIEW

Of the 80 pins in the KS88C4400's QFP package, 42 pins are used for I/O. There are eight ports:

— Three 8-bit I/O ports (port 3 through port 5)
— One 8-bit open-drain output port (port 6)
— One 8-bit input port (ADC0–ADC7, P7.0–7.7)
— One 2-bit I/O port (P2.6, P2.7)

Each port can be easily configured by software to meet various system configuration and design requirements. The CPU accesses I/O ports by directly writing or reading port register addresses. For this reason, special I/O instructions are not needed. The 8-bit input port can be used as analog inputs for the A/D converter module, or as general input port 7.

Table 9–1. KS88C4400 Port Configuration Overview

Port	Configuration Options
A	Address high port; configured as external address lines A8–A15 for the external interface.
AD	Address low/ Data pot; configured as multiplexed address/ data lines AD0–AD7 for the external interface.
C	External interface control port; used as bus signal control lines for the external interface: PM, DM, AS, DS, DR, DW
2	General I/O port; P2.6 and P2.7 can be configured as timer module 0 (timer A and timer B)outputs.
3	General I/O port; lower nibble pins (P3.0–P3.3) can be used alternately as inputs for timer module 1 or as external interrupt inputs INT0–INT3; the upper nibble pins P3.4 and P3.5 are for general I/O only; P3.6 can be configured as a capture input for the PWM module; P3.7 can be used for external device WAIT signal input.
4	General I/O port; can alternately serve as external interrupt inputs INT4–INT11.
5	General I/O port
6	N-channel, open-drain output with high-current capability
7	Analog input channels ADC0–ADC7; alternately, geneal input port

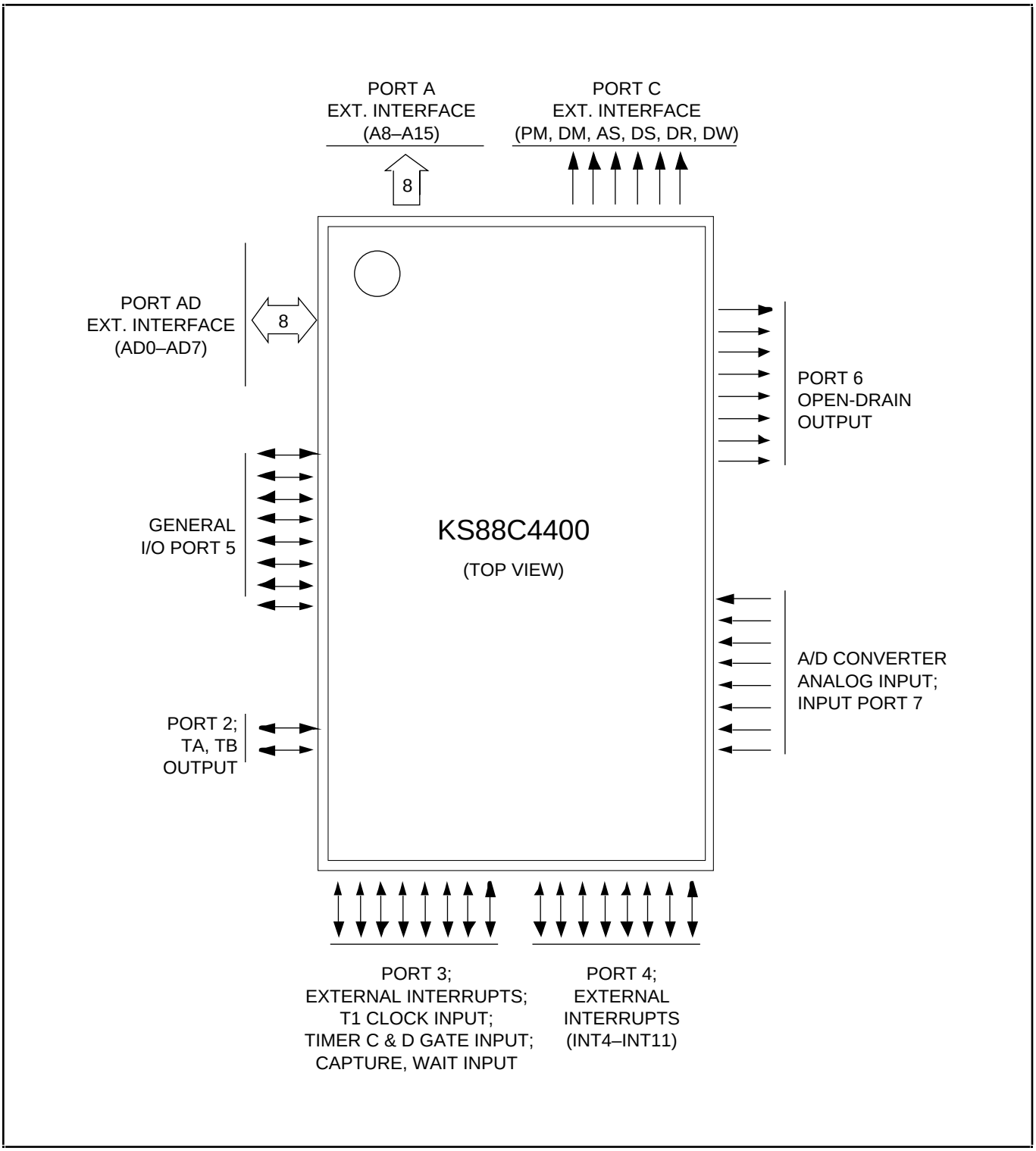


Figure 9–1. KS88C4400 Port Pin Arrangement

Port Data Registers

All nine port data registers have the identical structure shown in Figure 9–2 below. The following section of the KS88C4400 Register Map (Table 4–1) gives you a summary of the data register locations:

Table 9–2. Port Data Register Summary

Register Name	Mnemonic	Decimal	Hex	R/W
Port 2 Data Register	Port 2	R226	E2H	R/W
Port 3 Data Register	Port 3	R227	E3H	R/W
Port 4 Data Register	Port 4	R228	E4H	R/W
Port 5 Data Register	Port 5	R229	E5H	R/W
Port 6 Data Register	Port 6	R230	E6H	R/W
Port 7 Data Register	Port 7	R231	E7H	R/W

NOTE: All KS88C4400 port data registers are located in set 1, bank 0.

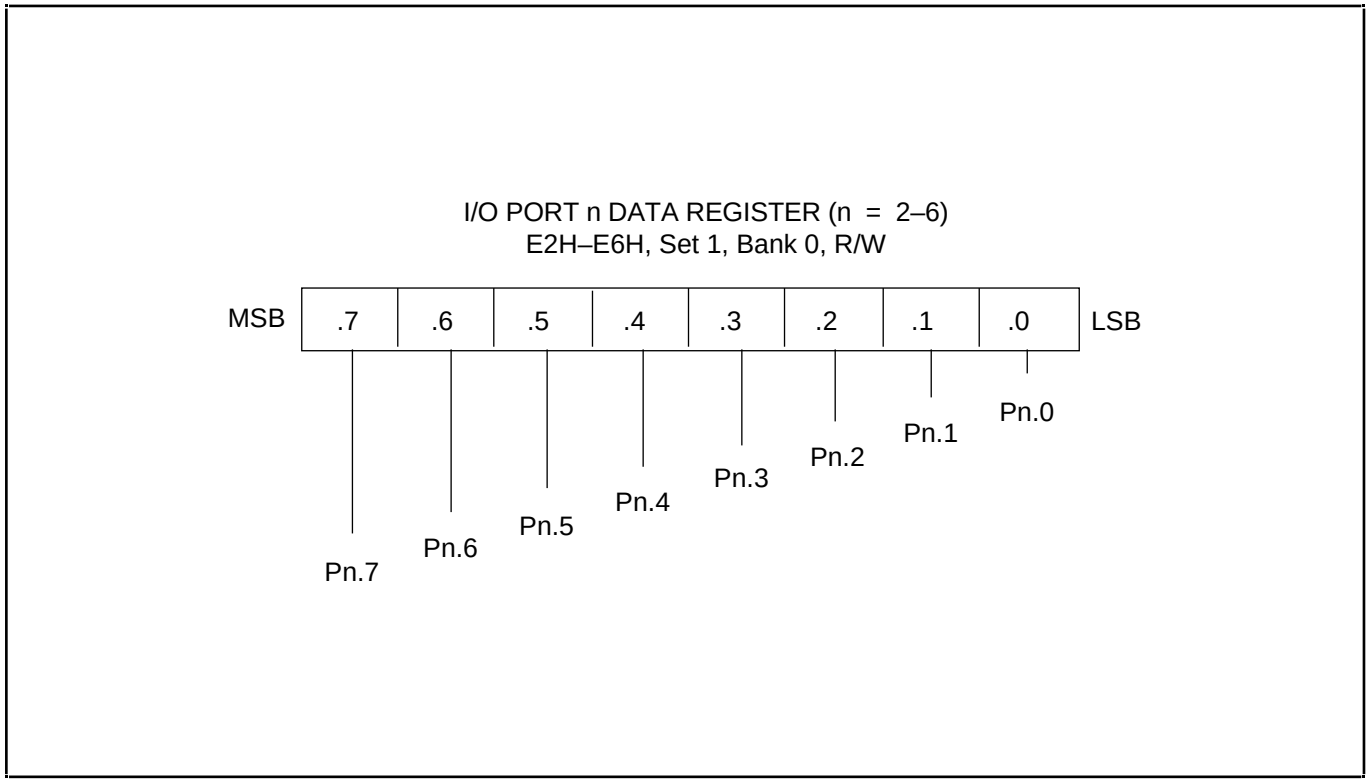


Figure 9–2. Port Data Register Structure

Port a

Port A pins can be configured as address lines (A8–A15) for the external peripheral interface. In ROM-less operating mode (when the state of the EA pin is high), a reset operation automatically configures PORT A as address lines A8–A15 of the external peripheral interface.

PORT AD

Port AD is used the multiplexed address/ data lines (AD0–AD7) for the external interface. In ROM-less operating mode (when the state of the EA pin is high), a reset operation automatically configures port AD as address/ data lines AD0–AD7 of the external peripheral interface.

PORT C

Port C is an 6-bit external interface control output port pins. Pot C is used for the following functions:

— DM (data memory), and PM (program memory) signals

— DS (data strobe), and AS (address strobe) signals

— DR (data memorry read), and DW (data memory write) signals

Please also remember that when you use port C pins for functions other than generral I/O, you must still set the corrresponding port C control register value to configure each bit to input or output mode.

PORT 2

Port 2 is an 2-bit I/O port with individually configurable pins. It is accessed directly by writing or reading the port 2 data register, P2 (R226, E2H) in set 1, bank 0. You can use port 2 for general I/O, or for the following alternative functions:

— P2.6 and P2.7 can be configured, respectively, as timer A and timer B output

The special functions that you can program using the port 2 high byte control register (timers A and B) must also be enabled in the associated peripheral. Also, when using port 2 pins for functions other than general I/O, you must still set the corresponding port 2 control register value to configure each bit to input or output mode.

PORT 2 CONTROL REGISTER

Two 8-bit control registers are used to configure port 2 pins: P2CON (F2H, set 1, bank 0) for pins P2.6–P2.7. Each byte contains four bit-pairs and each bit-pair configures one port 2 pin. The P2CON register also controls the alternative functions described above.

Port 2 (Continued)

Port 2 Control Register (P2CON)

Four bit-pairs in the port 2 control register (P2CON) configure port 2 pins P2.6–P2.7 to Schmitt trigger input or push-pull output mode. In addition, you can select alternate functions for P2.6–P2.7.

Table 9–3. Port Data Register Summary

P2CON Bit-Pair	Corresponding Port 2 Pin	Alternate Pin Function
Bits 0 and 1	–	–
Bits 2 and 3	–	–
Bits 4 and 5	P2.6	Timer A output function (TA)
Bits 6 and 7	P2.7	Timer B output function (TB)

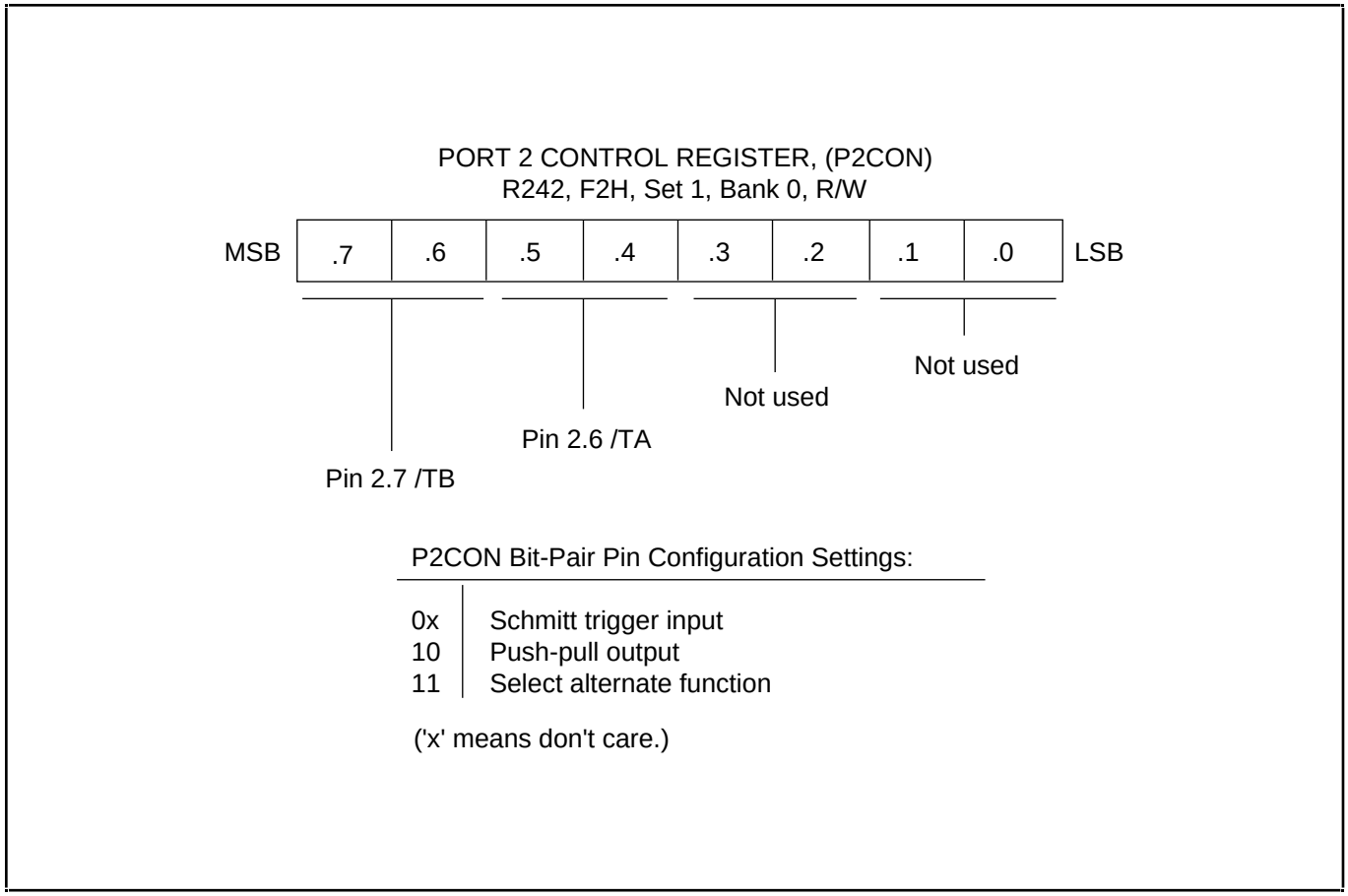


Figure 9–5. Port 2 Control Register (P2CON)

Port 3

Port 3 is an 8-bit I/O port with individually configurable pins. Port 3 pins can be used for general I/O, or for the following alternative input mode functions (except for P3.4 and P3.5, which are for general I/O only):

— Pins P3.0 and P3.1 can be used as clock inputs to timer/counters C and D. The share pin names are TCCK and TDK, respectively.

— Pins P3.2 and P3.3 can be configured as gate signal inputs for timer C and timer D (TCG and TDG, respectively)

— The lower nibble port 3 pins (P3.0–P3.3) can also serve as external interrupt inputs INT0–INT3, respectively

— P3.6 can be used as a capture data input pin for the PWM module (CAP)

— P3.7 can be used as a WAIT signal input line for the external interface (WAIT)

Port 3 is accessed directly by writing or reading the port 3 data register, P3 (R227, E3H) in set 1, bank 0. Each bit is configured for either

Schmitt trigger input or push-pull output. If you configure a port 3 pin to input mode, the alternative input function is also configured. To enable the special function, however, the corresponding enable bit must also be set in the respective peripheral control register.

The special I/O functions you can configure using the port 3 control registers — WAIT, CAP, TDG, TCG, TDCK and TCCK — must also be enabled in the associated peripheral device. When you use port 3 pins for functions other than general I/O, remember that the port 3 control registers must still written to specify which pins are set to input mode and which to output mode.

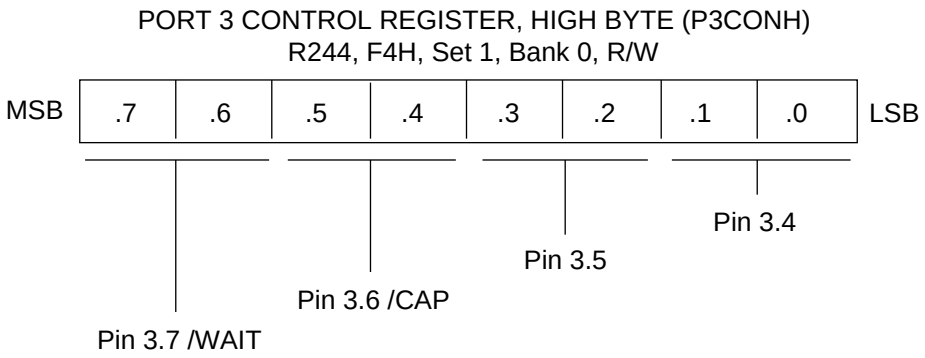
Port 3 Control Registers

Two 8-bit control registers are used to configure port 3 pins: P3CONH (R244, F4H, set 1, bank 0) for pins P3.4–P3.7 and P3CONL (R245, F5H, set 1, bank 0) for pins P3.0–P3.3. Each byte contains four bit-pairs; each bit-pair configures a specific port 3 pin. In addition to the basic port 3 I/O configuration options, the P3CONH and P3CONL registers are used to configure the various alternative function settings described above.

Port 3 High-Byte Control Register (P3CONH)

A reset operation clears the P3CONH register to '00H'. This configures all high byte pins (P3.4–P3.7) to normal Schmitt-trigger input mode. It also configures the WAIT and CAP input functions at P3.7 and P3.6, respectively. To configure individual pins as normal push-pull outputs, you must set the appropriate P3CONH bit-pair values to '11B'.

To enable the WAIT function for P3.7, you must first set bit 7 in the external memory timing register, EMT (FEH, set 1, bank 0), to "1". To enable the CAP function for P3.6, you must first set bit-pair 0/1 in the PWMCON register (FCH, set 1, bank 1) to any of the three capture enable modes. A reset sets this PWMCON bit-pair to '00B', disabling the capture function.



P3CONH Bit-Pair Pin Configuration Settings:

0x	Schmitt trigger input
10	Schmitt trigger input, pull-up
11	Push-pull output

('x' means don't care.)

NOTES:
1. The alternate function for P3.6 and P3.7 is automatically configured when the pin is set to input mode.
2. To enable the WAIT or CAP input function, you must first make the appropriate settings in the EMT and PWMCON registers, respectively.

Figure 9–7. Port 3 High-Byte Control Register (P3CONH)

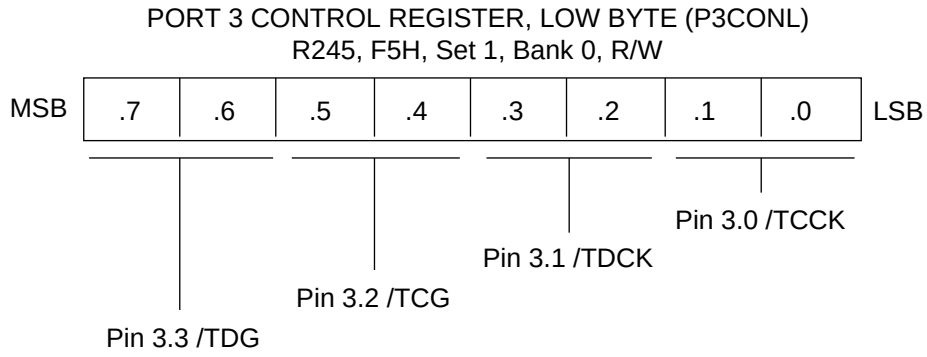
Port 3 Low-Byte Control Register (P3CONL)

A reset operation clears all P3CONL register values to 00H. This configures all low byte pins (P3.0–P3.3) to Schmitt-trigger input mode with falling-edge interrupts.

It also configures the TCCK, TDCK, TCG, and TDG functions at P3.0–P3.3, respectively.

By setting bit-pair values in the P3CONL register to '11B', you can also configure individual pins as normal push-pull outputs.

If you intend to use the alternative timer module 1 functions of the port lower nibble pins, you must enable the function by setting the appropriate control bits in the peripheral control register T1MOD (FBH, set 1, bank 0).



P3CONL Bit-Pair Pin Configuration Settings:

00	Schmitt trigger input, falling-edge interrupt
01	Schmitt trigger input, rising-edge interrupt
10	Schmitt trigger input, falling-edge interrupt, pull-up
11	Push-pull output

NOTES:
1. The alternate function for pins P3.3–P3.0 is configured by any one of the three input mode selections.
2. To enable an alternate timer module 1 input function, you must first make the appropriate settings in the T1MOD register.

Figure 9–8. Port 3 Low-Byte Control Register (P3CONL)

Port 3 Interrupt Enable and Pending Registers

Figures 9–9 and 9–10 show the control settings for the port 3 interrupt enable register P3INT and the port 3 interrupt pending register P3PND. Please note that the upper byte (bits 4–7) is not mapped. See also the detailed port 3 register descriptions in Section 4, "Control Registers."

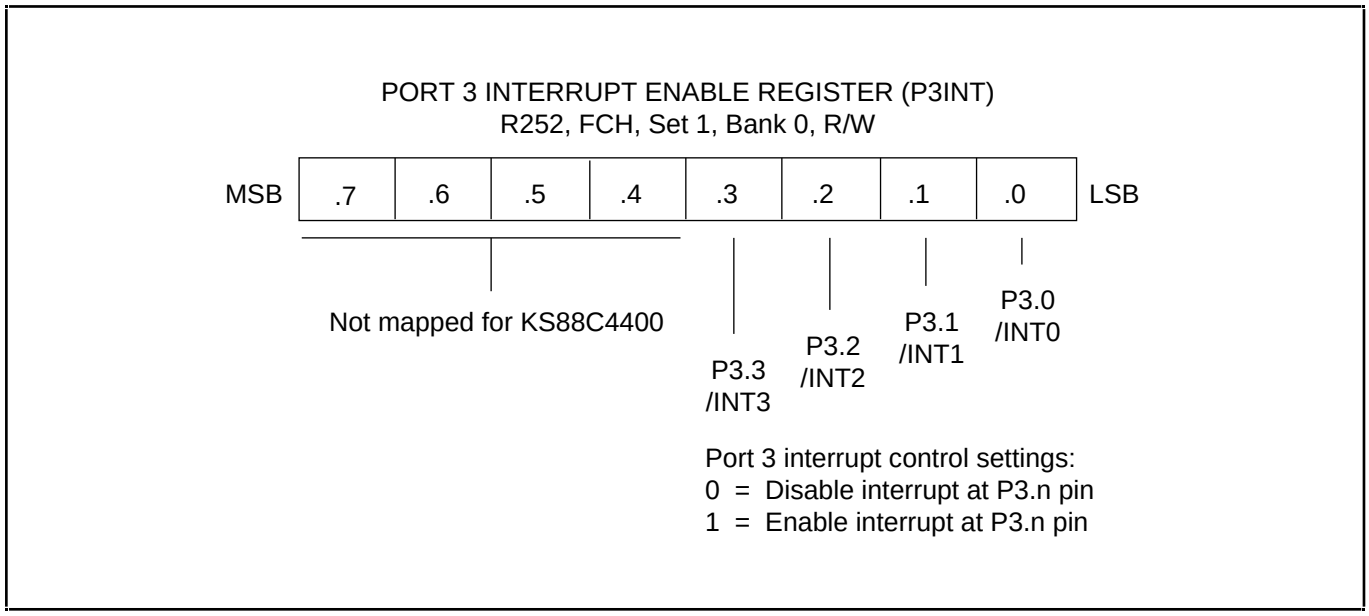


Figure 9–9. Port 3 Interrupt Enable Register (P3INT)

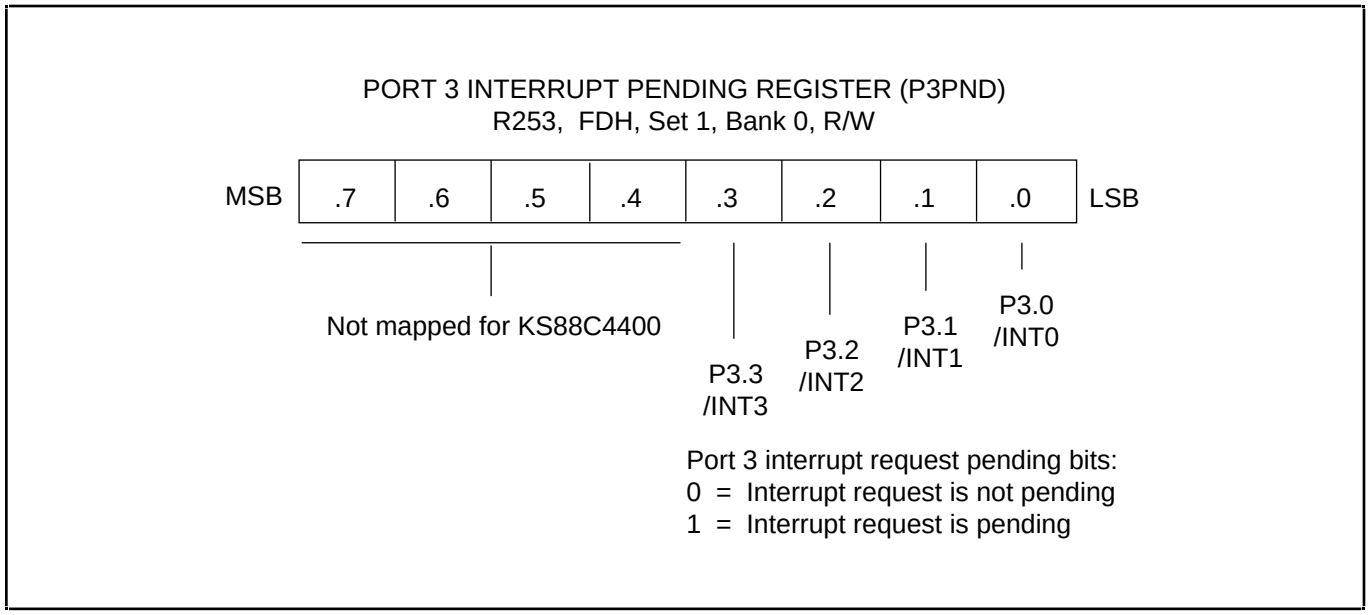


Figure 9–10. Port 3 Interrupt Pending Register (P3PND)

Port 4

Port 4 can serve either as a general-purpose 8-bit I/O port or its pins can be configured individually as external interrupt inputs. All inputs are Schmitt-triggered. Port 4 is accessed directly by writing or reading the port 4 data register, P4 (R228, E4H) in set 1, bank 0.

PORT 4 CONTROL REGISTERS

The direction of each port pin is configured by bit-pair settings in two control registers: P4CONH (high byte, F6H, set 1, bank 0) and P4CONL (low byte, F7H, set 1, bank 0). P4CONH controls pins P4.0–P4.3 (pins 33–36) and P4CONL controls pins P4.4–P4.7 (pins 37–40). Both registers are read-write addressable using 1-bit or 8-bit instructions.

When output mode is selected, a push-pull circuit is automatically configured. In input mode, three interrupt trigger selections are available: falling edge, rising edge, and falling edge detection with pull-up resistor.

A reset clears all P4CONH and P4CONL bits to logic zero. This configures port 4 pins to Schmitt trigger input with falling-edge triggered interrupts.

Port 4 Interrupt Enable and Pending Registers (P4INT, P4PND)

To process external interrupts, two additional control registers are provided: the port 4 interrupt enable register, P4INT (R240, F9H, set 1, bank 0) and the port 4 interrupt pending register, P4PND (R212, D4H, set 1).

By setting bits in the port 4 interrupt enable register P4INT to "1", you can use specific port 4 pins to generate interrupt requests when specific signal edges are detected. The interrupt names INT4–INT11 correspond to pins P4.0–P4.7. After a reset, P4INT bits are cleared to '00H', disabling all external interrupts.

The port 4 interrupt pending register P4PND lets you check for interrupt pending conditions and clear the pending condition when the interrupt request has been serviced. Incoming interrupt requests are detected by polling the P4PND bit values.

When the interrupt enable bit of any port 4 pin is set to "1", a rising or falling signal edge at that pin generates an interrupt request. (Remember that the port 4 interrupt pins must first be configured by setting them to input mode in the corresponding P4CONH or P4CONL register).

The corresponding P4PND bit is then set to "1" and the IRQ pulse goes low to signal the CPU that an interrupt request is waiting.

When a port 4 interrupt request has been serviced, the application program must clear the appropriate interrupt pending register bit by writing a "1" to the correct pending bit in the P4PND register. Please note that writing a "0" value has no effect.

Since port 4 is not used for the external peripheral interface, it functions identically in normal operating mode and in ROM-less mode.

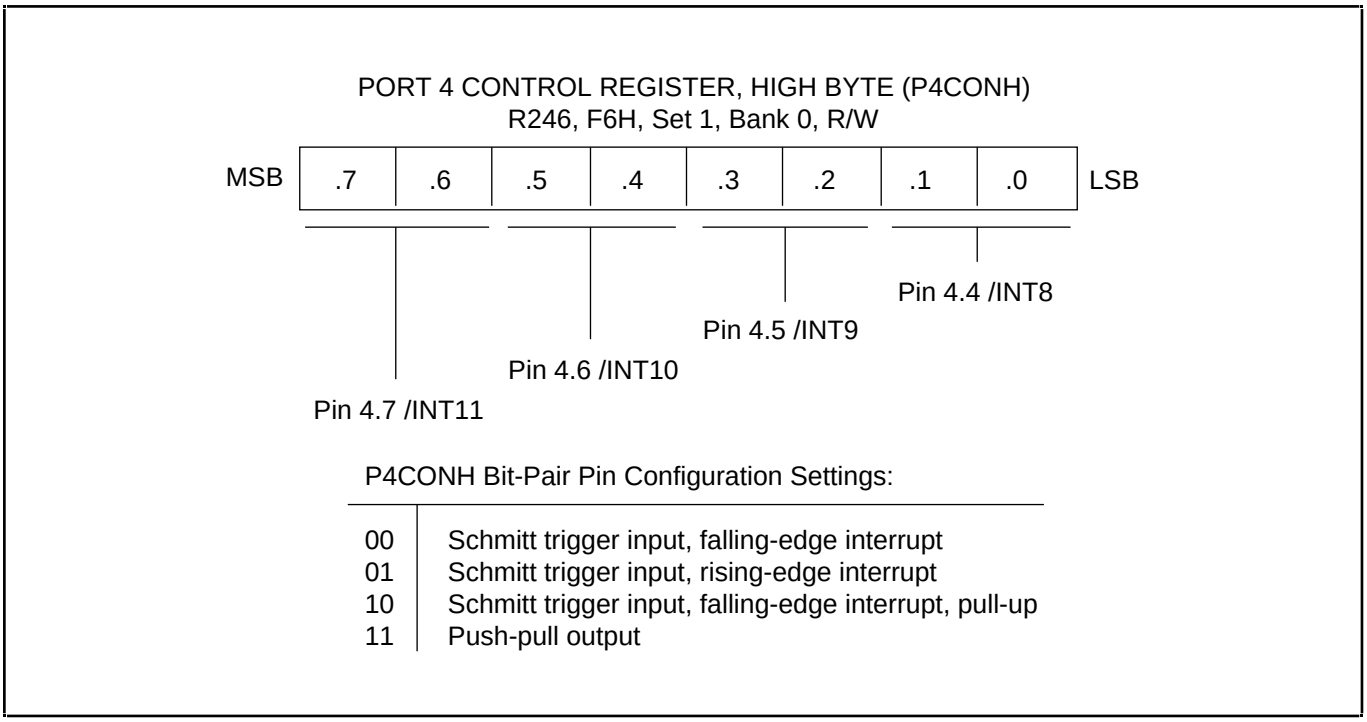


Figure 9–11. Port 4 High-Byte Control Register (P4CONH)

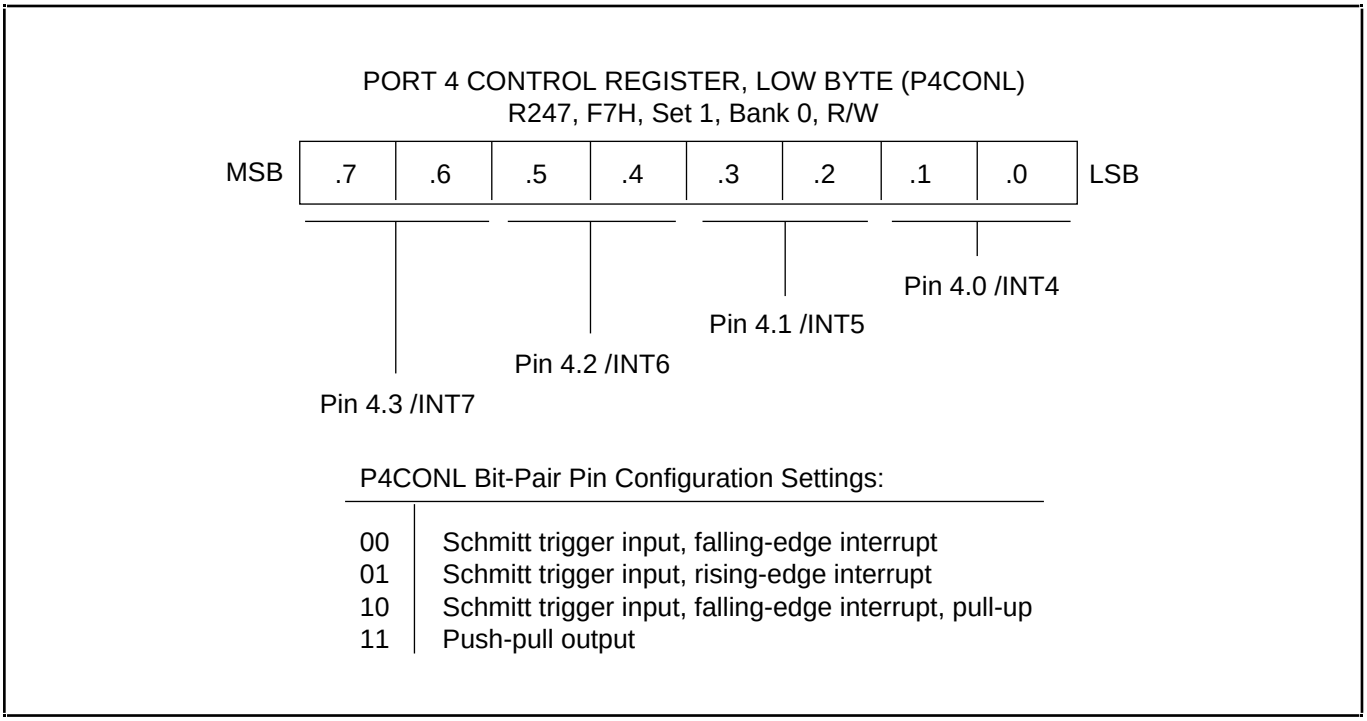


Figure 9–12. Port 4 Low-Byte Control Register (P4CONL)

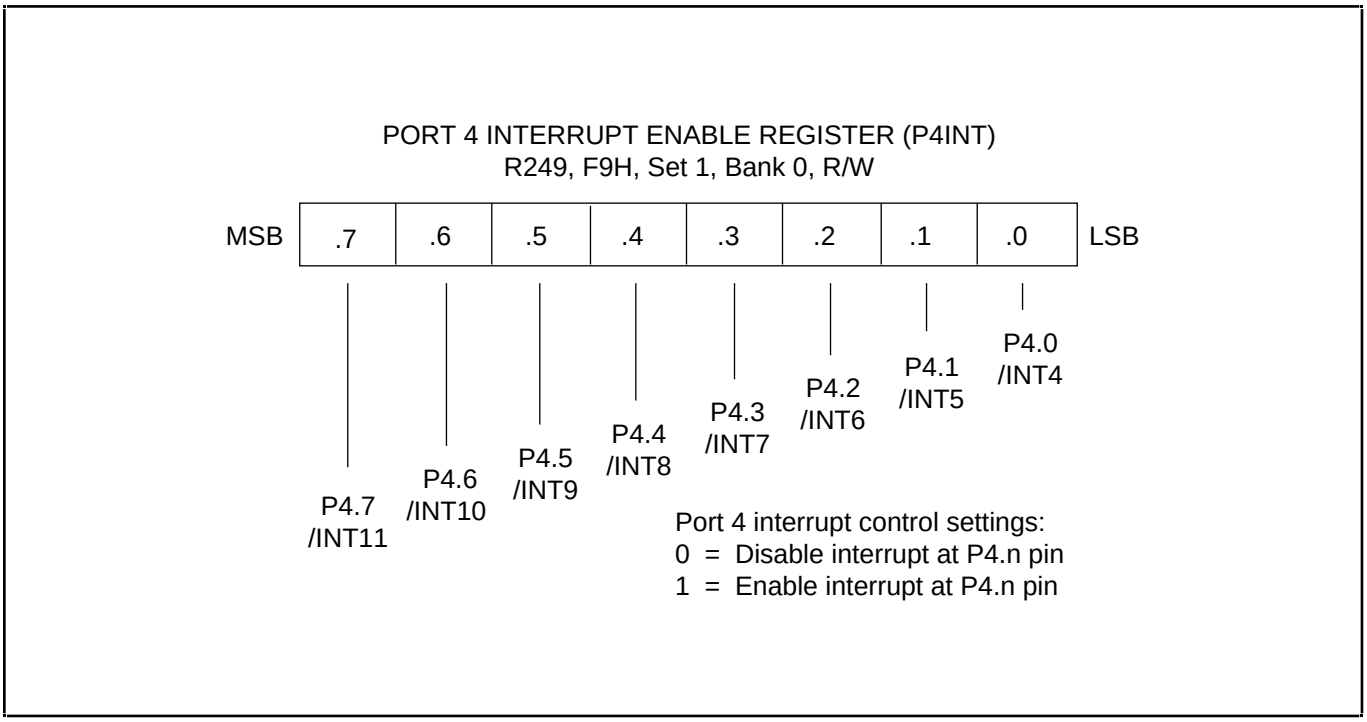


Figure 9–13. Port 4 Interrupt Enable Register (P4INT)

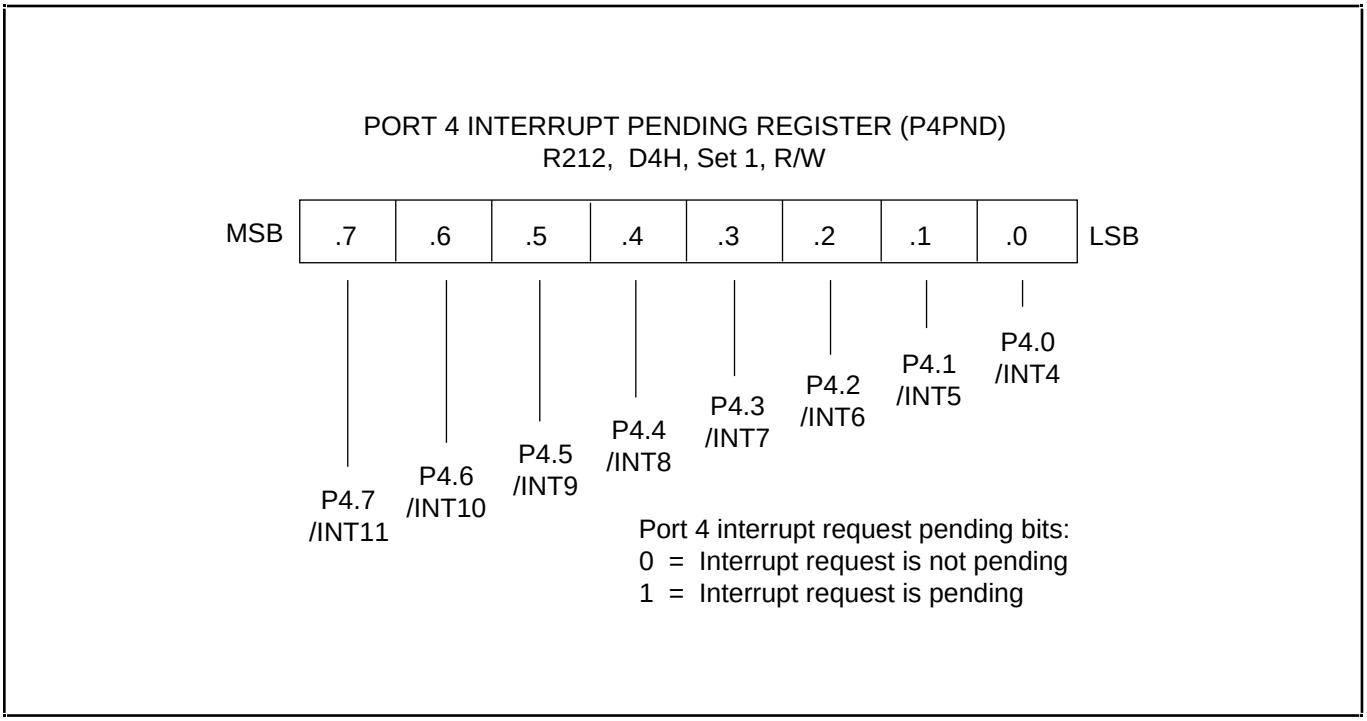


Figure 9–14. Port 4 Interrupt Pending Register (P4PND)

Port 5

Port 5 is a general-purpose 8-bit I/O port with nibble-programmable pins. Port 5 is accessed directly by writing or reading the port 5 data register, P5 (R229, E5H) in set 1, bank 0.

The port 5 control register, P5CON (R248, F8H), also located in set 1, bank 0, controls the direction of the I/O pins. P5CON settings lets you optionally configure a pull-up resistor in input mode, and select push-pull, open-drain, or open-drain with pull-up options for output mode.

Bits 0–3 of the P5CON register control the lower nibble pins (P5.0–P5.3) while bits 4–7 control the upper nibble configuration (P5.4–P5.7).

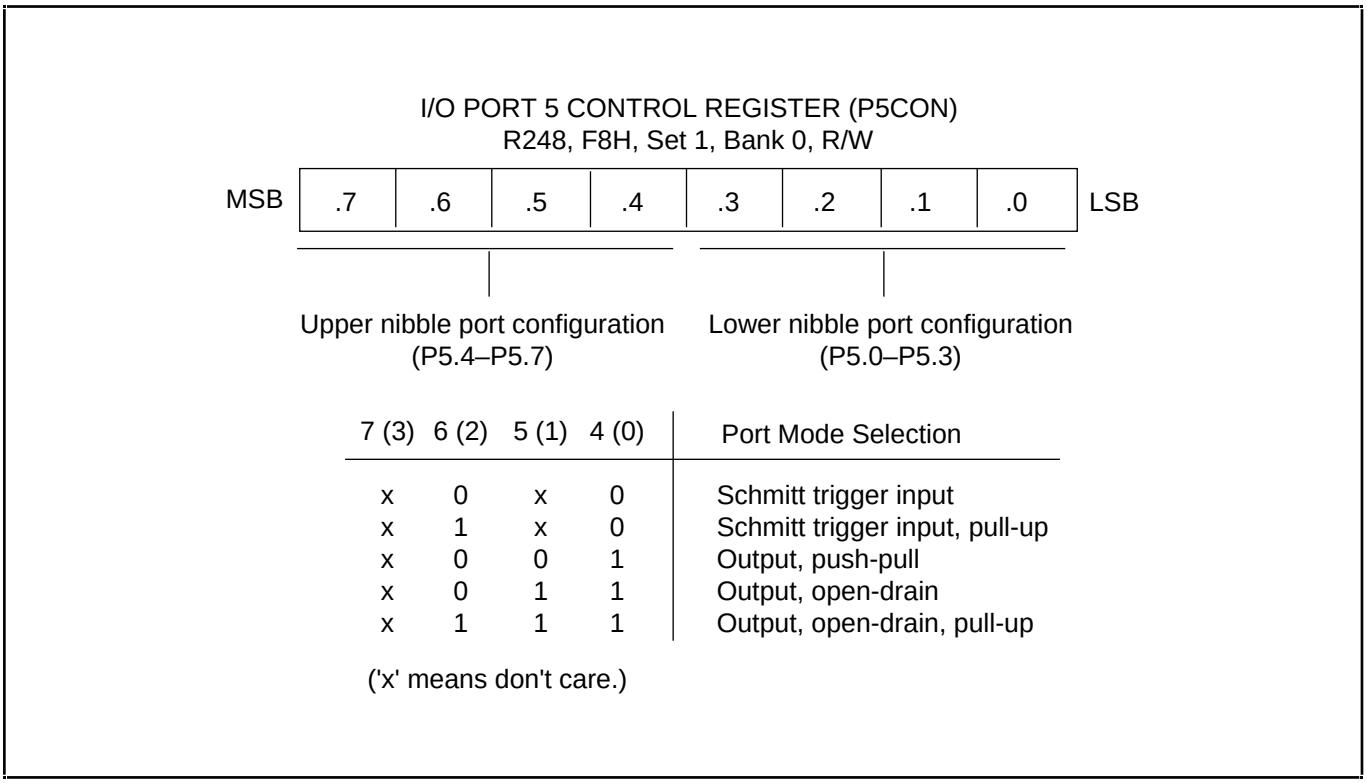


Figure 9–15. Port 5 Control Register (P5CON)

PORT 6

Port 6 is an n-channel, open-drain output port that is accessed directly by addressing the P6 data register at location E6H in set 1, bank 0. Since port 6 has no configuration options, it does not have a control register. Port 6 is designed to be used in high-voltage drive applications, and can withstand up to 9 V loads.

PORT 7

The 8-bit input port (pins 41, 43, 44, and 46–50) can be used either as analog inputs for the A/D converter module or as general input port pins P7.0–P7.7. Incoming port 7 data values are read directly from the A/D converter's 8-bit digital input register ADIN, located in set 1, bank 1 at address F9H.

☞ Programming Tip — Configuring KS88C4400 Port Pins to Specification

This example shows how to configure KS88C4400 I/O ports according to sample specifications. The program configures ports 0 through 6 as follows:

— P2.6 and P2.7 to input mode
— Set P3.0 to input mode
— Set P3.1–P3.7 to push-pull output mode
— Set P4.0–P4.4 to push-pull output mode
— Set P4.5 to input mode with rising-edge interrupts
— Set P4.6 to input mode with falling-edge interrupts
— Set P4.7 to input mode, falling-edge interrupts, push-pull
— Set P5.0–P5.7 to open-drain output mode with pull-up
— Port 6 is automatically set to output mode, open-drain type (there is no port 6 control register)

```
•
•
•
LD      P2CON,#00H      ;  P2.6 and P2.7 ← input mode
LD      P3CONH,#0FFH    ;  P3.1–P3.7 ← output, push-pull
LD      P3CONL,#0FEH    ;  P3.0 ← input, timer C clock input enable
LD      P3INT,#00H      ;  Disable interrupts INT0–INT3
LD      P4CONH,#087H    ;  P4.7 ← input, falling edge interrupt, pull-ups
LD      P4CONL,#0FFH    ;  P4.6 ← input, falling edge interrupt
                        ;  P4.5 ← input, rising edge interrupt
                        ;  P4.0–P4.4 ← output, push-pull
LD      P5CON,#77H      ;  P5.0–P5.7 ← output, open-drain, pull-ups
LD      P4PND,#0FFH     ;  Reset all pending register bits for port 4 interrupts
LD      P4INT,#0E0H     ;  Enable port 4 interrupts
•
•
•
```

10 Timer Module 0

OVERVIEW

The KS88C4400 timer module 0 (T0) has two 8-bit timers, timer A and timer B. Each timer has an 8-bit counter register, an 8-bit data register, an 8-bit comparator, and a corresponding output pin. Two control registers, T0CON and TBCON, control timer module 0 operation. Timers A and B run continuously. Counter values cannot be modified or reset because they do not have mapped register addresses.

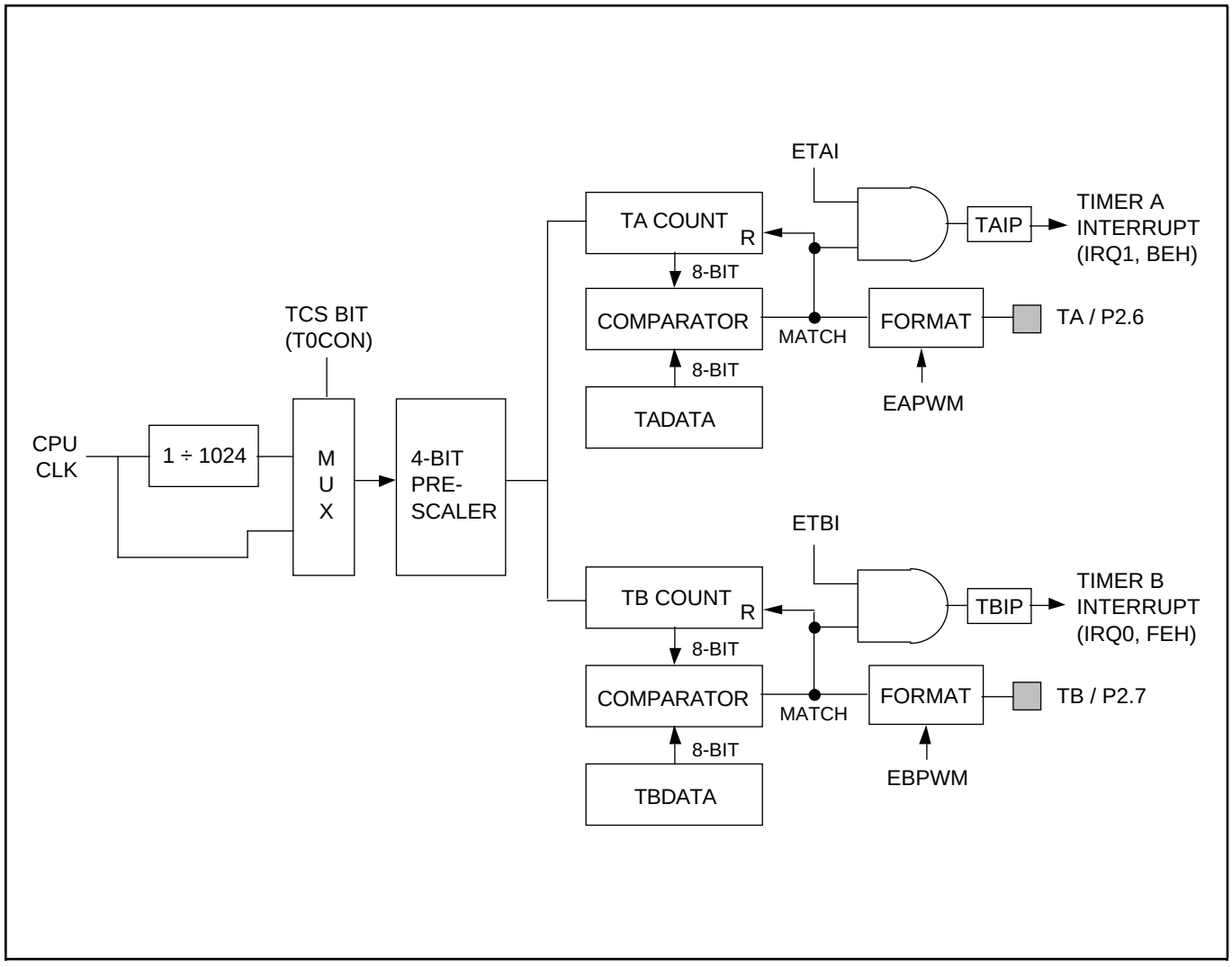


Figure 10–1. Timer Module 0 Function Block Diagram

Timer A and B Operating Modes

Both timer A and B operate either in interval mode or pulse width modulation (PWM) mode. The LSB of the T0CON register controls the timer A operating mode, and the LSB of the TBCON register controls the operating mode for timer B.

TIMER CLOCK INPUT

Timers A and B are driven by the same clock input. There are two options for timer clock input: the non-scaled CPU clock or the CPU clock divided by 1024 (decimal).

When the TCS bit (bit 3) in the T0CON register is "0", the T0 module runs on the divided-by-1024 CPU clock. When TCS = "1", T0 runs on the non-scaled CPU clock pulse. The CPU clock frequency is scaled using the 4-bit prescaler in bits 4–7 of the T0CON register (TPS3–TPS0).

TIMER A AND TIMER B INTERRUPT CONTROL

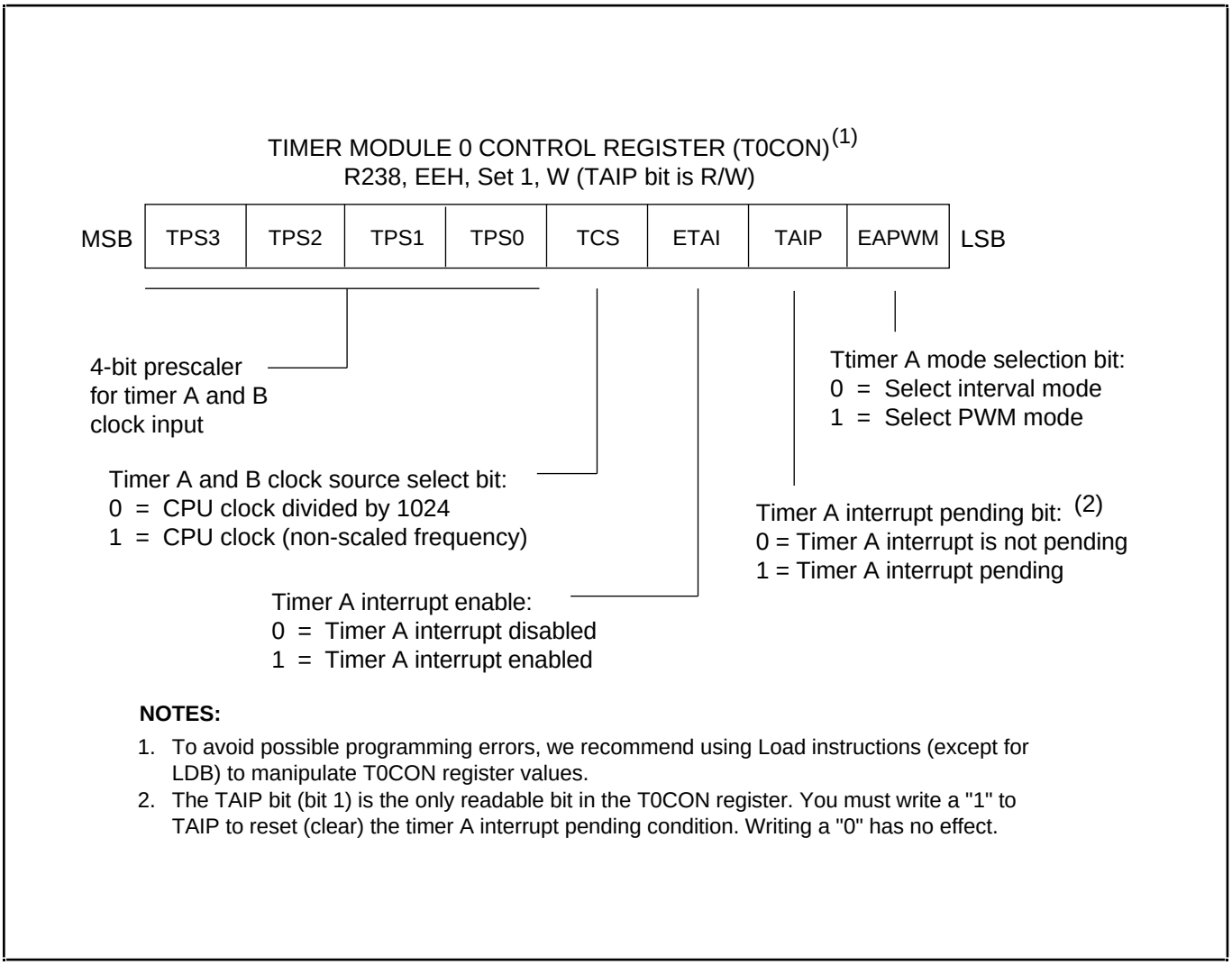
In interval mode, both timers generate a match signal when the count value and the referenced data value in the TADATA or TBDATA register is the same. When the interrupt enable bit is set for timer A or timer B, an interrupt is generated when the match is detected. The corresponding count register is cleared and counting resumes.

You enable the timer A interrupt by setting the ETAI bit (bit 2) in the T0CON register. To enable the timer B interrupt, you set the ETBI bit (bit 2) in the TBCON register.

The timer A and B interrupt pending bits, TAIP and TBIP, are located in the T0CON and TBCON registers, respectively. These bits can be polled by software to detect interrupt pending conditions. When a pending bit read operation shows a "0" value, no interrupt is pending; when it is "1", it means that a timer A or B interrupt is pending.

When the interrupt is acknowledged and the service routine has been initiated, the pending bit must be cleared by software. To do this, you must write a logic one value ("1") to the TAIP or TBIP bit — writing a "0" value has no effect.

Timer Module 0 Control Register (T0CON)



NOTES:

1. To avoid possible programming errors, we recommend using Load instructions (except for LDB) to manipulate T0CON register values.
2. The TAIP bit (bit 1) is the only readable bit in the T0CON register. You must write a "1" to TAIP to reset (clear) the timer A interrupt pending condition. Writing a "0" has no effect.

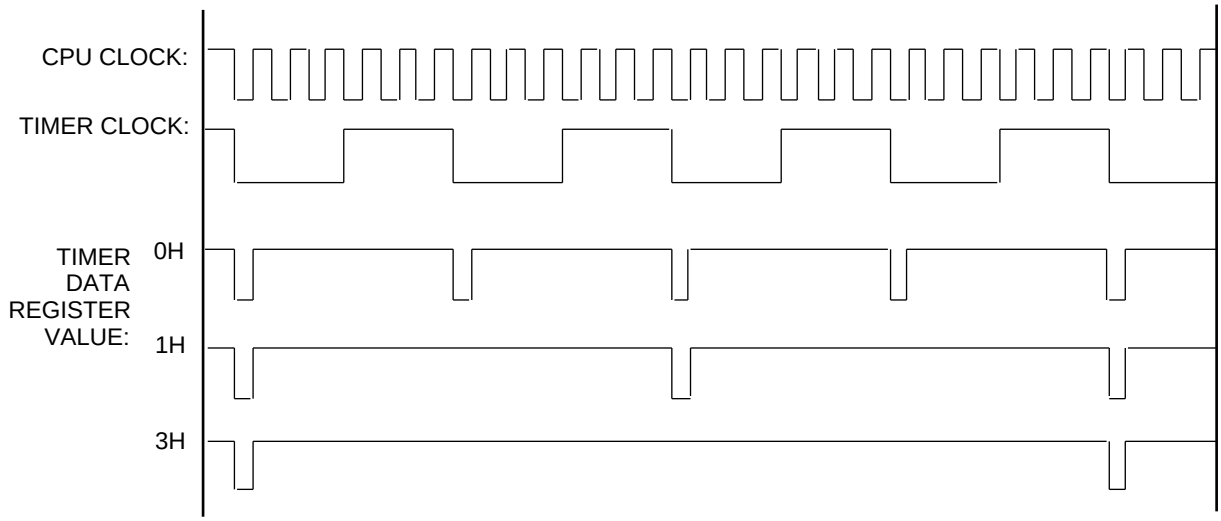
Figure 10–2. Timer Module 0 Control Register (T0CON)

TimeR Module 0 Function Description

Timer A and timer B both operate in either interval mode or pulse width modulation mode, as selected by the EAPWM bit (T0CON.0) and the EBPWM bit (TBINT.0). Timer B functions in exactly the same way as timer A.

Interval Mode

In interval mode, the TA pin toggles low during the high period of one timer A clock cycle on every match of the timer and the TADATA register. It remains high level at all other times. With each match, the timer is also reset to logic zero. In interval mode, the pulse width is fixed and the interval at which that pulse occurs is determined by the combination of the timer frequency and the value written to the data register.



NOTES:
1. To generate the timer clock, the 4-bit prescaler value '0010B' is used.
2. Note that timer A & B uses a divided-by-1024 CPU clock source, in addition to the 4-bit prescaler value.

Figure 10–3. Timer A and B Waveforms in Interval Mode

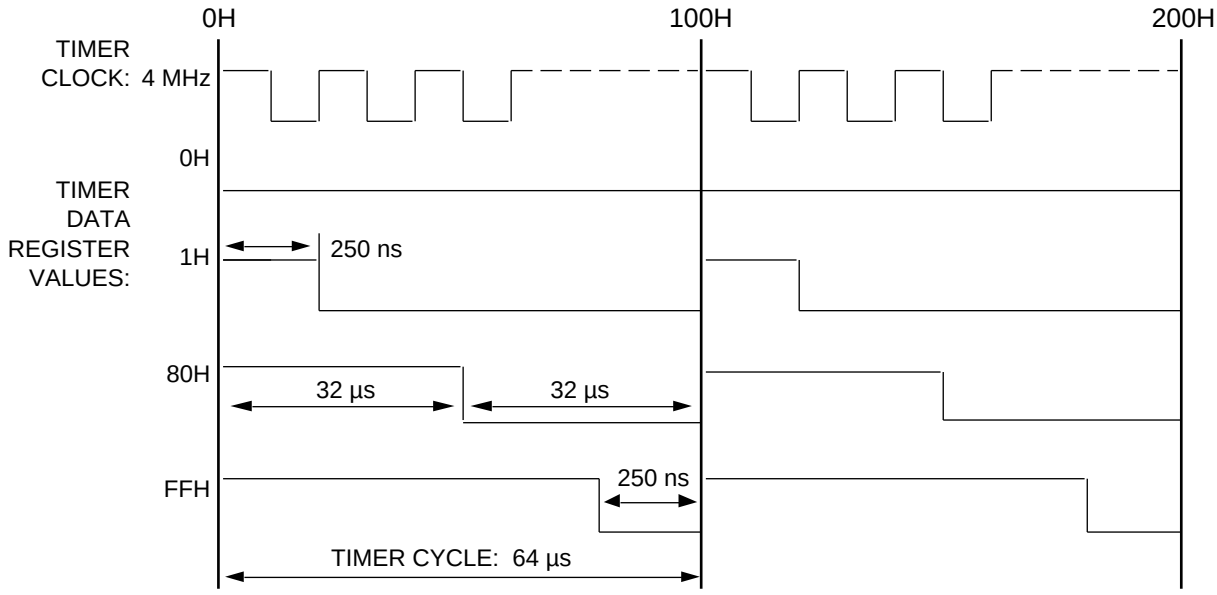
Pulse Width Modulation Mode

In PWM mode, the timer A data register is written by the CPU and used to modulate the pulse width of at output pin TA. This pin toggles at a frequency equal to the selected input clock divided by 256 of timer A (that is, by the prescaler output).

However, it will have a duty cycle from 0% to 99.6%, based on the value in the TADATA register. This is achieved by comparing the contents of the TADATA register to the 8-bit TA count value, toggling the TA pin to "1" whenever the TADATA value is greater than the TA value, and to "0" otherwise.

For example, assume the input clock to timer A is 4 MHz. The TA pin toggles high every 64 µs (4 MHz divided by 256). If the timer A data register has a value of 80H (128 decimal), the TA pin will be logic one for 128 cycles (32 µs), and logic zero for 128 cycles, for a duty cycle of 50% (128/256).

A value of '00H' in the timer A data register (which is true after a reset), will result in a constant "0" from the TA pin. A value of FFH therefore generates a 99.6% duty cycle (255/256).



NOTE: A 4-MHz timer clock value and a prescaler value of 00H are assumed.

Figure 10–4. Timer A and B Waveforms in Pulse Width Modulation Mode

Timer B Control Register (TBCON)

The timer B clock source is controlled by the T0CON register. Timer B also has a separate control register called TBCON, located at EFH in set 1, bank 0. The TBCON register has three functions:

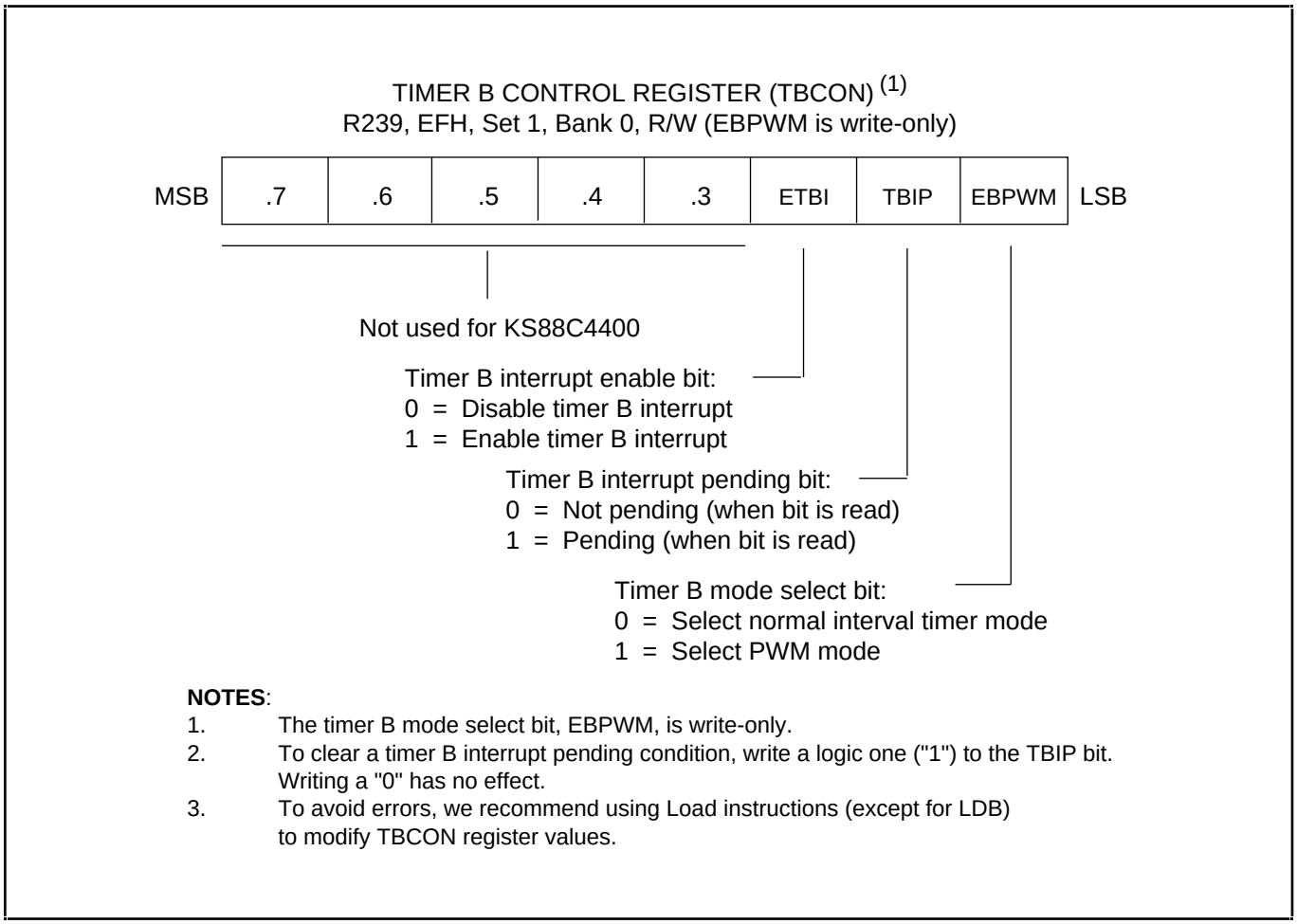
— Select timer B operating mode (interval or PWM)

— Enable the timer B interrupt

— Control timer B interrupt pending condition

The least significant bit of the TBCON register is called the EBPWM bit. When it is "0", timer B operates in normal interval timer mode; when it is "1", timer B operates in pulse width modulation (PWM) mode.

TBCON bit 2 (ETBI) is the timer B interrupt enable bit. Bit 1 (TBIP) is the timer B interrupt pending bit. Application software can poll the TBIP bit to detect a timer B interrupt pending condition. When the interrupt is acknowledged and the service routine is initiated, the TBIP bit must be cleared by software. To do this, you must write a logic one ("1") value to bit position 1; writing a "0" has no effect.



NOTES:

1. The timer B mode select bit, EBPWM, is write-only.
2. To clear a timer B interrupt pending condition, write a logic one ("1") to the TBIP bit. Writing a "0" has no effect.
3. To avoid errors, we recommend using Load instructions (except for LDB) to modify TBCON register values.

Figure 10–5. Timer B Control Register (TBCON)

☞ Programming Tip — Configuring Timer A and Timer B

This example sets timer A to normal interval mode, disables timer B, sets the oscillation frequency of the timer clock, and determines the execution sequence which follows a timer A interrupt. The program givens are as follows:

— Timer A is used in interval mode; the timer interval is set to approximately 2 milliseconds

— Timer B is disabled

— Oscillation frequency = 6 MHz

— 90H ← 90H + 91H + 92H + 93H + 94H is executed after a timer A interrupt

```
                ORG     0020H           ;  Reset address
                JP      T,START
                ORG     00BEH           ;  Timer A interrupt vector
                VECTOR  TA_int
                ORG     00FEH           ;  Timer B interrupt vector
                VECTOR  TB_int
                ORG     0100H
                •
                •
                •
START           DI
                •
                •
                •
                LD      PP,#00H         ;  Set page pointer
                LD      T0CON,#56H      ;  PS ← 5 (for divide-by-6)
                                        ;  CPU clock /1024 is selected for timer 0 (A & B)
                                        ;  Enable timer A interrupt, reset timer A pending register
                                        ;  Select interval mode for timer A
                LD      TADATA,#01H     ;  TADATA ← 01 (divided by two)
                                        ;  6 MHz /1024 ÷ 6 ÷ 2 = 0.5 kHz (= 2 ms)
                LD      TBCON,#02H      ;  Disable timer B interrupt
                EI                      ;  Enable interrupts
                •
                •
                •
TA_int          PUSH    RP0             ;  Save RP0 to stack
                SRP0    #90H            ;  RP0 ← 90H
                ADD     R0,R1           ;  R0 ← R0 + R1
                ADC     R0,R2           ;  R0 ← R0 + R2
                ADC     R0,R3           ;  R0 ← R0 + R3
                ADC     R0,R4           ;  R0 ← R0 + R4
                                        ;  (R0 ← R0 + R1 + R2 + R3 + R4)
                LD      T0CON,#56H      ;  Reset timer A pending register
                POP     RP0             ;  Restore register pointer 0 value
                IRET                    ;  Return from interrupt service routine
                •
                •
                •
TB_int          LD      TBCON,#02H
                IRET
```

note

11 Timer Module 1

OVERVIEW

The KS88C4400 has two 16-bit timer/counters, called timer C and D. Both can be configured to operate either as timers or as event counters. The functional components of the timer 1 module are summarized as follows:

— Timer module 1 control register (T1CON)
— Timer module 1 mode register (T1MOD)
— Timer C, D high-byte count registers (TCH, TDH)
— Timer C, D low-byte count registers (TCL, TDL)
— Timer C gate pin (TCG /P3.2)
— Timer D gate pin (TDG /P3.3)
— Timer C external clock input pin (TCCK /P3.0)
— Timer D external clock input pin (TDCK /P3.1)

Timer module 1 can be programmed to operate in four different modes (operating mode is controlled by bit settings in the timer 1 mode register, T1MOD):

Mode 0	13-bit timer/counter
Mode 1	16-bit timer/counter
Mode 2	8-bit auto-reload timer/counter
Mode 3	Two 8-bit timer/counters

When used as an interval timer, the corresponding count register is incremented based on the internal timer clock rate. The timer clock can be selected as either an external clock source, or a divided-by-6 internal CPU clock.

When used as an event counter, the timer's count register is incremented in response to a 1-to-0 transition at its corresponding external input pin (TCCK for timer C or TDCK for timer D). The external input is sampled every at fifth CPU clock pulse.

When a high-level sample is immediately followed by a low-level sample (that is, when a 1-to-0 transition occurs), the count register is incremented by one. (Note that the new count value is actually written to the count register five CPU clocks *after* the one in which the 1-to-0 transition was detected.) It therefore takes two complete sampling cycles (12 CPU clocks) to recognize a 1-to-0 transition.

There are no restrictions on the duty cycle of the external signal input, but to ensure that a given level is sampled at least once before it changes, it should be held for at least the equivalent of 6 CPU clocks.

Timer Module 1 Mode Register (T1MOD)

The timer 1 mode register T1MOD (R251, FBH, set 1, bank 0) controls the following timer C and D functions:

— Clock source selection for timer/counter operation

— Gate function enable/disable

— Operating mode selection (four available modes)

The lower nibble bits (0–3) correspond to timer C and the upper nibble bits (4–7) correspond to timer D. After a reset, T1MOD values are cleared to logic zero. The '00H' setting selects the CPU clock (divided by 6) as the clock source, disables the gate functions, and configures timers C and D to 13-bit timer/counter mode.

Clock Source Selection Options

Timers C and D each have two clock source selection options: 1) the internal CPU clock pulse, divided by 6, or 2) an external clock source.

If the divided-by-6 CPU clock is selected as the timer C or timer D clock (TxC = "0"), the timer functions as an interval timer; if an external clock is selected (TxC = "1"), it functions as an event counter for an external device.

The clock source select bits in T1MOD are bit 2 for timer C (TCC) and bit 6 for timer D (TDC). If an external clock source is used, the corresponding input pin must be configured: for timer C, the clock input pin is TCCK (P3.0, pin 25); for timer D, the external clock input pin is TDCK (P3.1, pin 24). The port 3 low-byte control register P3CONL configures the clock input pins for this timer module 1 function.

Gate Function Enable

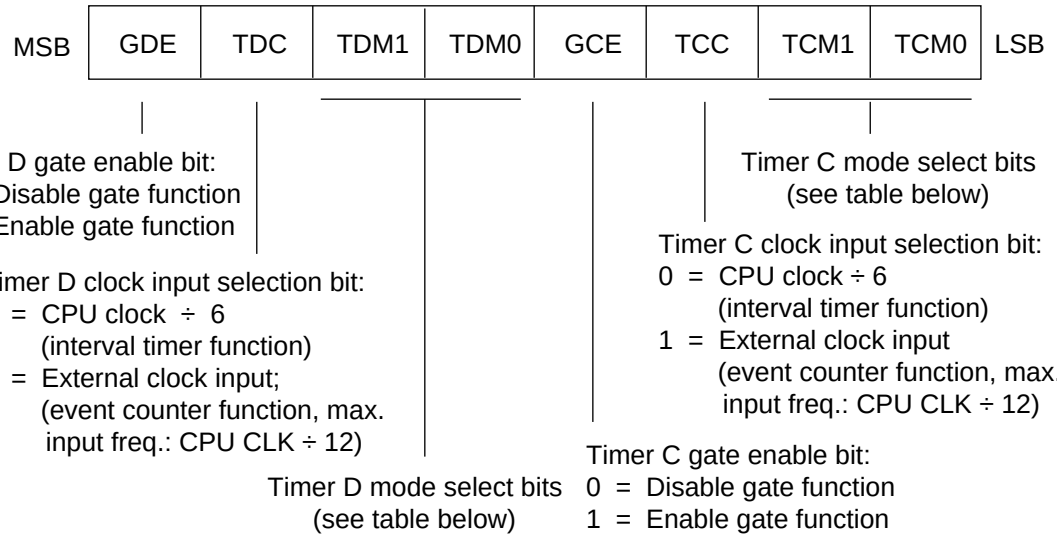
Timers C and D each have a gate function enable bit in the T1MOD register: bit 3 (GCE) is for timer C and bit 7 (GDE) for timer D. After a reset, the GxE bits are cleared to "0", and the gate function is turned off.

The timers can be enabled in one of the four available timer module 1 operating modes independently of the gate control function. The gate function is described below in the subsection, "Timer Module 1 Gate Function Description."

Timer Module 1 Operating Modes

Two bit-pairs in T1MOD (bit-pair 0/1 for timer C and bit-pair 4/5 for timer D) are used to select one of the four available operating modes for timer module 1. Modes 0, 1, and 2 are functionally identical for both timers. Mode 3 operates differently for timer C and D. Each operating mode is described in detail in the following subsections.

TIMER MODULE 1 MODE REGISTER (T1MOD)
R251, FBH, Set 1, R/W



TxM1	TxM0	Operating Mode Description
0	0	13-bit timer/counter mode.
0	1	16-bit timer/counter; TxH and TxL are cascaded.
1	0	8-bit auto-reload timer/counter; TxH holds a value to be reloaded into the TxL count register each time it overflows.
1	1	Timer C: TCL operates as an 8-bit timer/counter controlled by the standard timer C control bits; TCH is an 8-bit timer only that is controlled using timer D control bit settings. Timer D: Timer/counter D is disabled.

NOTES:

1. When you enable the gate function for timer C or timer D (GxE = "1"), the timer is incremented only when the TCG or TDG pin is held high, and if the corresponding timer enable bit is set in the T1CON register. When GxE = "0", the corresponding timer is enabled whenever its run bit is set.
2. 'x' means timer C and timer D.

Figure 11–1. Timer Module 1 Mode Register (T1MOD)

Timer Module 1 Control Register (T1CON)

The timer 1 control register T1CON (R250, FAH, set 1, bank 0) controls the following timer C and D functions:

— Timer/counter run enable bits
— Interrupt enable and interrupt pending bits
— Baud rate select bit (for UART baud rate generation)

After a reset, T1CON values are cleared to logic zero. The '00H' setting disables timers C and D, disables timer module 1 interrupt processing, and selects the normal baud rate setting for the UART baud rate generator function. Bit 6 of the T1CON register is not mapped.

Due to the complex read/write characteristics of the T1CON register, and to avoid possible program errors, we recommend using Load instructions only (except for LDB) to manipulate control values.

Timer/Counter Run Control Bits

T1CON bits 0 and 1 (TCE and TDE, respectively) are the timer C and D run control bits. The timer/ counters can be started or stopped independently of each other. Before you enable a timer by setting its run bit to "1", you must first make all the necessary control settings (for clock source, operating mode, and gate function) in the T1MOD register.

Interrupt Control Function

External interrupts can be gated to the timer 1 module via the TCG and TDG pins (pins 16 and 15, respectively). The gate function is enabled externally when these pins are set to input mode by the appropriate port 4 (P4CONL) control register settings.

When using the gate function, external interrupt INT4 is gated to timer C and INT5 is gated to timer D. This lets you use the timer as an event counter for an external device. The timer interrupt enable bits are bit 2 (TCIE) for timer C and bit 3 (TDIE) for timer D.

Each timer has an interrupt pending bit which serves as a flag for gated external interrupts. These flags can be polled by software: Bit 4 (TCIP) is the pending bit for timer C; bit 5 (TDIP) is the pending bit for the timer D interrupt.

When the appropriate interrupt enable bit is enabled, an interrupt request will branch to that timer's interrupt vector location whenever the pending flag is "1". (The branch occurs, however, only after the current instruction has executed, and if no other interrupts with higher priority are being serviced).

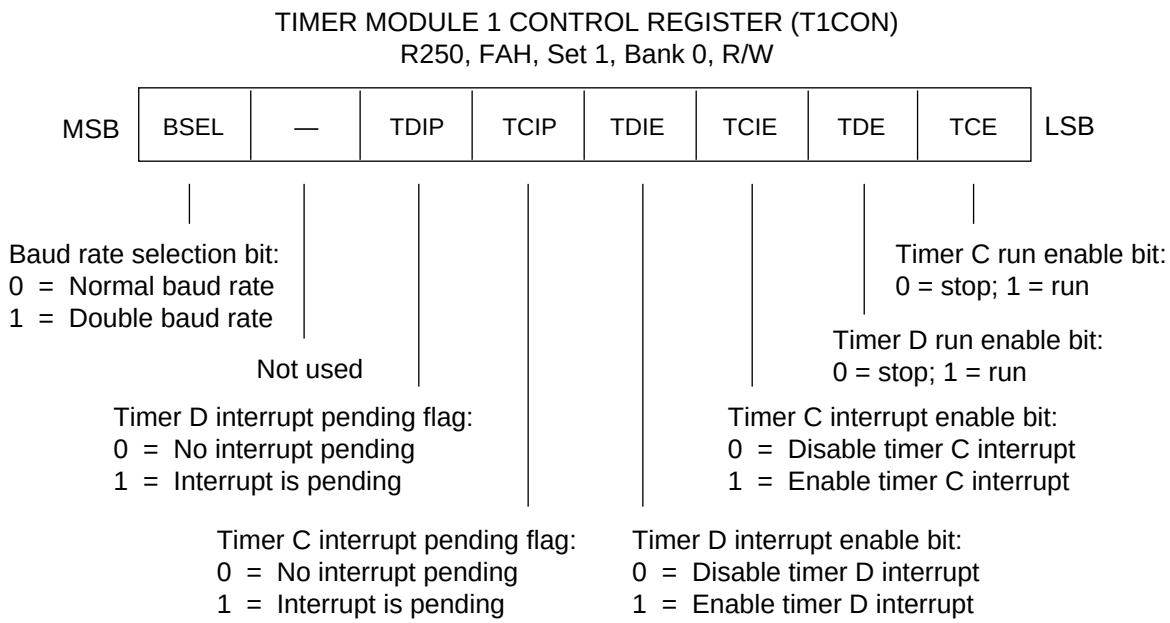
The pending flag is not automatically cleared by hardware when the branch occurs; you must clear it by software by writing a "1" to the appropriate pending bit location in the T1CON register (writing a "0" has no effect).

Baud Rate Generator Function

You can use timer module 1 as a baud rate generator for the UART module. Bit 7 of the T1CON register (BSEL) is the baud rate select bit. The "0" setting selects a normal baud rate based on the timer module 1 clock source (either CPU clock ÷ 6 or an external clock source).

The "1" setting doubles the frequency of the normal baud rate selection. If you do select the double baud rate, you must also set the UART module to operate in mode 1, 2, or 3 (mode 0 operation is not allowed).

For more information about the UART baud rate generation function, please refer to Section 12, 'Serial Port', in this manual.



NOTES:

1. Due to the complex read/write characteristics of the T1CON register, and to avoid possible program errors, we recommend using Load instructions only (except for LDB) to manipulate T1CON register values.
2. If the BSEL bit is set to "1" for double baud rate generation, the UART module must be set to operate in mode 1, 2, or 3 (that is, UART mode 0 is not a valid selection).

Figure 11–2. Timer Module 1 Control Register (T1CON)

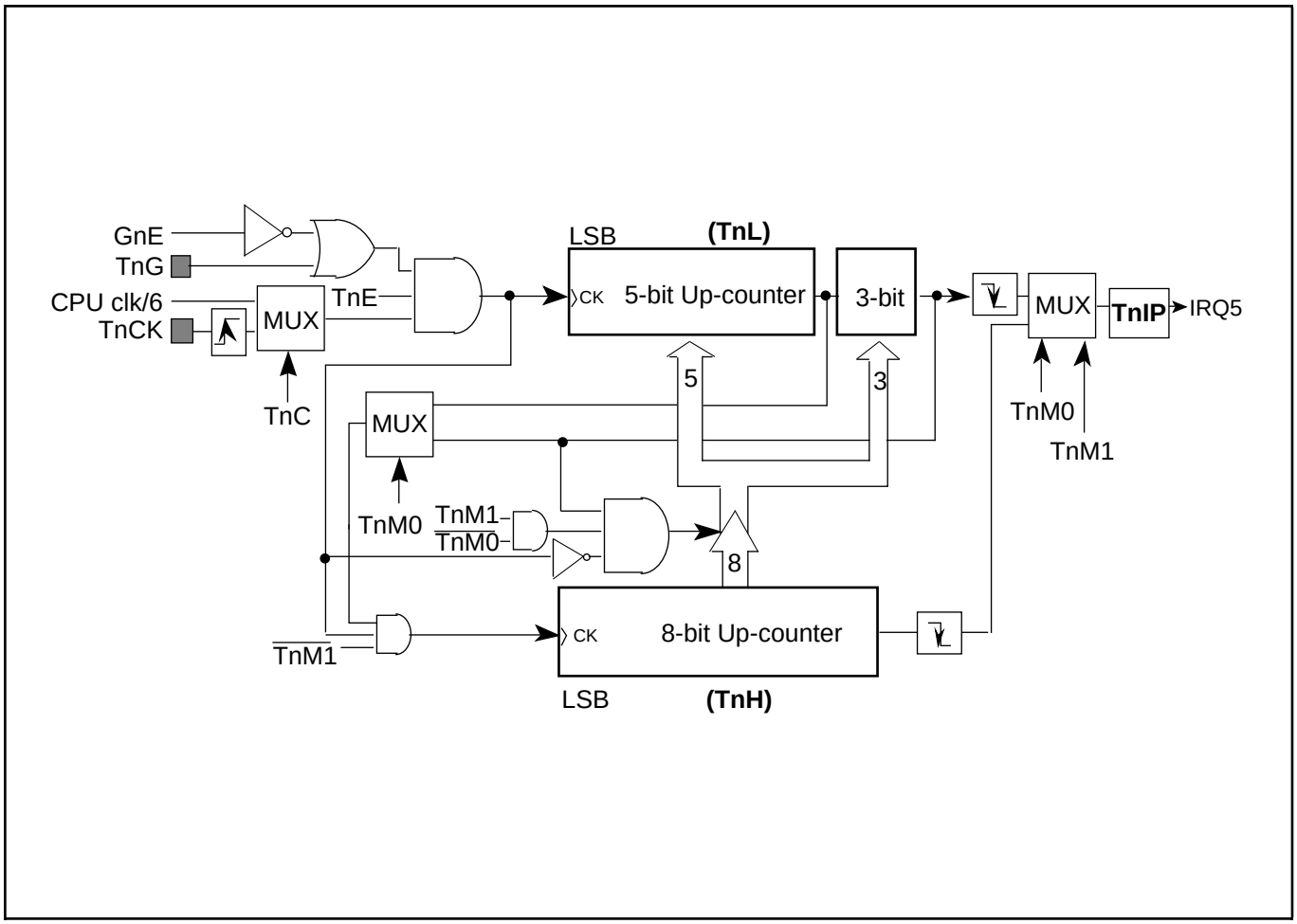


Figure 11-3. Timer C & D Block Diagram (MODE0–MODE2)

Timer Module 1 Gate Function Description

Two port 3 pins (P3.0 and P3.1) are available as interrupt input gate pins for timers C and D. These pins normally serve as input ports for external interrupts INT0 and INT1, respectively.

The P3.0/INT0 pin is used as the timer C gate TCG, and the P3.1/INT1 pin is used as the timer D gate, TDG. In order for the gate function to operate, two conditions must be met:

— The corresponding timer run bit in the T1CON register (TCE or TDE) must be set to "1"

— The corresponding external interrupt pin must be configured to input mode and be at high level

The gate function lets the timer measure the duration of pulses applied to the external interrupt pin relative to the timer's count source. If the pin's interrupt function is enabled, it will continue to trigger the corresponding external interrupt. The interrupt service routine can then read the counter value that accumulated while the signal at the TCG or TDG pin was at high level. In this way, you can use timer module 1 as an event counter for an external device.

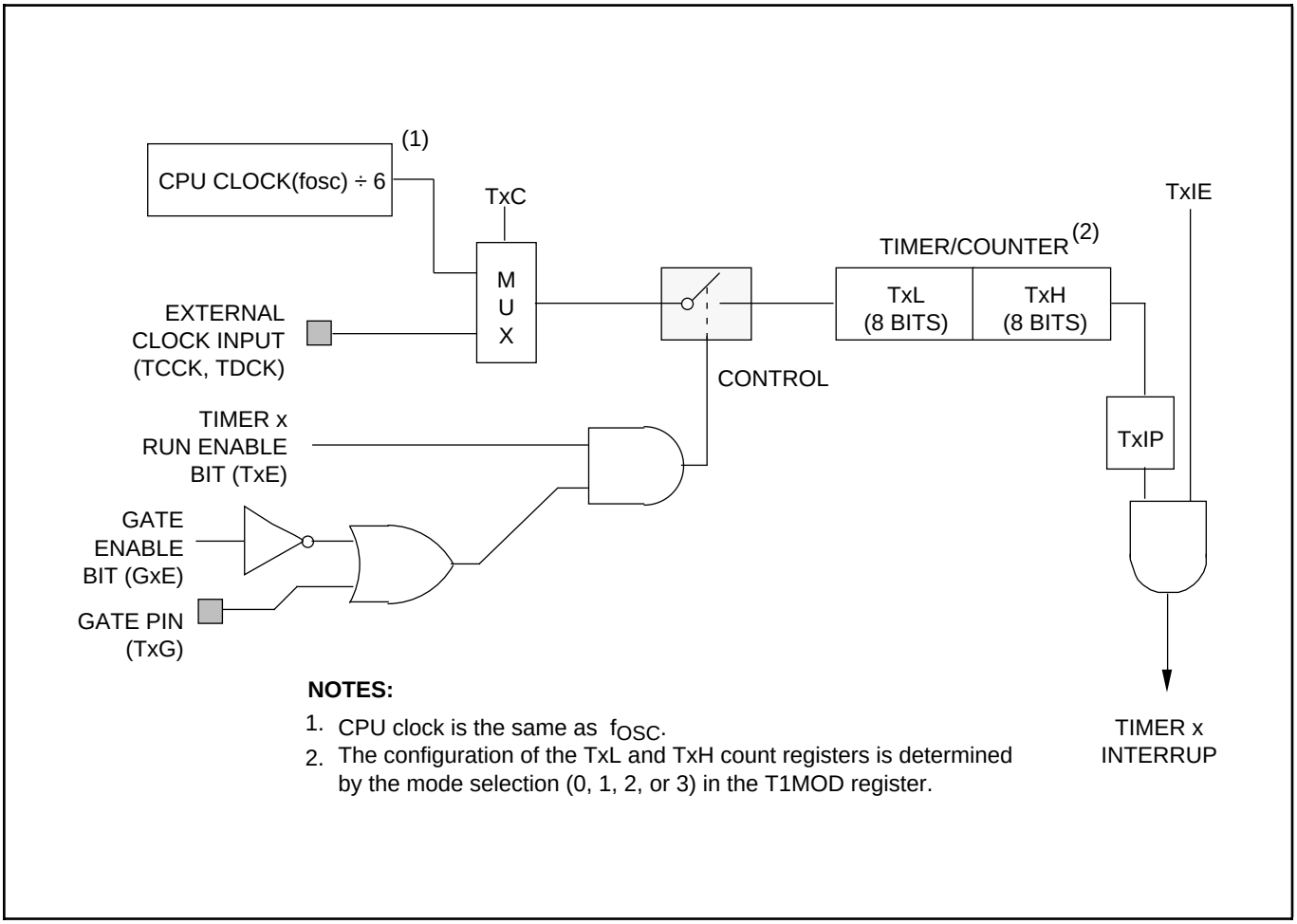


Figure 11–4. Timer Module 1 Gate Function Block Diagram

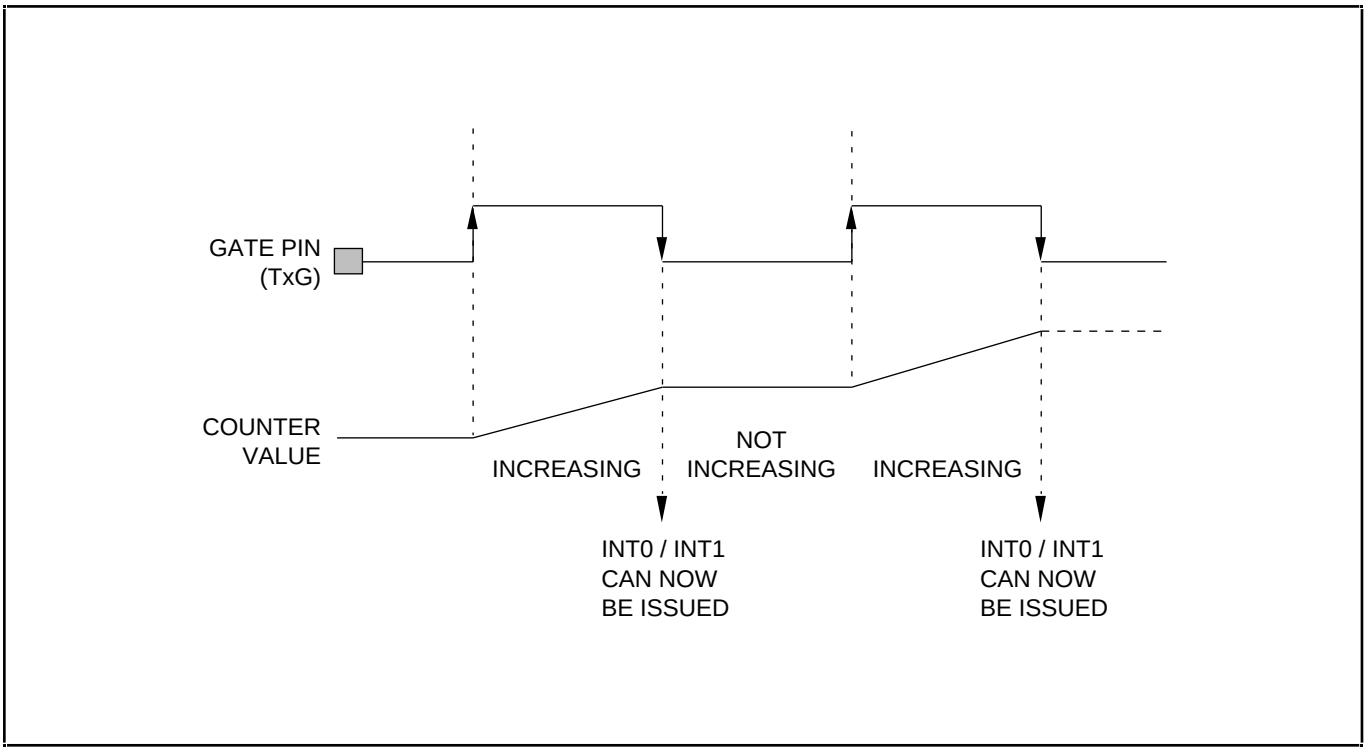


Figure 11–5. Count Value Incrementing With Gate Function Enabled

Timer Module 1, Mode 0 Operation

Mode 0 operation is functionally identical for timers C and D. In mode 0, the corresponding count registers (TxH, TxL) are configured as a 13-bit timer/counter with a divide-by-32 prescaler. A reset automatically selects mode 0.

Eight bits of the high-byte count register (TxH) are used as the 8-bit counter; the five lower bits of the low-byte count register (TxL) are used as the 5-bit counter. The value of the upper three bits of TxL is undetermined and should be ignored. Both registers are read-write programmable.

The value of the 8-bit counter (TxH) is undetermined after a reset. Enabling a timer by setting its run bit to "1" does not automatically clear the TxH count value; you must clear it by software.

When the TxH count overflows, the timer's interrupt pending flag TxIP (T1CON bits 4 and 5) is set to "1". The interrupt pending flags can be polled by software to control timer C and D interrupt processing.

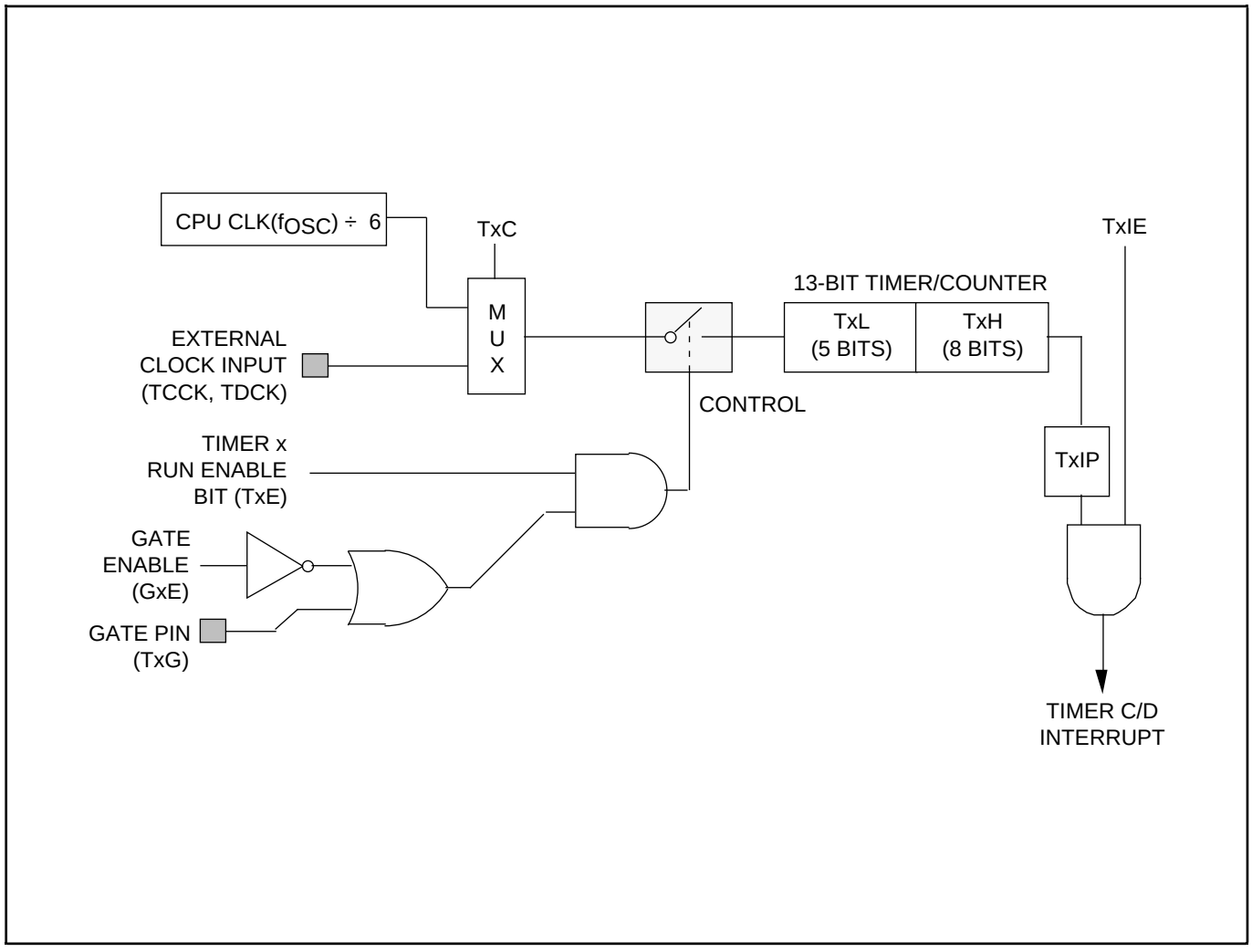


Figure 11–6. Timer Module 1 Mode 0 Function Diagram

Timer Module 1, Mode 1 Operation

Mode 1 is the same as mode 0, except that the 8-bit timer C and D counters (TxH and TxL) operate together as a 16-bit event counter.

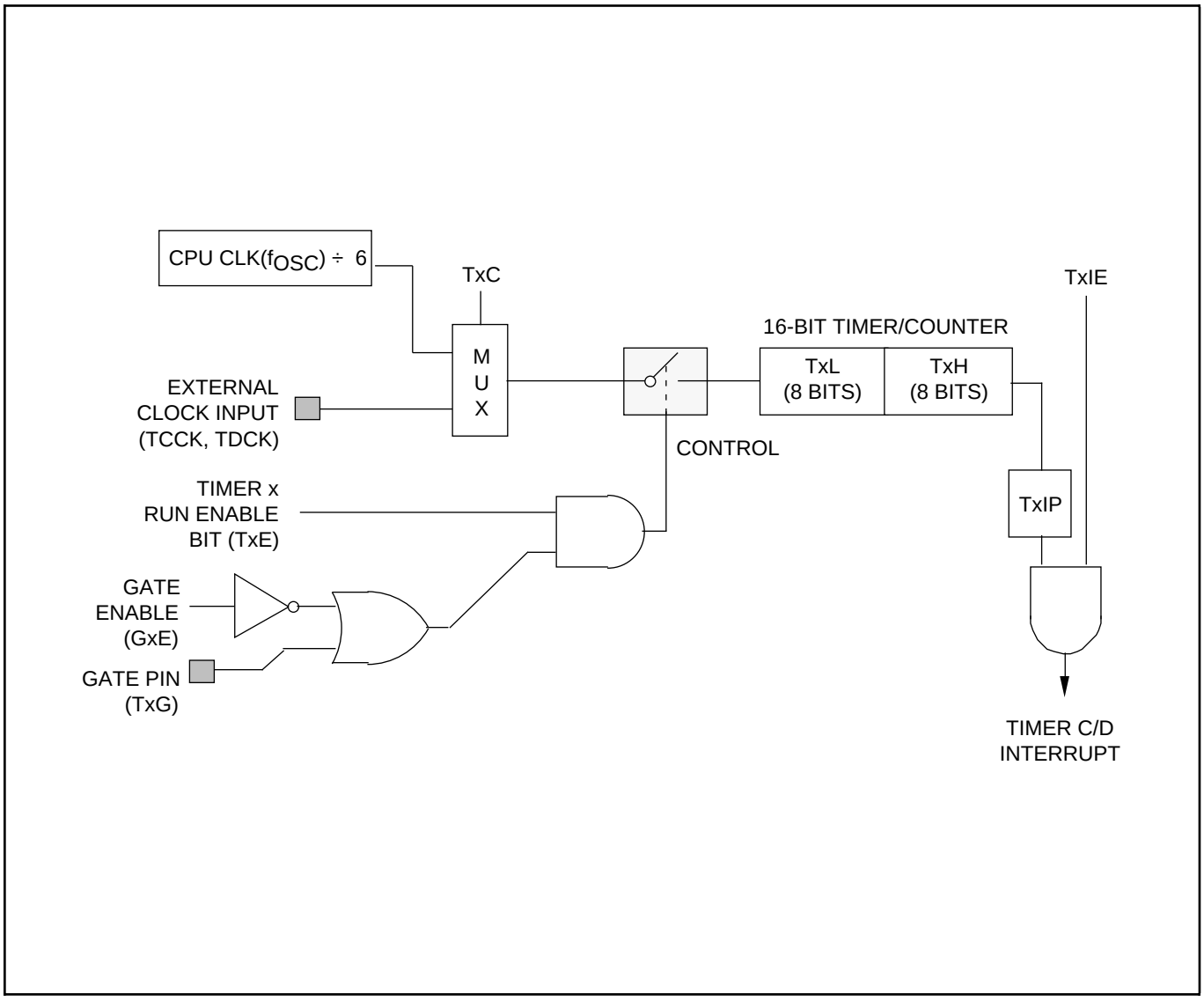


Figure 11–7. Timer Module 1 Mode 1 Function Diagram

Timer Module 1, Mode 2 Operation

Mode 2 establishes the timer registers as one 8-bit counter (TxL) which is automatically reloaded with an 8-bit value stored in the TxH register when the TxL counter overflows.

When the counter overflow occurs, the corresponding interrupt pending flag (TxIP) in the T1CON register is set to "1", the counter is reloaded with the value stored in TxH, and counting resumes. The reload value that is stored in TxH must be preset by software. The reload value is unchanged by the reload operation.

Assuming that the appropriate interrupt enable bit (TxIE) in the T1CON register is set, the timer's interrupt pending flag can then be polled to generate the timer C or D interrupt request.

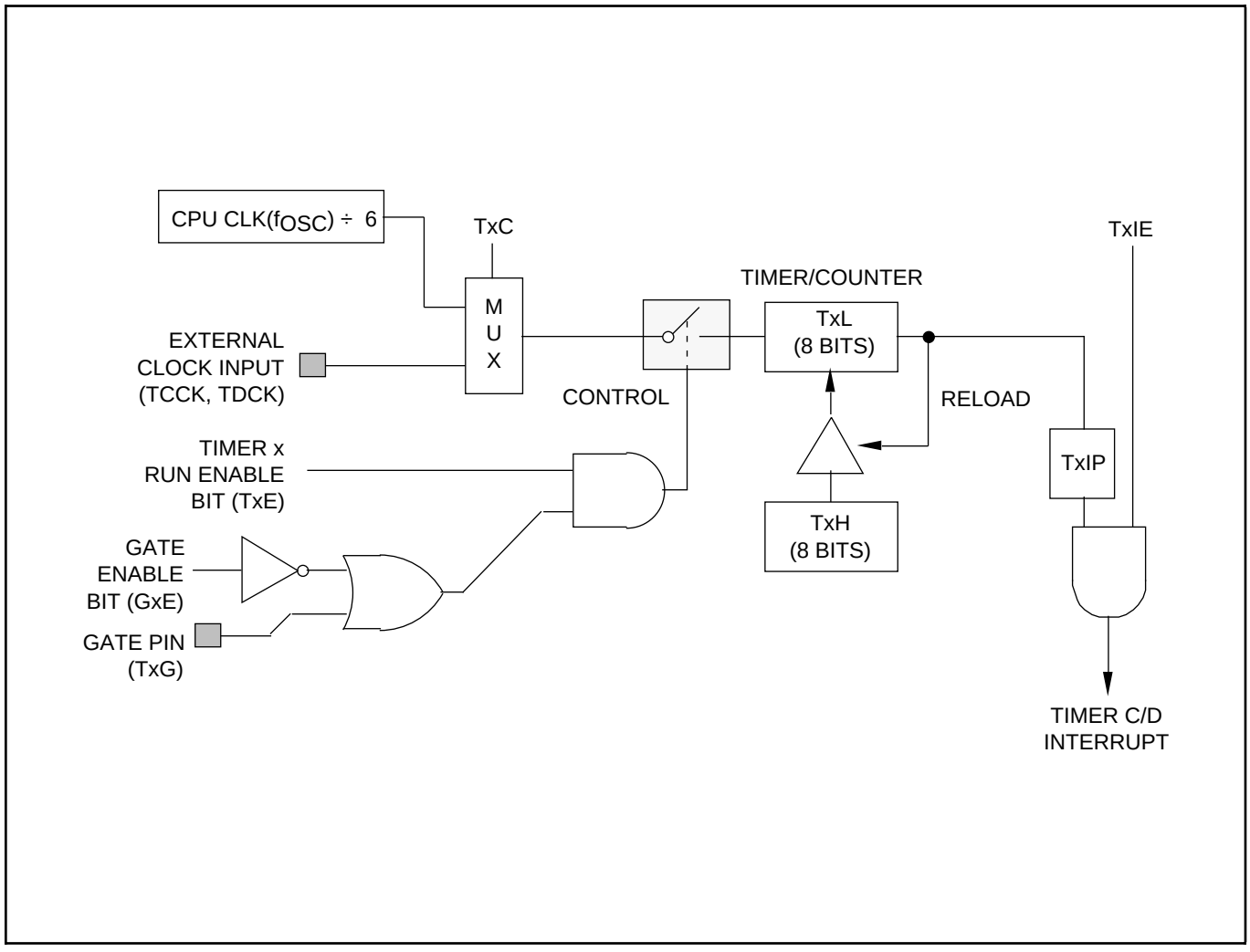


Figure 11–8. Timer Module 1 Mode 2 Function Diagram

Timer Module 1, Mode 3 Operation

Unlike modes 0, 1, and 2, in mode 3 timers C and D behave differently from each other: timer C functions as two separate 8-bit counters, while timer D continues operating, but with no interrupt source.

To select mode 3, the TCM1/M0 bit-pairs in the T1MOD register are set to '11B'.This setting establishes the timer C count registers, TCL and TCH, as two separate counters with one difference: TCL is an 8-bit timer/counter and TCH is an 8-bit timer only (that is, it has no external input). To program TCL for mode 3 operation, you must also write the appropriate values to the timer C mode control bits in the T1MOD register.

With TCH locked into a timer function of counting internal clocks, it assumes control of the timer D interrupt control bits. In effect, the "timer D interrupt" is generated by the timer C counter. When timer C is set to operate in mode 3, timer D can be turned on and off by switching it in and out of its own mode 3 operating status, or it can be used by the serial I/O port as a baud rate generator. In fact, timer D can be used during mode 3 operation for any application which does not require that it generate an interrupt.

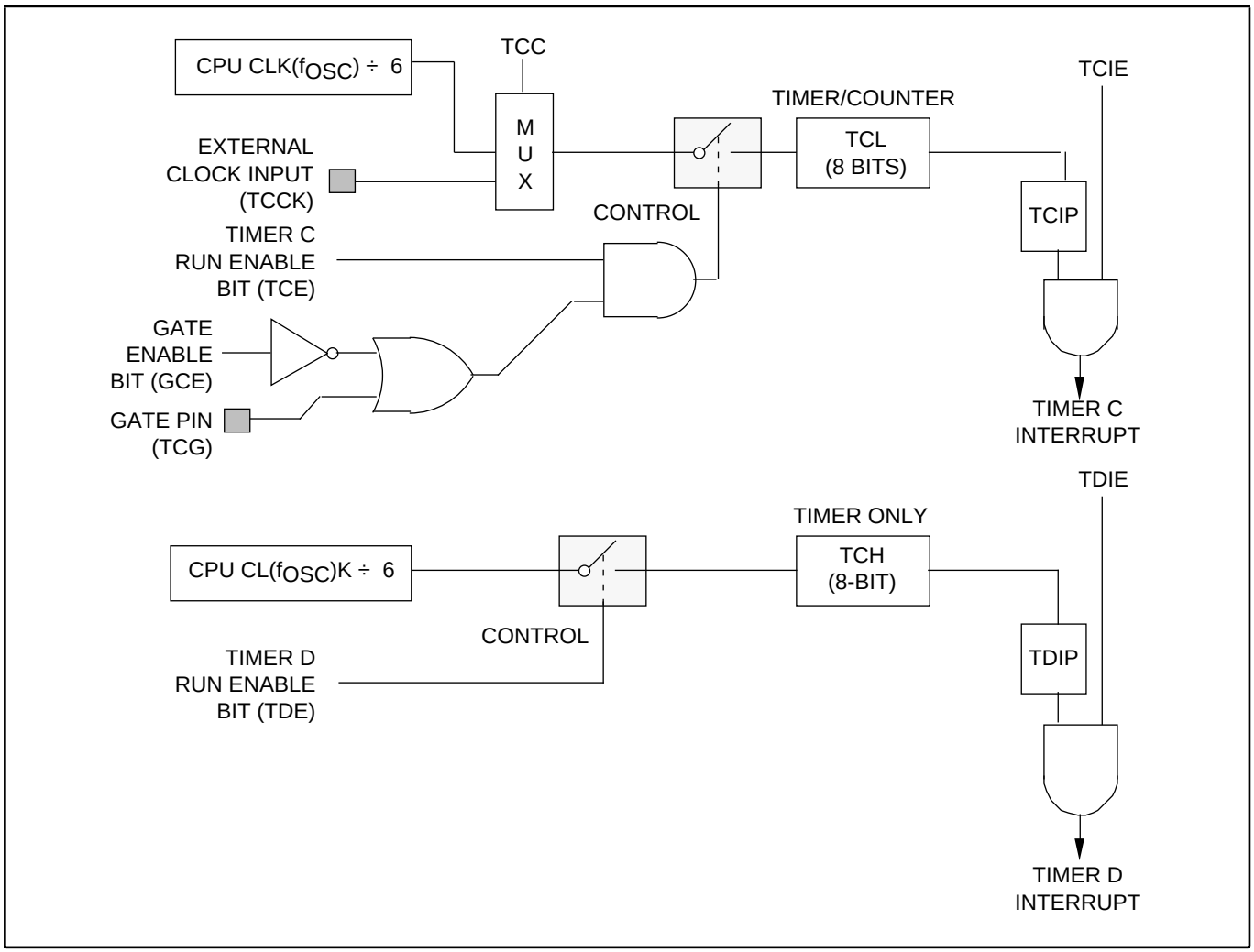


Figure 11–9. Timer Module 1 Mode 3 Function Diagram

☞ Programming Tip — Timer Module 1, Operating Mode 0

This example shows how to program timer module 1 (timers C and D) to operate in 13-bit timer/counter mode (that is, in mode 0). The parameters of the sample program are as follows:

— Only timer C is used for this example; timer D is disabled
— CPU clock frequency = 6 MHz
— Timer C input clock = 1 MHz
— Timer C interrupts occur in 2-millisecond intervals
— Each timer C interrupt toggles the P0.0 pin

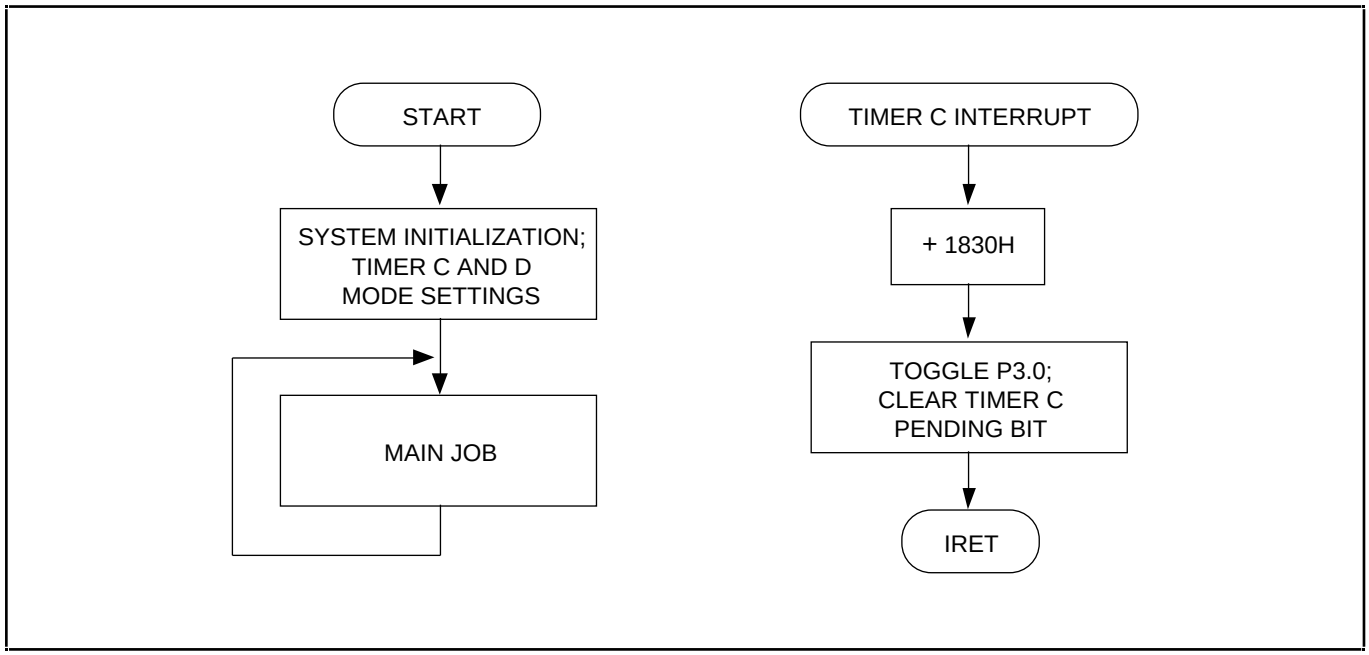


Figure 11–10. Timer Module 1 Mode 0 Programming Tip Flowchart

☞ Programming Tip — Timer Module 1, Operating Mode 0 (Continued)

```
START   DI                          ;  Disable interrupts
        •
        •                           ;  System initialization settings
        •
        LD      P2CON,#8AH          ;  Port 2.7 output push-pull mode select
        LD      TCH,#0C1H           ;  07D0H counter value is equal to 2 ms
                                    ;  2000H – 07D0H  =  1830H
        LD      TCL,#10H            ;  (TCH, TCL)  ←  1830H
                                    ;  The low 5 bits of 1830H are '10H'
                                    ;  Bits 5–12 of 1830H are 'C1H'
        LD      T1MOD,#30H          ;  Timer D disable, timer C 13-bit timer mode select,
                                    ;  timer clock = CPU clock ÷ 6
        LD      T1CON,#35H          ;  Timer C interrupt enable; timer C run enable
        EI                          ;  Enable interrupts
        •
        •
        •
MAIN    NOP
        •
        •
        •
        CALL    JOB                 ;  Run other job
        •
        •
        •
        JP      T,MAIN
        •
        •
        •
```

Then, the timer C overflow interrupt service routine (TC_INT):

```
TC_INT:

        ADD     TCH,#0C1H           ;  TC  ←  TC + 1830H
        ADD     TCL,#10H            ;  TCL is just a 5-bit counter
        ADC     TCH,#00H            ;
        XOR     P2,#80H             ;  Toggle the P2.7 pin
        LD      T1CON,#35H          ;  Clear the timer C pending bit
        IRET                        ;  Return from interrupt
```

☞ Programming Tip — Timer Module 1, Operating Mode 1

This example shows how to program timer module 1 (timers C and D) to operate in 16-bit timer/counter mode (that is, mode 1). The parameters of the sample program are as follows:

— CPU clock frequency = 6 MHz

— Clock input pulse at the TCCK pin is an unknown frequency

— Clock input pulse at the timer C gate pin (TCG) is equal to 62.5 Hz (50% duty)

— Interrupt INT2 occurs with each falling edge at the TCG pin

Program Function Description

Timer C operates as a frequency counter. An unknown frequency is being input through the timer C clock input pin (TCCK). Let's suppose, however, that the frequency range at the TCCK pin is less than 667 kHz.

Using the reference pulse at the timer C gate input pin (TCG), we can count the unknown clock pulses at the TCCK pin during an 8-ms interval (1/62.5 Hz ÷ 2). The timer C count value is saved into B_TC0 and B_TC1 in hexadecimal format. The values in B_TC0 and B_TC1 are the actual frequency value in kHz.

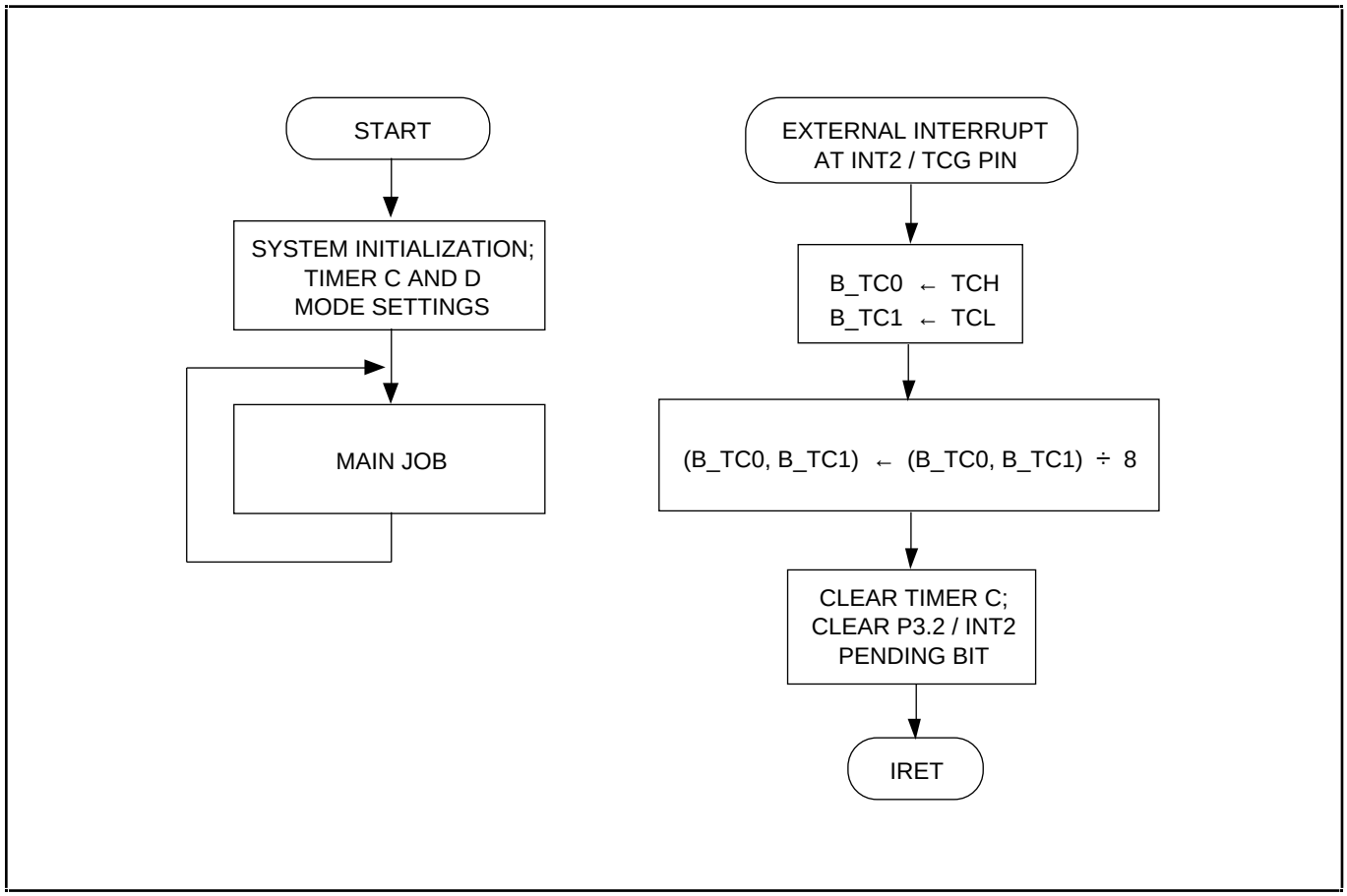


Figure 11–11. Timer Module 1 Mode 1 Programming Tip Flowchart

☞ **Programming Tip — Timer Module 1, Operating Mode 1 (Continued)**

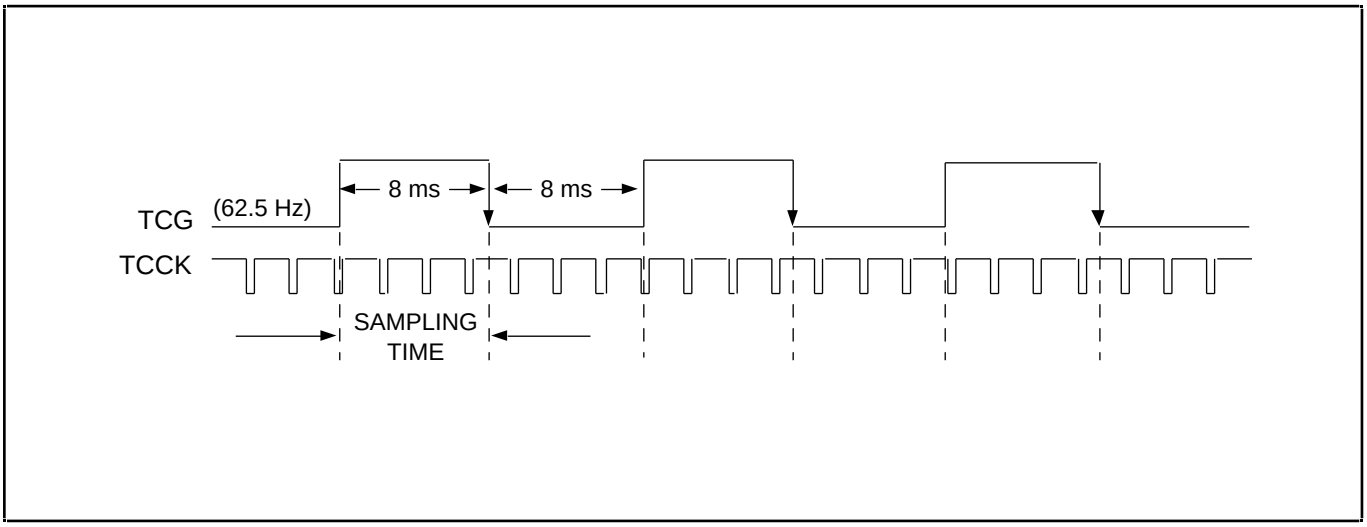


Figure 11–12. Timer Module 1 Mode 1 Timing Diagram

```
B_TC0     EQU     00H                 ;  Timer C data buffer register
B_TC1     EQU     01H
START     DI                          ;  Disable interrupts
          •
          •                           ;  System initialization settings
          •
          LD      P3CONL,#00H         ;  TCCK input mode select
                                      ;  TCG (INT2) falling-edge input select
          LD      P3INT,#04H          ;  P3.2/TCG interrupt enable
          LD      P3PND,#0FH          ;  Clear pending bit
          LDW     TCH,#0000H          ;  Initialize timer C counter
          LD      T1MOD,#3DH          ;  Disable timer D, enable TCG input,
                                      ;  timer clock  =  external clock input select,
                                      ;  16-bit counter mode select
          LD      T1CON,#31H          ;  Disable timer C and D interrupt,
                                      ;  timer C run enable
          EI                          ;  Enable interrupts
          •
          •
          •
MAIN      NOP
          •
          •
          •
          CALL    JOB                 ;  Run other job
          •
          •
          •
          JP      T,MAIN
```

☞ Programming Tip — Timer Module 1, Operating Mode 1 (Concluded)

The external interrupt occurs at P3.2 /TCG in 16-ms intervals (falling edges):

EXTINT_TCG:

```
LDW     B_TC0,TCH       ;  Count value detect (2 bytes)
RCF                     ;
RRC     B_TC0           ;
RRC     B_TC1           ;  (B_TC0, B_TC1) ← (B_TC0, B_TC1) ÷ 2
RCF                     ;
RRC     B_TC0           ;  (B_TC0, B_TC1) ← (B_TC0, B_TC1) ÷ 2
RRC     B_TC1           ;
RCF                     ;
RRC     B_TC0           ;  (B_TC0, B_TC1) ← (B_TC0, B_TC1) ÷ 2
RRC     B_TC1           ;  The value in B_TC0, B_TC1 indicates the clock
                        ;  frequency at the TCCK pin (in kHz)
LDW     TCH,#0000H      ;
LD      P3PND,#04H      ;  Clear the P3.2 (INT2) pending bit
IRET
```

☞ Programming Tip — Timer Module 1, Operating Mode 2

This example shows how to program timer module 1 (timers C and D) to operate in auto-reload timer/counter mode (that is, in mode 2). The parameters of the sample program are as follows:

— CPU clock frequency = 6 MHz
— Clock input pulse at the TCCK pin is 60 Hz (square wave)
— Timer C operates in auto-reload mode using external clock input
— Timer C interrupt occurs in 0.5-second intervals
— The level at the P3.4 pin toggles whenever an interrupt occurs

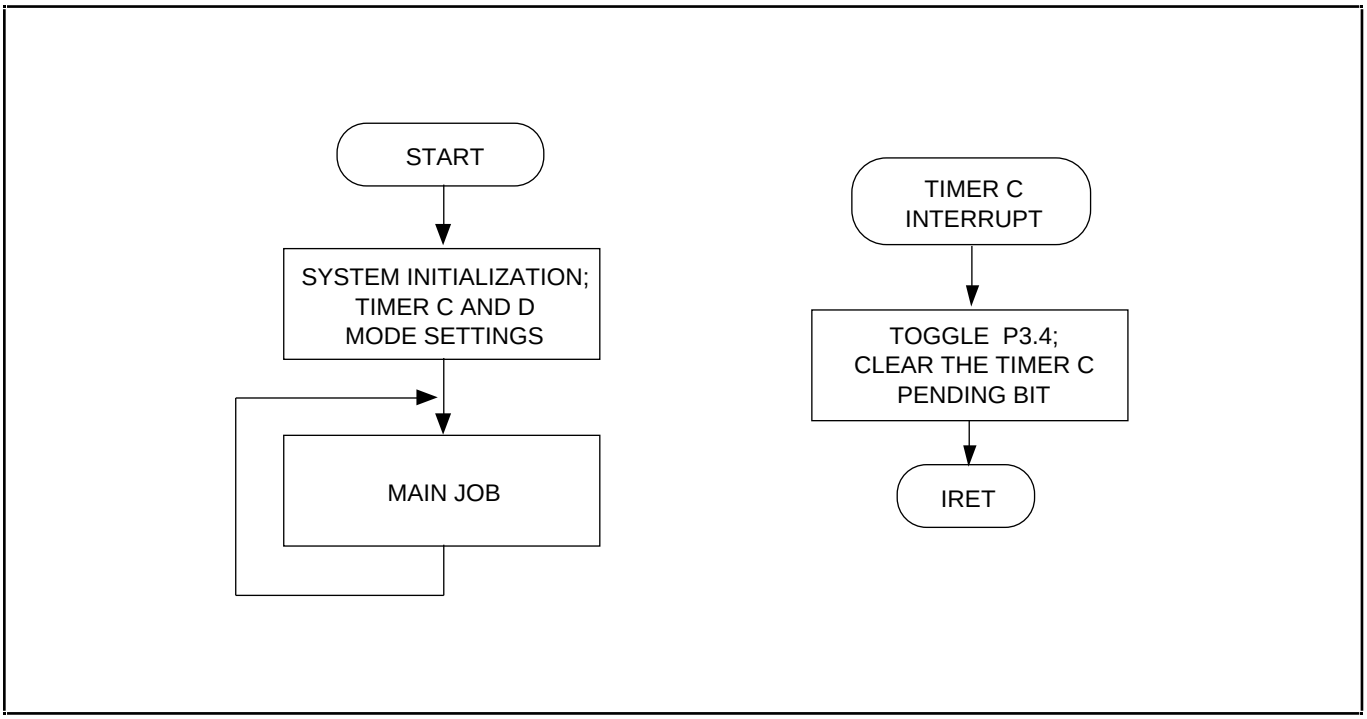


Figure 11–13. Timer Module 1 Mode 2 Programming Tip Flowchart

☞ **Programming Tip — Timer Module 1, Operating Mode 2 (Continued)**

```
START   DI                          ;  Disable interrupts
        •
        •                           ;  System initialization settings
        •
        LD      P3CONH,#11H         ;  P3.4–P3.7 set to output, push-pull mode
        LD      P3CONL,#00H         ;  P3.0 (TCCK), P3.1 (TDCK) clock input select
        LD      TCL,#(0FFH–1EH)     ;  Initial value for timer C low byte
        LD      TCH,#(0FFH–1EH)     ;  0.5 sec = 1/60 × 30 (decimal) = 1EH
                                    ;  1EH is the automatically reload value
        LD      T1MOD,#36H          ;  Disable timer D
                                    ;  Select timer C auto-reload operating mode
                                    ;  Select external clock source for timer C
                                    ;  Disable timer C gate function
        LD      T1CON,#35H          ;  Enable timer C interrupt
                                    ;  Timer C run enable
        EI                          ;  Enable interrupts
        •
        •
        •
MAIN    NOP
        •
        •
        •
        CALL    JOB                 ;  Run other job
        •
        •
        •
        JP      T,MAIN
        •
        •
        •

TMRC_INT:

        XOR     P3,#10H             ;  P3.4 toggle
        LD      T1CON,#35H          ;  Clear timer C pending bit
        IRET
```

☞ Programming Tip — Timer Module 1, Operating Mode 3

This example shows how to program timer module 1 to operate as two 8-bit timer/counters (that is, in mode 3). The parameters of the sample program are as follows:

— Main oscillator frequency = 11.0592 MHz (CPU clock = 11.0592 MHz)

— Clock input pulse at the TCCK pin is 60 Hz (square wave)

— Timer C operates in mode 3 (as two 8-bit timer/counters)

— Timer D operates in auto-reload mode with no interrupt function

Program Function Description

Timer C's low-byte count register (TCL) counts interrupts which occur every 0.5 seconds. These interrupts are generated by external 60 Hz clock input through the TCCK pin. The high-byte count register (TCH) counts interrupts generated at 278-µs intervals.

The timer D interrupt structure is used to the TCH count register overflow at a frequency of CPU clock divided by 6. Timer D serves as a baud rate generator for the UART module.

The baud rate is 9600 BPS. Timer D does not generate an interrupt. The state of the P3.0 and P3.1 pins toggles each time an interrupt service routine is initiated.

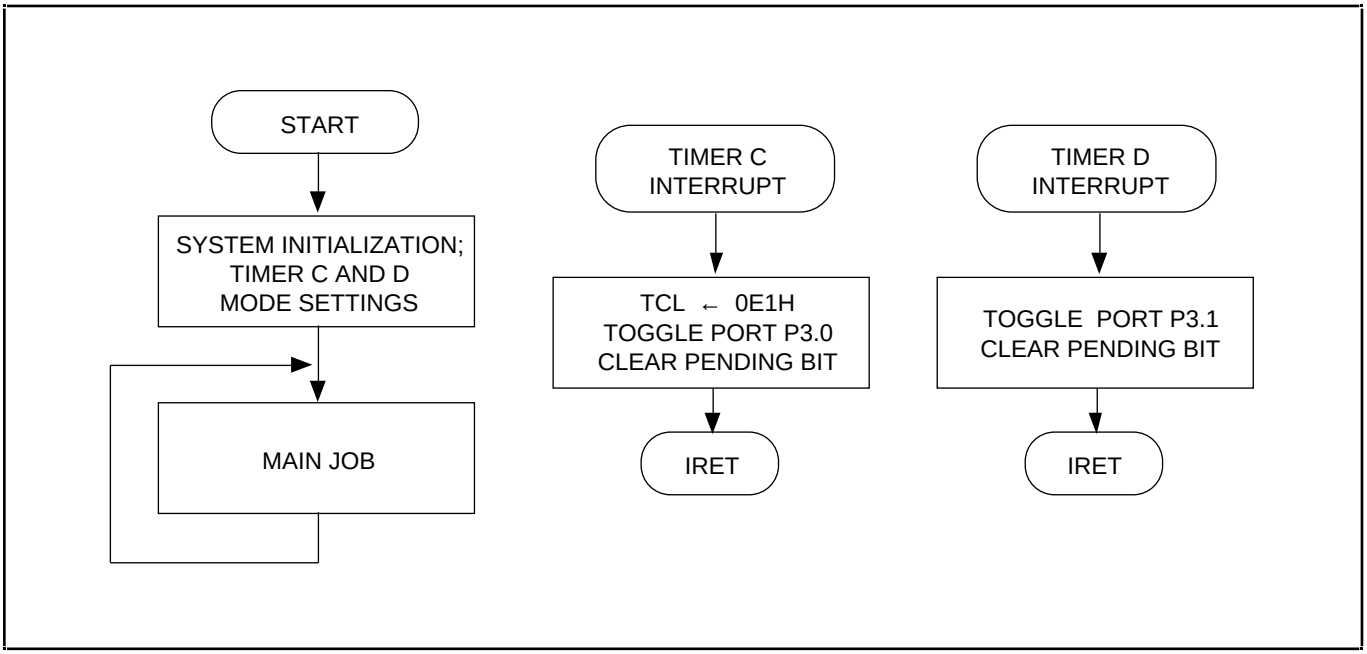


Figure 11–14. Timer Module 1 Mode 3 Programming Tip Flowchart

☞ **Programming Tip — Timer Module 1, Operating Mode 3 (Continued)**

```
START   DI                          ;  Disable interrupts
        •
        •                           ;  System initialization settings
        •
        LD      P2CON,#0A0H         ;  Select P2.6 and P2.7 output mode
        LD      P3CONL,#00H         ;  Select TCCK pin input mode
        LD      TCL,#0E1H           ;  External 60 Hz clock gives a 0.5 second interval value
        LD      TCH,#00H            ;  CPU clock divided by 6 gives a 278 µs value
        LD      TDH,0FAH            ;  Baud rate value of 9600 BPS with divided-by-6 CPU
clock
        LD      TDL,#0FAH           ;
        LD      T1MOD,#27H          ;  Timer D auto-reload mode (CPU clock /6)
                                    ;  Disable gate function
                                    ;  Select timer C two 8-bit timer/counter mode
                                    ;  Low byte clock source  =  60 Hz external
                                    ;  High byte clock source  =  CPU clock /6
                                    ;  Disable gate function
        LD      T1CON,#3FH          ;  Enable timer C and D interrupts
                                    ;  Enable baud rate generator function
                                    ;  Timer C and D run enable
        EI                          ;  Enable interrupts
        •
        •
        •
MAIN    NOP
        •
        •
        •
        CALL    JOB                 ;  Run other job
        •
        •
        •
        JP      T,MAIN
```

The timer C interrupt is generated by a timer C low byte counter (TCL) overflow (0.5-second intervals):

```
TMRC_INT:
        ADD     TCL,#0E1H           ;  TCL ← TCL + 0E1H (0.5 seconds)
        XOR     P2,#80H             ;  P2.7 toggle
        LD      T1CON,#1FH          ;  Clear pending bit (bit 4)
        IRET
```

The timer D interrupt is generated by a timer C high byte counter (TCH) overflow. The full count range of 00H–0FFH occurs once every 278 µs, resulting in a 278-µs interrupt interval:

```
TMRD_INT:
        XOR     P2,#40H             ;  P2.6 toggle
        LD      T1CON,#2FH          ;  Clear pending bit (bit 5)
        IRET
```

NOTE

12 Serial Port (UART)

OVERVIEW

The KS88C4400 has a full-duplex serial port with programmable operating modes: There is one synchronous mode and three UART (Universal Asynchronous Receiver/Transmitter) modes:

— Serial I/O with baud rate of CPU clock /6
— 8-bit UART mode; variable baud rate
— 9-bit UART mode; CPU clock /32 or /16
— 9-bit UART mode, variable baud rate

Serial port receive and transmit buffers are both accessed via the shift register, SIO (R233, E9H). Writing to the shift register loads the transmit buffer; reading the shift register accesses a physically separate receive buffer.

The serial port is receive-buffered. Using a receive data buffer, reception of the next byte can commence before the previously received byte has been read from the receive register. However, if the first byte has not been read by the time the next byte has been completely received, one of the bytes will be lost.

In all operating modes, transmission is initiated when any instruction addresses the shift register SIO (R233, E9H) as its destination register. In mode 0, reception of serial data is initiated when the receive interrupt pending bit (RIP) in the SIOPND register is cleared to "0" and the receive enable bit (RE, SIOCON.4) is set to "1". In modes 1, 2, and 3, reception is initiated when the incoming start bit ("0") is received and the receive enable (RE) bit is "1".

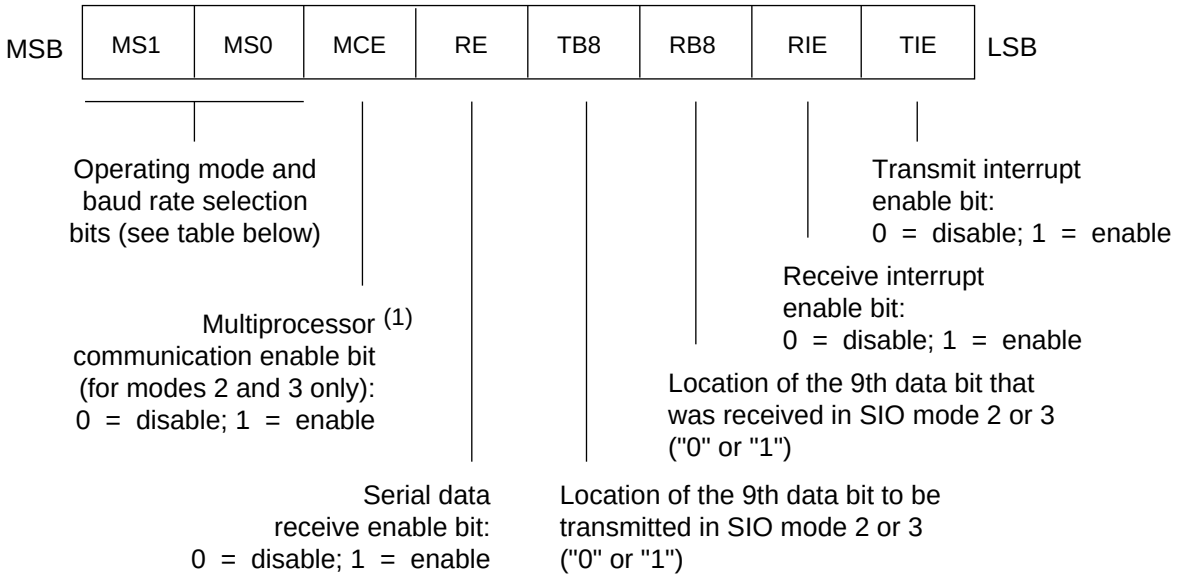
Serial Port Control Register (SIOCON)

The control register for the serial port is called SIOCON (R234, EAH). It has the following control functions:

— Operating mode selection
— 9th data bit location for transmit and receive operations (TB8, RB8)
— Multiprocessor communication and interrupt control

These SIOCON control functions are described in detail in Figure 12–1 below.

SERIAL PORT CONTROL REGISTER (SIOCON)
R234, EAH, Set 1, Bank 0, R/W



MS1	MS0	Mode	Description (2)	Baud Rate
0	0	0	Shift register	CPU clock /6
0	1	1	8-bit UART	variable
1	0	2	9-bit UART	CPU clock /32 or /16
1	1	3	9-bit UART	variable

NOTES:

1. In mode 2 or 3, if the MCE bit is set to "1" then the receive interrupt will not be activated if the received 9th data bit is "0". In mode 1, if MCE = "1" then the receive interrupt will not be activated if a valid stop bit was not received. In mode 0, the MCE bit should be "0".
2. The descriptions for 8-bit and 9-bit UART mode do not include start and stop bits for serial data receive and transmit.

Figure 12–1. Serial Port Control Register (SIOCON)

Serial Port Interrupt Pending Register (SIOPND)

The serial I/O interrupt pending register SIOPND (R235, EBH, set 1, bank 0) contains the serial data transmit interrupt pending bit (TIP) and the receive interrupt pending bit (RIP) in register positions SIOPND.0 and SIOPND.1, respectively.

In mode 0, the receive interrupt pending flag RIP is set to "1" when the 8th receive data bit has been shifted. In mode 1, 2, and 3, the RIP bit is set to "1" at the halfway point of the stop bit's shift time. When the CPU has acknowledged the receive interrupt pending condition, the RIP flag must then be cleared by software.

In mode 0, the transmit interrupt pending flag TIP is set when the 8th transmit data bit has been shifted. In mode 1, 2, or 3, the RIP bit is set at the start of the stop bit. When the CPU has acknowledged the transmit interrupt pending condition, the TIP flag must then be cleared by software.

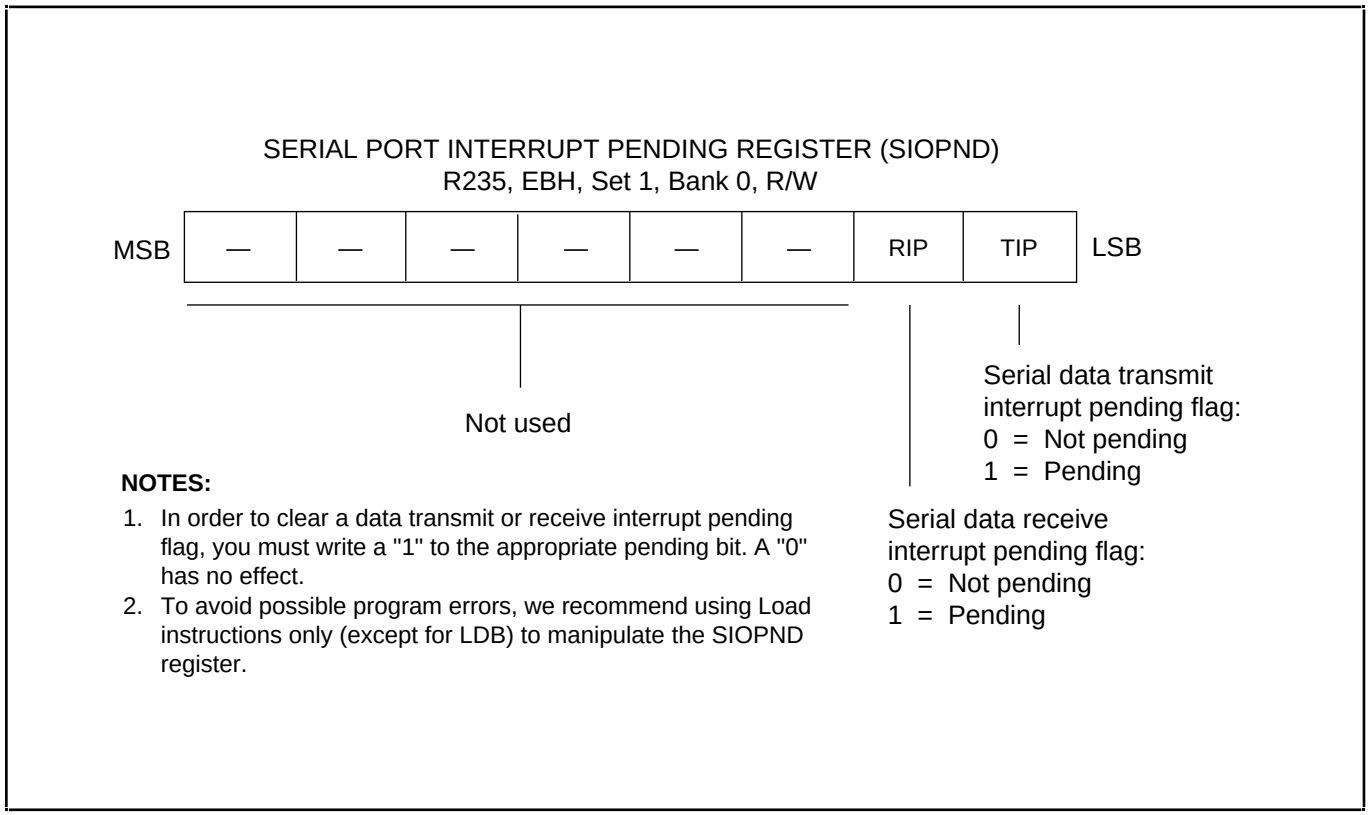


Figure 12–2. Serial Port Interrupt Pending Register (SIOPND)

Serial Port Mode 0 Function Description

In mode 0, serial data enters and exits through the RxD pin (pin 20); the TxD pin (pin 21) outputs the shift clock. Data are transmitted or received in 8-bit units only. The LSB of the 8-bit value is transmitted (or received) first. The baud rate for mode 0 is equal to the CPU clock frequency divided by 6.

Mode 0 Transmit Procedure

1. Select mode 0 by setting SIOCON bits 6 and 7 to '00B'.
2. Write transmission data to the shift register SIO (E9H) to initiate the transmit operation.

Mode 0 Receive Procedure

1. Select mode 0 (shift register; CPU clock /6) by setting SIOCON bits 6 and 7 to '00B'.
2. Clear the receive interrupt pending bit (RIP, SIOPND.1) by loading a "1".
3. Set the serial data receive enable bit (RE, SIOCON.4) to "1".
4. The shift clock will now be output to the TxD pin (pin 21) and will read the data at the RxD pin (pin 20). Interrupt requests are generated if the TIE bit in the SIOCON register is "1".

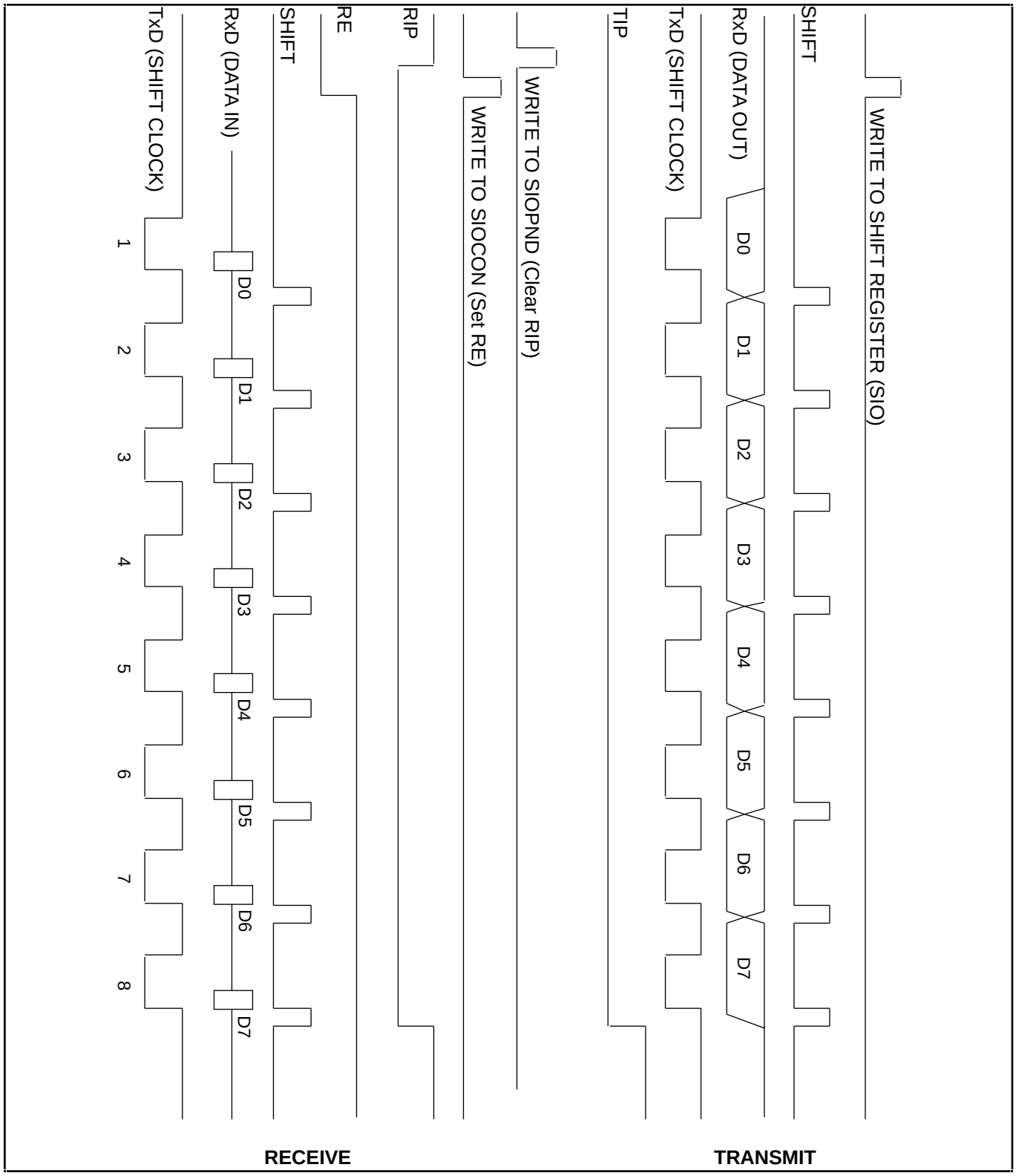


Figure 12–3. Timing Diagram for Serial Port Mode 0 Operation

Serial Port Mode 1 Function Description

In mode 1, a total of 10 bits are transmitted (through the TxD pin) or received (through the RxD pin). Each data frame has three components:

— Start bit ("0")

— 8 data bits (LSB first)

— Stop bit ("1")

When receiving, the stop bit is written to the RB8 bit in the SIOCON register. The baud rate for mode 1 is variable.

Mode 1 Transmit Procedure

1. Select the baud rate generated by timer/counter D using the timer module 1 control register T1CON. The baud select bit (BSEL, T1CON.7) offer a choice of normal ("0" or double ("1")baud rate generation for the UART module.
2. Select mode 1 (8-bit UART) by setting SIOCON bits 6 and 7 to '01B'.
3. Write transmission data to the shift register SIO (E9H). (The start and stop bits will be generated automatically by hardware.)

Mode 1 Receive Procedure

1. Select the baud rate to be generated by timer/counter D.
2. Select mode 1 and set the RE (Receive Enable) bit in the SIOCON register to "1".
3. The start bit low ("0") condition at the RxD pin will cause the UART module to initiate the serial data receive operation.

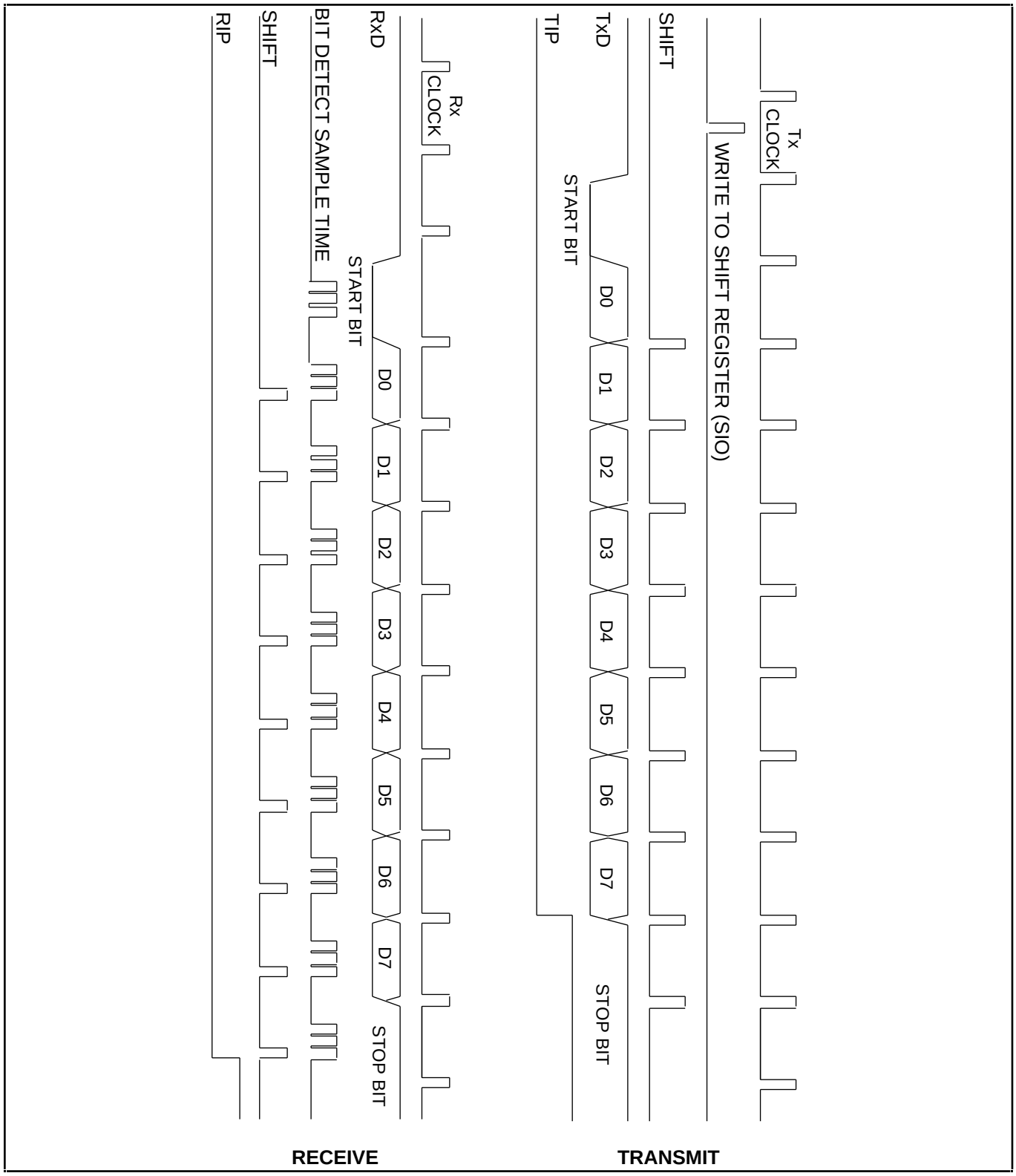


Figure 12–4. Timing Diagram for Serial Port Mode 1 Operation

Serial Port Mode 2 Function Description

In mode 2, 11 bits are transmitted (through the TxD pin) or received (through the RxD pin). Each data frame has four components:

— Start bit ("0")

— 8 data bits (LSB first)

— Programmable 9th data bit

— Stop bit ("1")

The 9th data bit to be transmitted can be assigned a value of "0" or "1" by writing the TB8 bit (bit 3) in the SIOCON register. When receiving, the 9th data bit that is received is written to the RB8 bit (SIOCON.2) while the stop bit is ignored. The baud rate for mode 2 is programmable to either 1/16 or 1/32 of the CPU clock frequency.

Mode 2 Transmit Procedure

1. Select mode 2 (9-bit UART) by setting SIOCON bits 6 and 7 to '10B'. Also, select the 9th data bit to be transmitted by writing SIOCON bit 3 (TB8) to "0" or "1".
2. Select the baud rate by setting the BSEL bit in the T1CON register to "0" for normal baud or to "1" for double baud rate generation.
3. Write transmission data to the shift register, SIO (E9H) to initiate the transmit operation.

Mode 2 Receive Procedure

1. Select the baud rate by setting or clearing the BSEL bit in the T1CON register.
2. Select mode 2 and set the RE (Receive Enable) bit in the SIOCON register to "1".
3. The receive operation will be initiated when the signal at the RxD pin goes to low level.

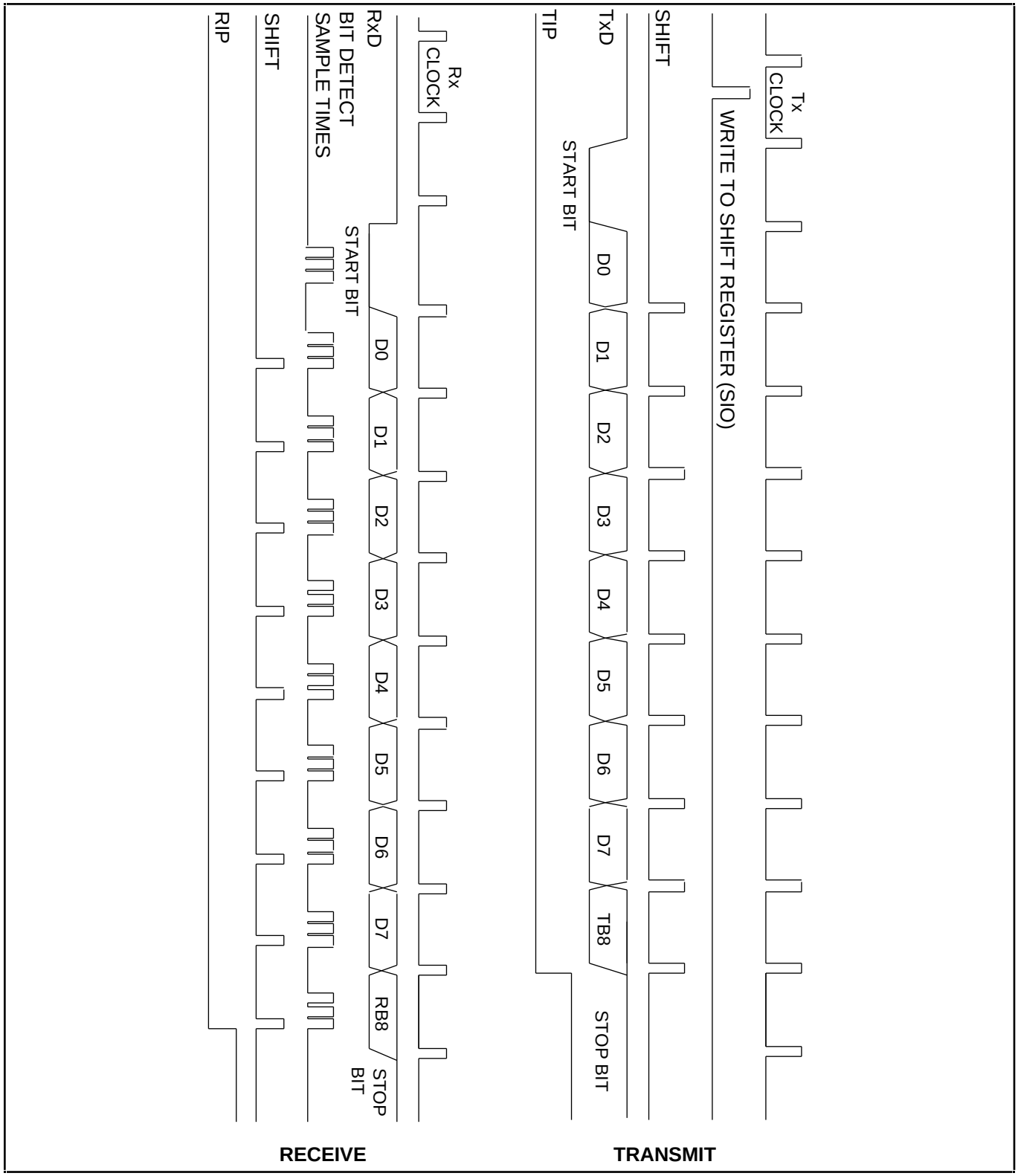


Figure 12–5. Timing Diagram for Serial Port Mode 2 Operation

Serial Port Mode 3 Function Description

In mode 3, 11 bits are transmitted (through the TxD pin) or received (through the RxD pin). Mode 3 is identical to mode 2 in all respects except for baud rate, which is variable. Each data frame has four components:

— Start bit ("0")

— 8 data bits (LSB first)

— Programmable 9th data bit

— Stop bit ("1")

Mode 3 Transmit Procedure

1. Select the baud rate by setting the BSEL bit in the T1CON register to "0" for normal baud or to "1" for double baud rate generation, and then enable timer D by setting the TDE bit in the T1CON register.
2. Select mode 3 operation (9-bit UART) by setting SIOCON bits 6 and 7 to '11B'. Also, select the 9th data bit to be transmitted by writing SIOCON bit 3 (TB8) to "0" or "1".
3. Write transmission data to the shift register, SIO (E9H) to initiate the transmit operation.

Mode 3 Receive Procedure

1. Select the baud rate to be generated by timer/counter D by setting or clearing the BSEL bit in the T1CON register.
2. Select mode 3 and set the RE (Receive Enable) bit in the SIOCON register to "1".
3. The receive operation will be initiated when the signal at the RxD pin goes to low level.

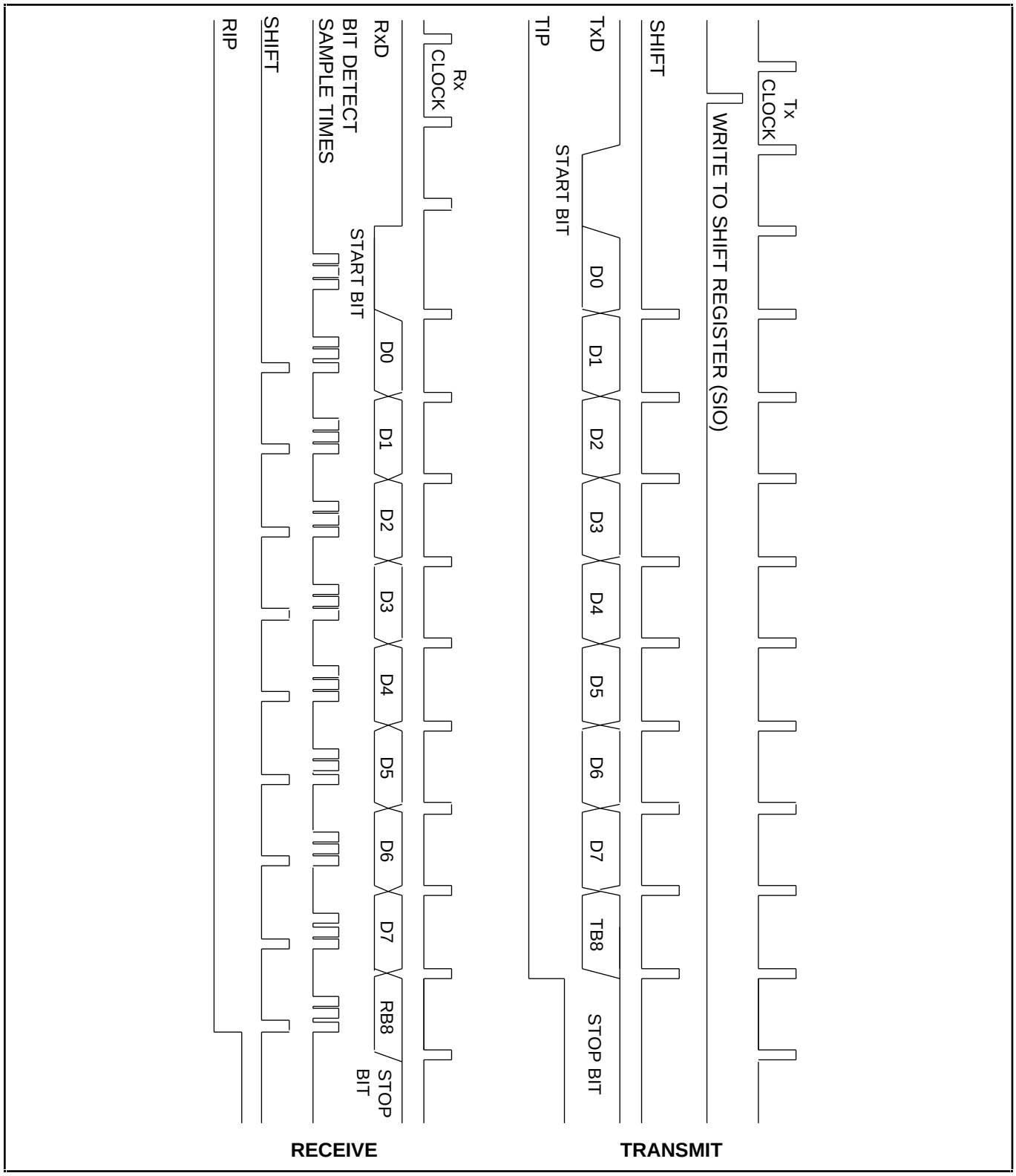


Figure 12–6. Timing Diagram for Serial Port Mode 3 Operation

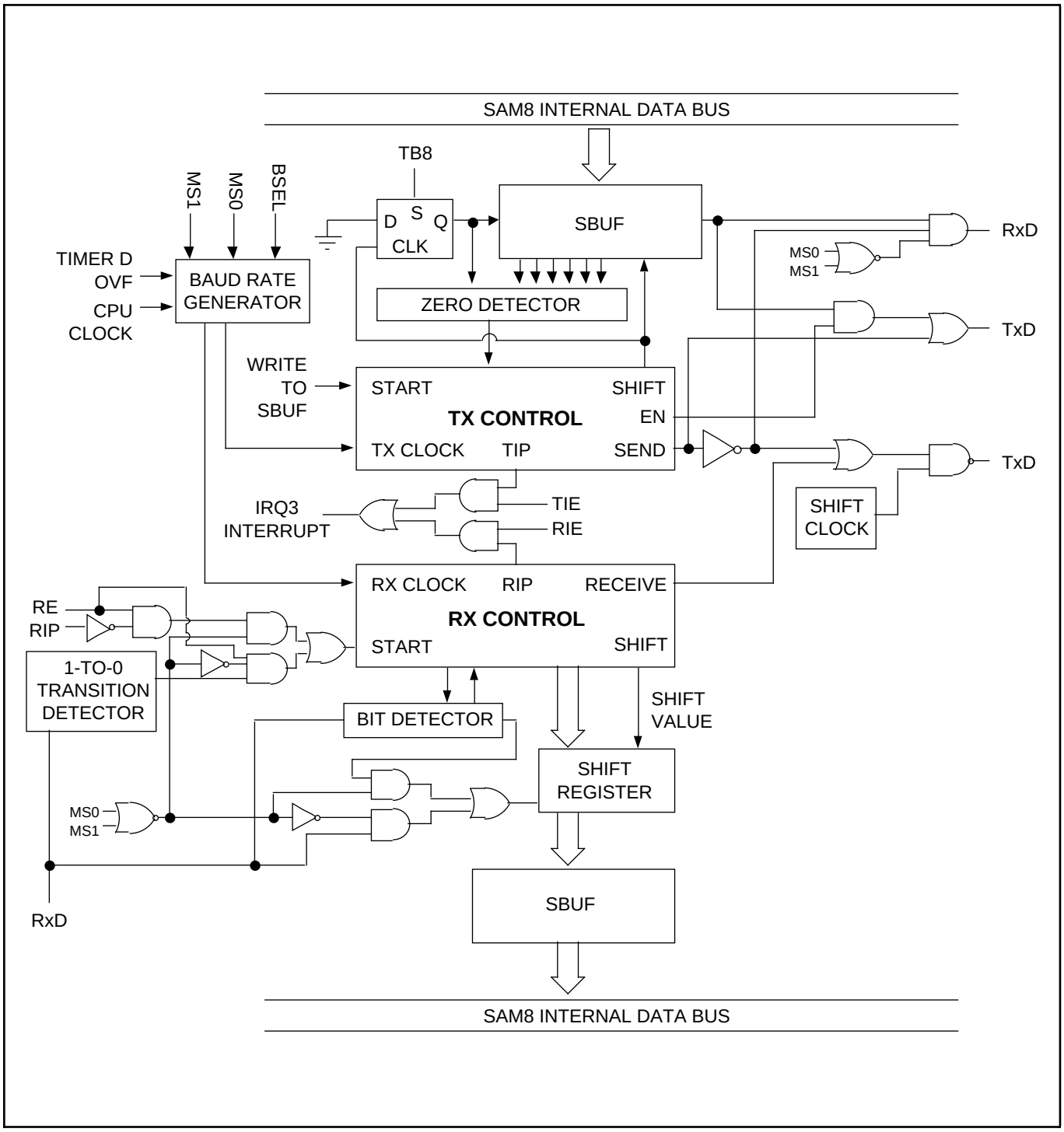


Figure 12–7. Serial Port (UART) Function Diagram

Baud Rate Calculations

Mode 0 Baud Rate Calculation

The baud rate in mode 0 is fixed at the CPU clock frequency (same as f_{OSC}) divided by 6:

$$\text{Mode 0 baud rate} = \frac{\text{CPU clock}}{6}$$

Mode 2 Baud Rate Calculation

The mode 2 baud rate depends on the value of the double baud rate select bit, BSEL (T1CON.7). If BSEL = "0" (its default value after a reset), the mode 2 baud rate is 1/32 of the CPU clock frequency. If BSEL = "1", the baud rate is 1/16 of the CPU clock frequency.

$$\text{Mode 2 baud rate} = \frac{2^{\text{BSEL}}}{32} \times \text{CPU clock}$$

Modes 1 and 3 Baud Rate Calculation

When timer/counter D is used as the baud rate generator for modes 1 and 3, the baud rate is determined by the timer/counter D overflow rate and the value of the BSEL bit (T1CON.7), as follows.

$$\text{Mode 1 and 3 baud rate} = \frac{2^{\text{BSEL}}}{16} \times \text{timer D overflow rate}$$

The timer D interrupt enable bit (TDIE, T1CON.3) should be disabled for baud generator applications. The timer itself can be configured for either "timer" or "counter" operation and any one of its three operating modes may be selected. In most applications, it is configured to "timer" operation in 8-bit auto-reload mode (the high nibble of T1MOD = 0010B), where baud rate is calculated by the following formula:

$$\text{Baud rate in auto-reload mode} = \frac{2^{\text{BSEL}}}{16} \times \frac{\text{CPU clock}}{12 \times (256 - \text{TDH})}$$

You can achieve very low baud rates using timer D by leaving the timer D interrupt enabled, configuring the timer to run as a 16-bit timer (the high nibble of T1MOD = 0001B), and then using the timer D interrupt to do a 16-bit software reload.

Table 12–1. Commonly Used Baud Rates Generated by Timer D

Baud Rate	CPU Clock	BSEL Bit	Timer D Values		
			TDC Bit	Mode	Reload Value
Mode 0; 1 MHz max.	6 MHz	x	x	x	x
Mode 2; 187.5 kHz	6 MHz	0	x	x	x
Mode 2; 375 kHz	6 MHz	1	x	x	x
Modes 1 and 3:					
62.5 kHz	6 MHz	1	0	2	FFH
19.2 kHz	11.0592 MHz	1	0	2	FAH
9.6 kHz	11.0592 MHz	0	0	2	FAH
4.8 kHz	11.0592 MHz	0	0	2	F4H
2.4 kHz	11.0592 MHz	0	0	2	E8H
1.2 Hz	11.0592 MHz	0	0	2	D0H
110 Hz	3 MHz	0	0	2	72H
110 HZ	6 MHz	0	0	1	FEE3H

NOTE: CPU clock is the same as f_{OSC}.

Table 12–2. Serial Baud Rate Calculations

Transmission Type	SIO Mode	Baud Rate Formula	Baud rate at 6 MHz
Synchronous	0	$\frac{\text{CPU clock}\ (1)}{6}$	1 MHz
Asynchronous	1, 3	$\frac{\text{CPU clock} \times 2^{\text{BSEL}}}{12 \times 16 \times [256 - (\text{TDH})]}$	$\frac{31250 \times 2^{\text{BSEL}}}{[256 - (\text{TDH})]}$
	2	$\frac{\text{CPU clock} \times 2^{\text{BSEL}}}{2 \times 16}$	187.5 kHz (if BSEL = "0") 375 kHz (if BSEL = "1")

NOTE: CPU clock frequency is the same as f_{OSC}.

Serial Communication for Multiprocessor Configurations

The multiprocessor communication feature allows a "master" KS88C4400 to send a multiple-frame serial message to one "slave" device in a multi-KS88C4400 configuration (see Figure 12–8). It does this without interrupting other slaves that may be on the same serial line.

This feature can be used only in UART modes 2 or 3. It is most commonly used with mode 2, which runs at 250 kHz without using timer C. (Mode 2 can also run at 500 kHz if you set the BSEL bit in the T1CON register to "1", selecting double baud rate.)

In modes 2 and 3, nine data bits are received. The 9th bit value is written to RB8 (SIOCON.2). Then comes a stop bit. You can program this function so that when the stop bit is received, the serial port interrupt will be activated only if RB8 = "1".

To enable this feature, you must set the multiprocessor communication enable bit in the SIOCON register (MCE, SIOCON.5). If the MCE bit is set to "1", serial data frames received in which the 9th bit is "0" do not generate an interrupt, but simply separate the address from the serial data.

Sample Protocol for Master/Slave Interaction

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that in an address byte, the 9th bit is "1" and in a data byte, it is "0".

An address byte interrupts all slaves so that each slave can examine the received byte and see if it is being addressed. The addressed slave then clears its MCE bit and prepares to receive incoming data bytes. The slaves that were not addressed leave their MCE bits set and continue operating normally while ignoring the incoming data bytes.

While the MCE bit setting has no effect in mode 0, it can be used in mode 1 to check the validity of the stop bit. For a mode 1 reception, if MCE = "1", the receive interrupt will not be activated until a valid stop bit is received.

Setup Procedure for Multiprocessor Communications

The following steps are a general guideline for configuring multiprocessor communications:

1. Set all KS88C4400 devices (masters and slaves) to SIO mode 2 or 3.
2. Write all MCE bits of the slave devices to "1".
3. The master device's transmission protocol is as follows:
   - First byte Address which identifies the target slave device (9th bit = "1")
   - Next bytes Data (9th bit = "0")
4. When the targeted slave device receives the first byte, all slaves are interrupted, since the 9th data bit is "1". Each slave must compare the address byte to its own address; the addressed slave then clears its MCE bit.

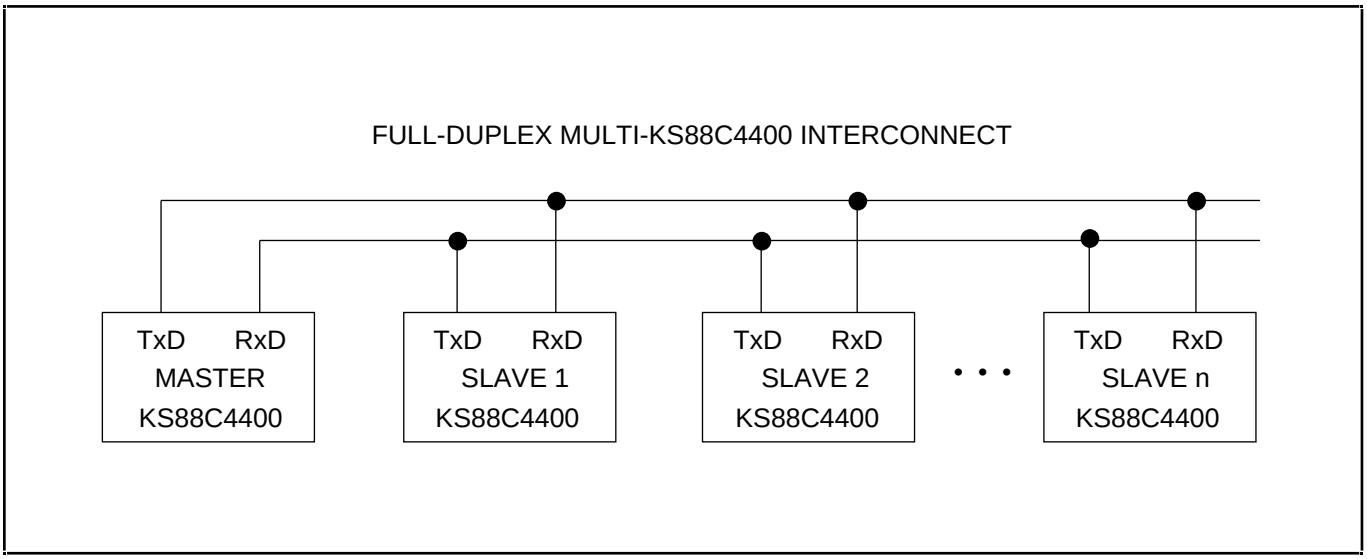


Figure 12–8. Connection Example for Multiprocessor Serial Data Communications

☞ PROGRAMMING TIP — Programming the Serial Port for Mode 0 Operation

This example shows how to program the KS88C4400 serial port to operate in synchronous mode (mode 0). Assume the following program parameters:

— CPU clock frequency = 6 MHz

— Device A is enabled at P2.6 (P2.6 is set to low level)

— Device B is enabled at P2.7 (P2.7 is set to low level)

Program Function Description

One-byte data is transmitted to device A and one-byte data is received from device B. The baud rate of CPU clock/6 (1 MHz) is selected. In other words, bit 7 (BSEL) in the timer module 1 control register T1CON is "0". Now, suppose the following conditions exist:

— Transmit data is in the register labeled 'TRANS'.

— Data which is received from a device is loaded to register 'RECEIVE'.

— The subroutine SIO_T_R is called every 50 ms.

☞ **Programming Tip — Programming the Serial Port for Mode 0 Operation (Continued)**

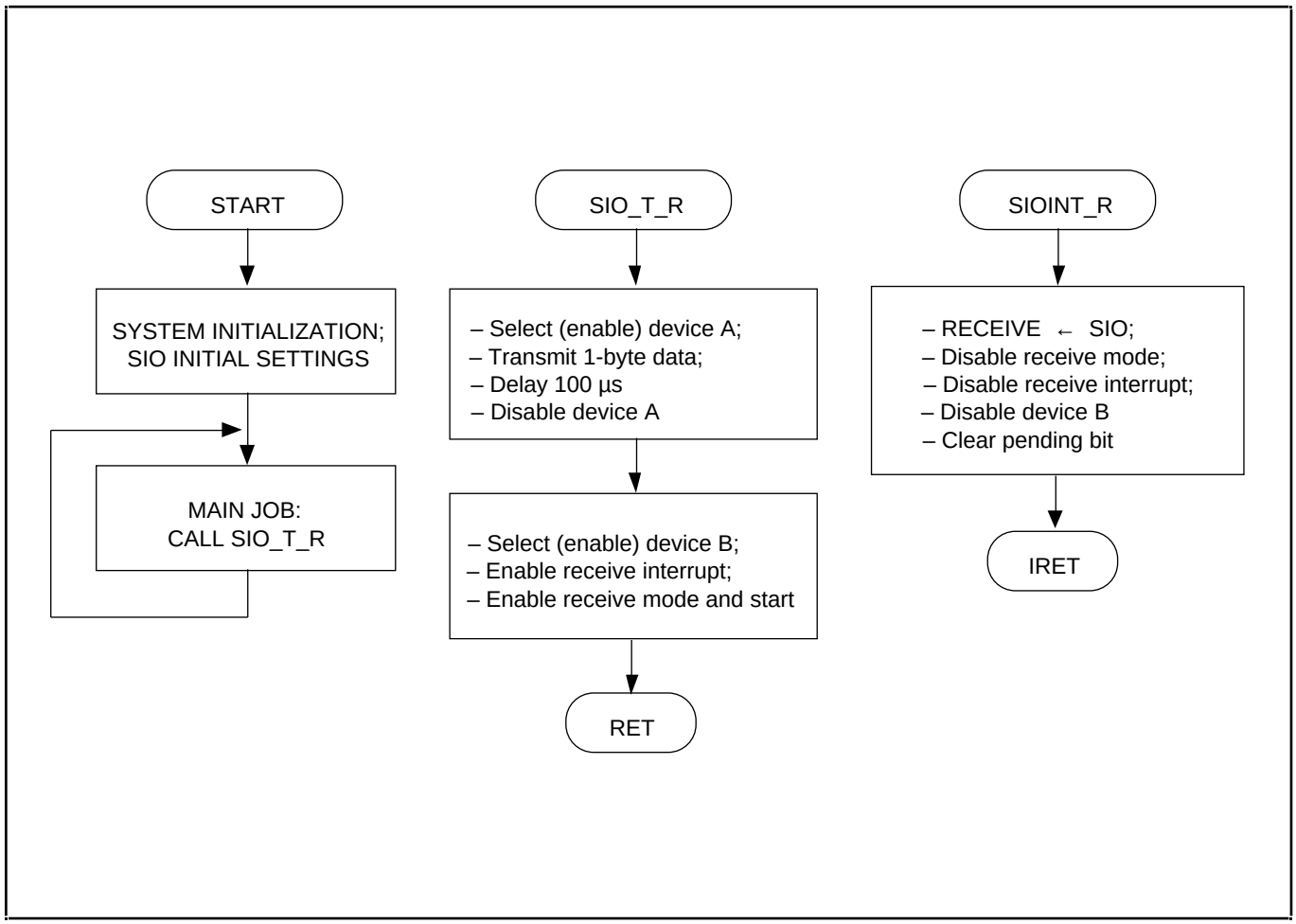


Figure 12–9. Flowchart for Serial Port Programming Tip (Mode 0)

☞ **Programming Tip — Programming the Serial Port for Mode 0 Operation (Continued)**

```
START   DI                          ;  Disable interrupts
        •                           ;  System initialization settings
        •
        •
        LD      P2CON,#0A0H
        LD      P2,#0C0H            ;  Set port 0 to output mode; devices A and B are not
                                    ;  selected
        LD      SIOCON,#02H         ;  Mode 0 select; enable only receive interrupt
                                    ;  Receive disable
        LD      SIOPND,#03H         ;  Clear pending bit
        EI                          ;  Enable interrupts
        •
        •
        •
MAIN    NOP
        •
        •
        •
        CALL    JOB                 ;  Run other job
        •
        •
        •
        CALL    SIO_T_R             ;  Run SIO subroutine
        •
        •
        •
        JP      T,MAIN
        •
        •
        •
TRANS   EQU     50H                 ;  Transmit data buffer
RECEIVE EQU     51H                 ;  Receive data buffer

SIO_T_R
        AND     P2,#0BFH            ;  Select (enable) device A
        NOP
        LD      SIO,TRANS           ;  TRANS data 1-byte output
        CALL    WAIT100µs
        OR      P2,#40H             ;  Disable device A
        NOP
        AND     P2,#7FH             ;  Select (enable) device B
        OR      SIOCON,#10H         ;  Enable receive mode and start receive operation
        RET

WAIT100 µs PUSH R3                  ;  100-µs delay routine
        LD      R3,#32H
LOOP    DJNZ    R3,LOOP
        POP     R3
        RET
```

(Continued on next page)

☞ Programming Tip — Programming the Serial Port for Mode 0 Operation (Concluded)

```
;           SIO receive interrupt service routine:

SIOINT_R    OR      P2,#80H          ;  Disable device B
            AND     SIOCON,#0EFH     ;  Receive disable
            LD      RECEIVE,SIO      ;  Data restore
            LD      SIOPND,#02H      ;  Clear pending bit
            IRET
```

☞ Programming Tip — Programming the Serial Port for Mode 3 Operation

This example shows how to program the KS88C4400 serial port to operate in 11-bit asynchronous transmit/ receive mode (mode 3). Assume these conditions:

— Main oscillator frequency = 11.0592 MHz
— No multiprocessor communication is required
— Baud rate is 9600 BPS (see formula below)
— SIO mode 3 (11-bit, asynchronous type, is used)
— Timer/counter D overflow output is used as the SIO shift clock
— Timer C is not used in this sample program

Program Function Description

Use this formula to calculate the 9600 BPS baud rate (assume a CPU clock of 11.0592 MHz, T1CON.7 = "0", and TDH = 0FAH):

$$9600\ \text{Baud} = \frac{\text{CPU clock} \times 2^{\text{BSEL}}}{12 \times 16 \times (256 - \text{TDH})}$$

The subroutine SIO_T_R transmits one byte at a time, but received data will be added to SR_SUM when the receive interrupt occurs.

☞ **Programming Tip — Programming the Serial Port for Mode 3 Operation (Continued)**

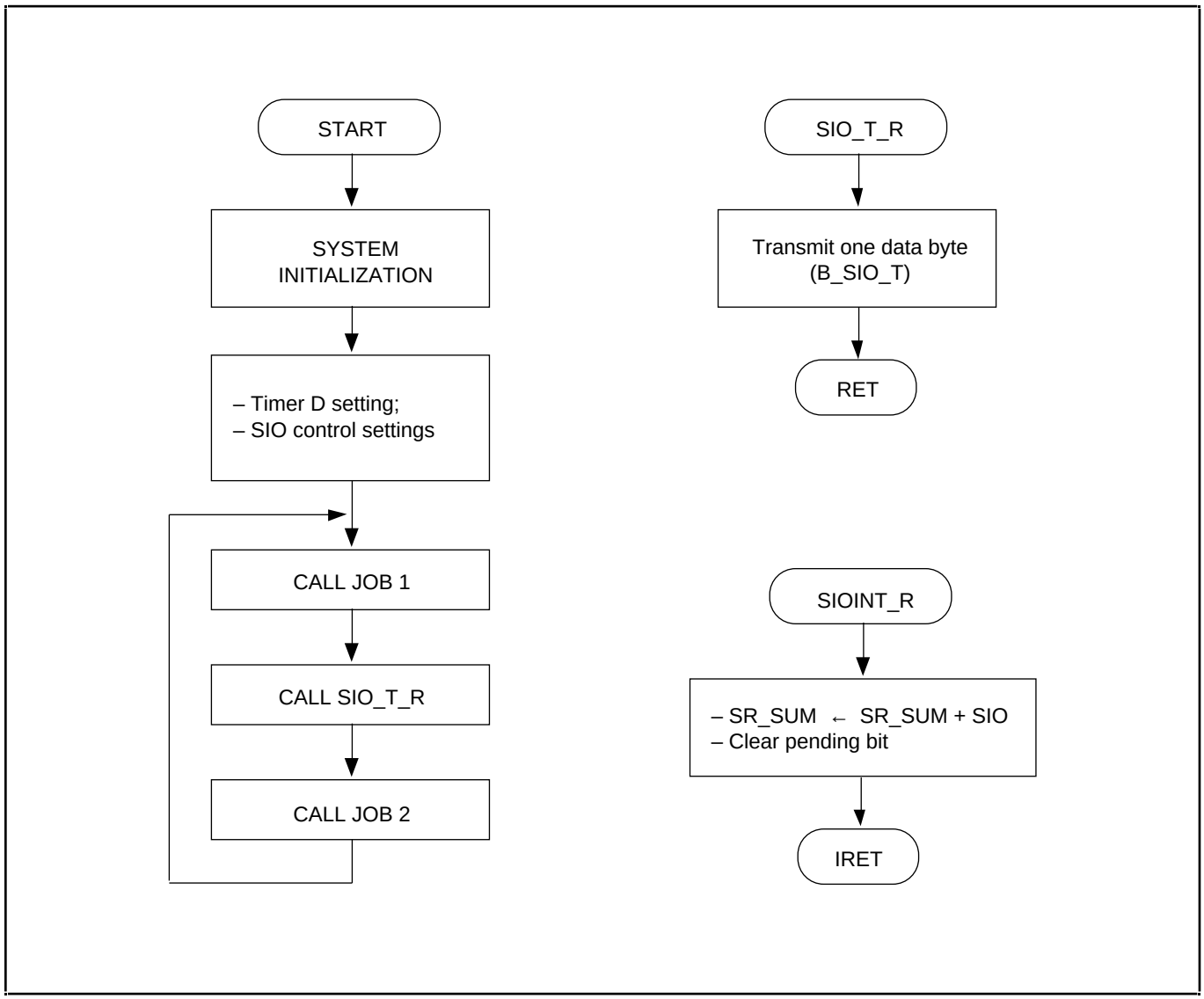


Figure 12–10. Flowchart for Serial Port Programming Tip (Mode 3)

☞ **Programming Tip — Programming the Serial Port for Mode 3 Operation (Continued)**

```
B_SIO_T   EQU     40H                 ;  SIO transmit buffer register
SR_SUM    EQU     41H                 ;  Total value of received data
          •
          •
          •
START
          •
          •                           ;  System initialization settings
          •
          LD      TDL,#0FAH
          LD      TDH,#0FAH           ;  Load the auto-reload value
          LD      T1MOD,#20H          ;  Disable the timer counter D gate function
                                      ;  Select CPU clock /6 (CPU clock = 11.0592 MHz)
                                      ;  Select auto-reload operating mode (mode 3)
                                      ;  (Timer C is not used in this program)
          LD      T1CON,#32H          ;  Select normal baud rate
                                      ;  Disable timer C and D interrupt
                                      ;  Timer D run enable
          LD      SIOCON,#0D2H        ;  SIO mode 3, multiprocessing bit is low
                                      ;  Receive enable; 9th transmit bit is low
                                      ;  RxD interrupt is enabled
                                      ;  TxD interrupt is disabled
          LD      SIOPND,#03H         ;  Clear the SIO pending bits

          EI                          ;  Enable interrupts
          •
          •
          •
MAIN      NOP
          •
          •
          •
          CALL    JOB1                ;  Run job 1
          •
          •
          •
          CALL    SIO_T               ;  Run SIO transmit subroutine
          CALL    JOB2                ;  Run job 2
          •
          •
          •
          JP      T,MAIN
          •
          •
          •
SIO_T     LD      SIO,B_SIO_T         ;  Transmit 1-byte data
          RET

;         SIO receive interrupt service routine

SIOINT_R  ADD     SR_SUM,SIO          ;  Add receive data to SR_SUM
          LD      SIOPND,#02H         ;  Clear receive interrupt pending bit
          IRET
```

13 PWM and Capture

OVERVIEW

The KS88C4400 pulse width modulation (PWM) unit has the following components:

— 16-bit counter
— 2-bit prescaler
— Two 8-bit comparators
— Two 8-bit PWM data registers (PWM0, PWM1)
— PWM control register (PWMCON)
— PWM counter overflow interrupt (IRQ1, vector B8H)
— Two PWM output pins (PWM0, PWM1)

An 8-bit capture unit is included in the PWM module. The capture unit is controlled by PWMCON register settings. It has the following components:

— 8-bit capture register (PWMCAP)
— Capture input pin (P3.6 /CAP, pin 30)
— Capture input interrupt (IRQ1, vector BAH)

PWM Control Register (PWMCON)

The control register for the PWM module, PWMCON, is located at register address FCH in set 1, bank 1. A reset clears the PWMCON register to '00H'.

See Figure 13–1 for an overview of PWMCON control functions.

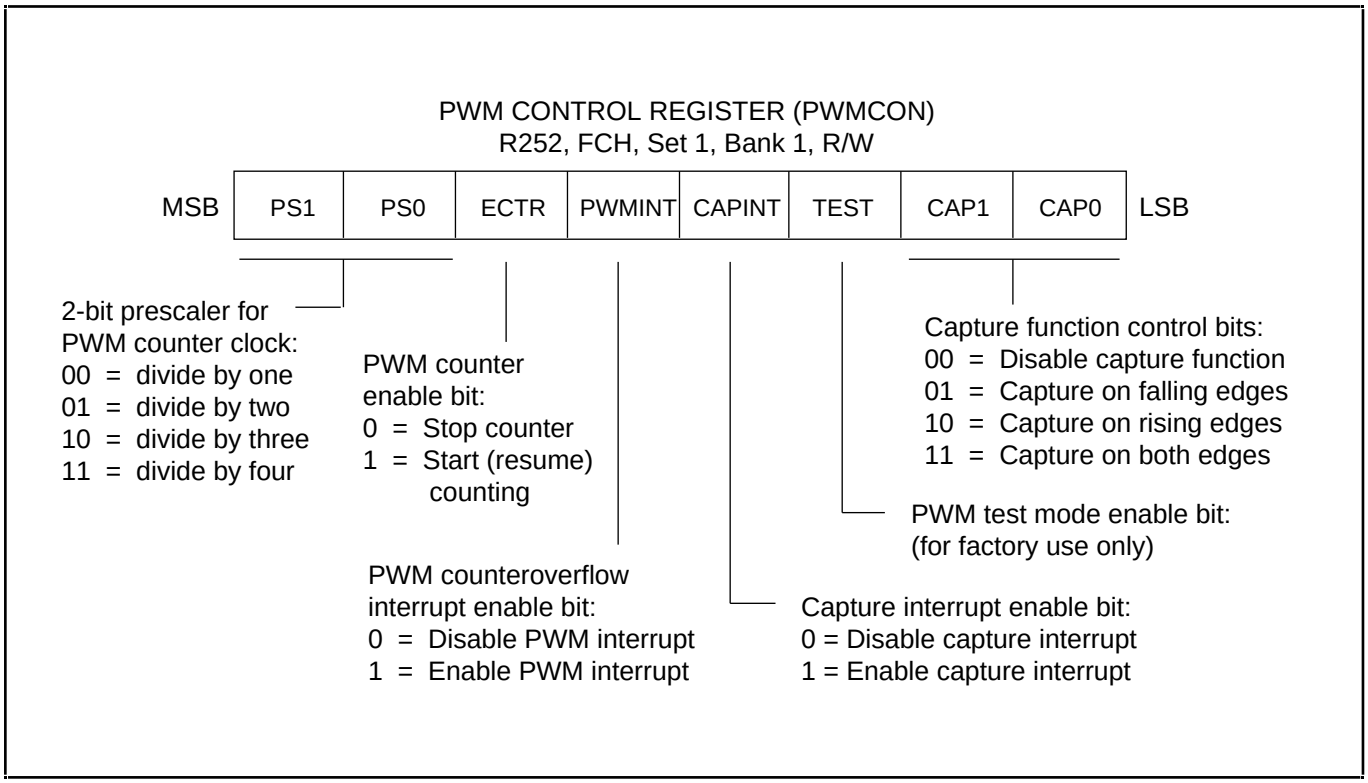


Figure 13–1. PWM/Capture Module Control Register (PWMCON)

PWM FUNCTION DESCRIPTION

The PWM counter is a 16-bit incrementing counter. To start the counter and enable the PWM module, you set bit 5 (ECTR) of the PWMCON register to "1". If the counter is stopped, it retains its current count value; when restarted, it resumes counting from the retained count value.

A 2-bit prescaler controls the clock input frequency to the PWM counter. Using prescaler bit settings, you can divide the input clock by one (non-divided), two, three, or four. The prescaler output is the clock frequency of the PWM counter.

The PWM counter overflows when it reaches FFFFH, and then continues counting from zero. If the PWM counter overflow interrupt is enabled, an IRQ1 interrupt (vector B8H) is generated. The interrupt enable bit is bit 4 in the PWMCON register (PWMINT).

The PWM0 data register, called PWM0, is located in set 1, bank 1, address FEH. The PWM1 data register, PWM1, is in set 1, bank 1, at address FDH. Both data registers are read-write addressable. By loading specific values into the respective data registers, you can modulate the pulse width at the corresponding PWM output pins, PWM0 and PWM1. The two 8-bit PWM circuits function identically: Each has its own 8-bit data register and 8-bit comparator, and comparing its unique data register value to the lower 8-bit value of the 16-bit PWM counter.

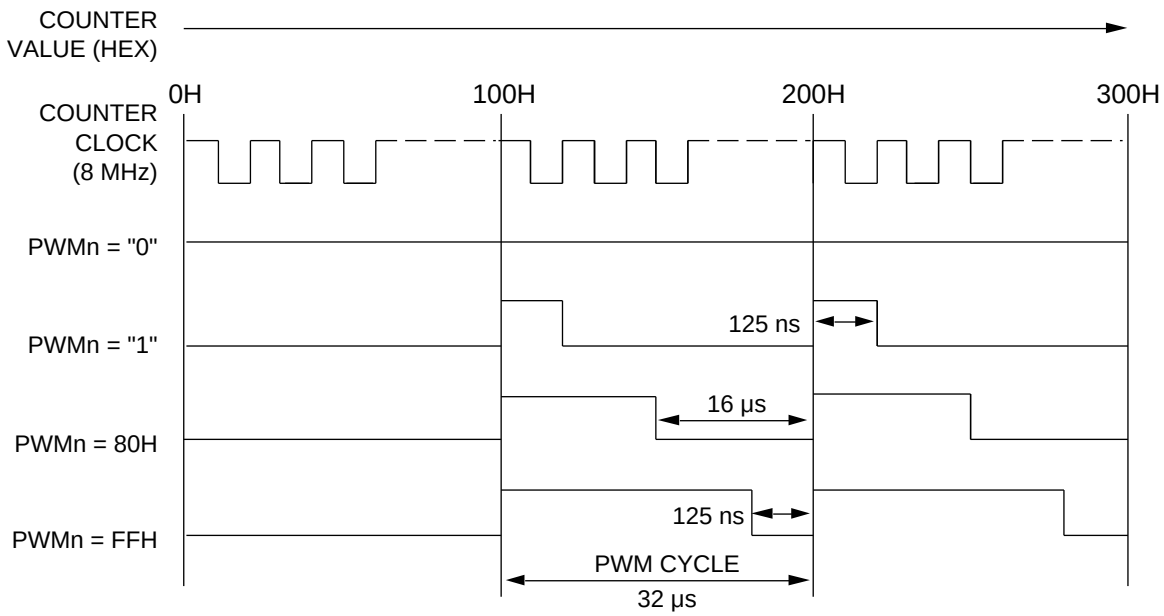
The level at the output pins toggles high and low at a frequency equal to the counter clock, divided by 256 (2^8). The duty cycle of the PWM0 and PWM1 pin ranges from 0% to 99.6%, depending on the corresponding data register value.

To determine the PWM circuit's duty cycle, its 8-bit comparator sends the output level high ("1") when the data register value is greater than the lower 8-bit count value. (The output level is low ("0") when the data register value is less than or equal to the lower 8-bit count value.) The output level at the PWM0 and PWM1 pins remains at low level for the first 256 counter clocks. Then, each PWM waveform is repeated continuously, at the same frequency and duty cycle, until one of three events occurs:

— The counter is stopped

— The counter clock frequency is changed

— A new value is written to the PWM data register



NOTE: 'n' = 0 and 1, for PWM0 and PWM1.

Figure 13–2. PWM Output Waveform and Timing Diagram

Staggered PWM Outputs

The PWM0 and PWM1 outputs are staggered in order to reduce the overall noise level on the pulse width modulation circuits. If you load the same value to both PWM data registers, a match condition (data register value = lower 8-bit count value) occurs on the same clock cycle for both PWM circuits.

In this case, only the PWM0 output will be toggled high on the clock edge following the match signal. The PWM1 output is delayed by two counter clock cycles, and so on for subsequent clocks (see Figure 13–3).

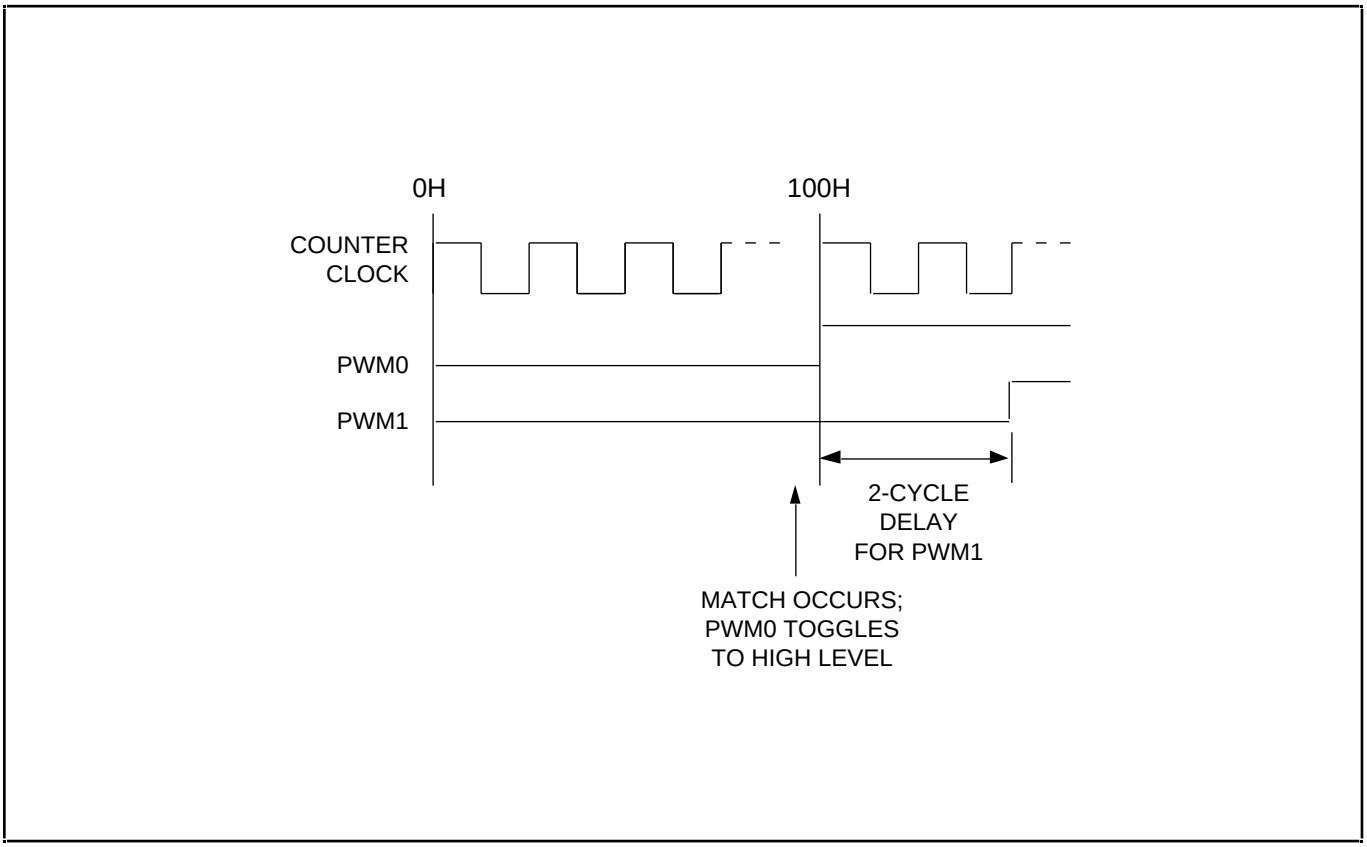


Figure 13–3. PWM0 and PWM1 Output Delay

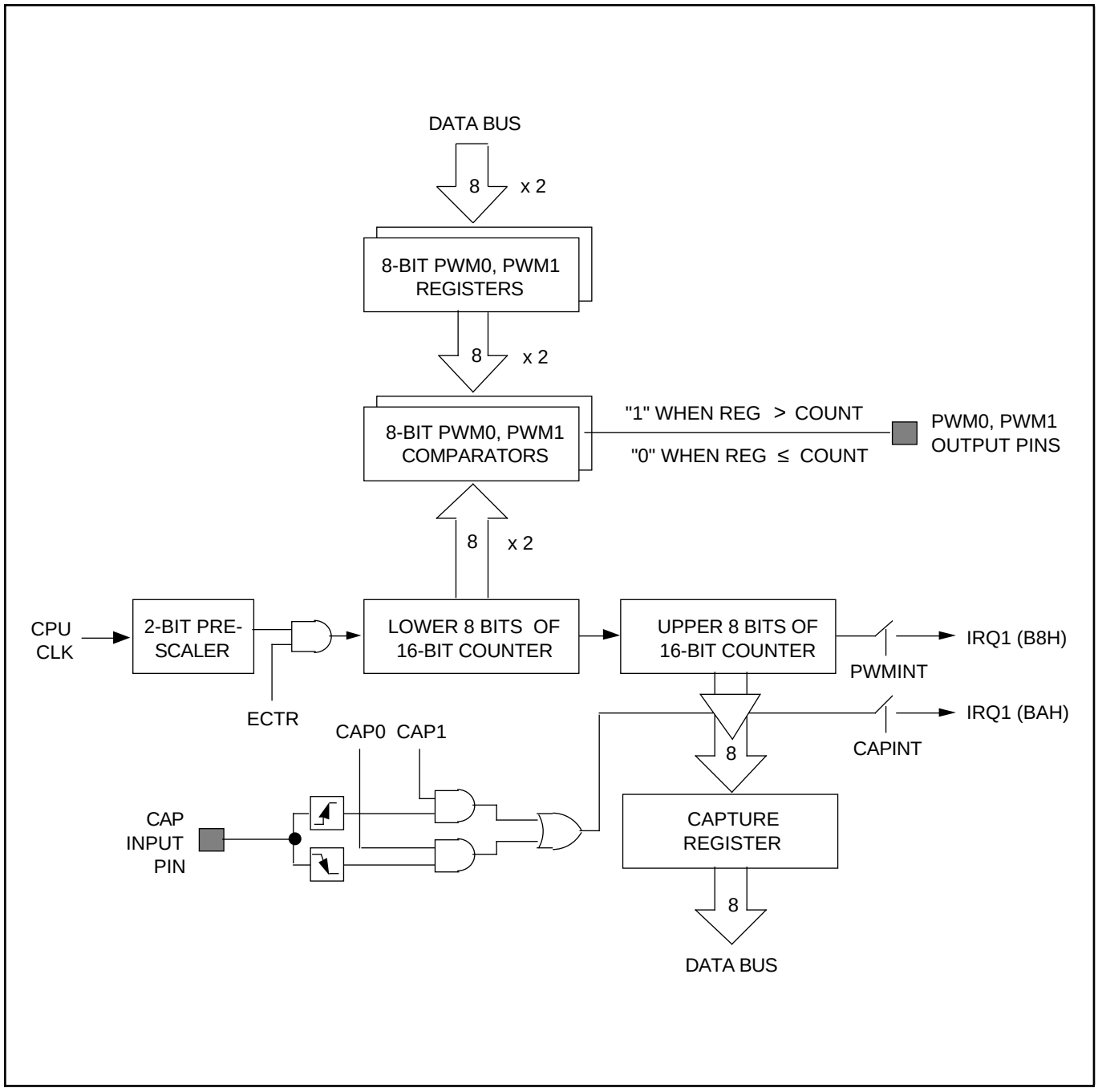


Figure 13–4. PWM/Capture Module Functional Block Diagram

☞ PROGRAMMING TIP — Programming the PWM Module to Sample Specifications

This example shows you how to program the 8-bit pulse-width modulation (PWM) module, assuming the following parameters:

— The oscillation frequency of the main crystal is 6 MHz

— PWM0 data is in working register R0

— PWM1 data is in working register R1

The sample program performs these actions:

1. Set the PWM1 frequency to 11.719 kHz
2. If R0F.0 = "1", then
   PWM0 ← PWM0 + 1H
   PWM1 ← PWM1 + 2H

   Else

   PWM1 ← PWM1 – 1H

```
                •
                •
                •
        SB1                         ;  Select bank 1
        LD      PWMCON,#60H         ;  P.S. ← '01B' (Select 11.719 kHz PWM frequency)
                                    ;  Enable the PWM counter (start PWM); disable interrupt
        CLR     PWM0                ;  Clear PWM0 data register
        CLR     PWM1                ;  Clear PWM1 data register
        SB0                         ;  Select bank 0
                •
                •
                •
        SB1                         ;  Select bank 1
        BTJRT   AA1,R0F.0
        DEC     PWM1                ;  PWM1 ← PWM1 – 1
        JR      t,AA2

AA1:    INC     PWM0                ;  PWM0 ← PWM0 + 1
        INC     PWM1                ;  PWM1 ← PWM1 + 1
        INC     PWM1                ;  PWM1 ← PWM1 + 2

AA2:    NOP
                •
                •
                •
```

Data Capture Unit

An 8-bit data capture unit is integrated with the PWM module. The capture unit detects incoming signal edges and can also be used to measure the pulse width of the incoming signals.

The capture unit captures the upper 8-bit value of the 16-bit counter when a signal edge transition is detected at the CAP pin. The captured value is then dumped into the PWMCAP register (set 1, bank 1, FFH) where it can then be read. By manipulating bits 0 and 1 (CAP0, CAP1) of the PWMCON register, you can set edge detection at the CAP pin for rising edges, falling edges, or both signal edge types.

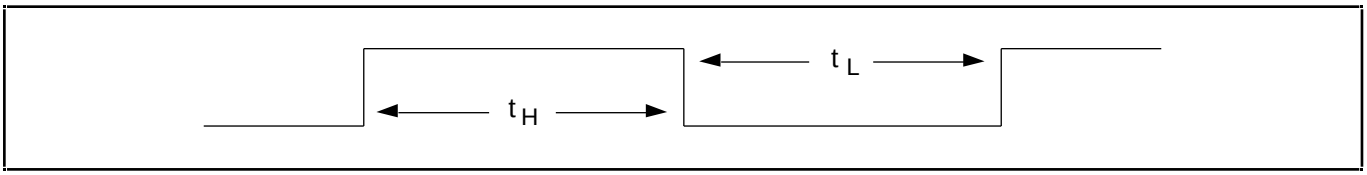
Signal edges captured at the CAP pin can also be used to generate an interrupt. Bit 3 (CAPINT) in the PWMCON register is the capture interrupt enable bit. The capture interrupt is level 1 (IRQ1) in the KS88C4400 interrupt structure; its vector address is BAH. Level IRQ1 has two interrupts: the PWM overflow interrupt (vector B8H) and the capture interrupt (vector BAH). The PWM overflow interrupt always has higher priority.

Using the capture interrupt, capture register contents can be read from edge to edge and the elapsed time between pulses calculated.

☞ PROGRAMMING TIP — Programming the Capture Unit to Sample Specifications

This example shows you how to program the KS88C4400 capture unit, assume the following parameters:

— The main oscillator frequency is 6 MHz
— Timer A interrupt occurs every 2 ms
— The following waveform is being input at the CAP pin:



— The following registers are assigned for program values:

Register 70H	First captured count value
Register 71H	Second captured count value
Register 72H	Third capture count value
Register 73H	Down-counter that is decremented by one at each timer A interrupt
Register 74H	Capture counter
Register 77H	Flags

Additional sample program information:

1. If 4.35 ms $<$ t_H, t_L $<$ 4.6 ms, then set bit zero (LDR) in register 77H; otherwise clear the zero bit (LDR) in register 77H.
2. If the interval between two rising signal edges (capture trigger) is $>$ 30 ms, disregard the capture setting.

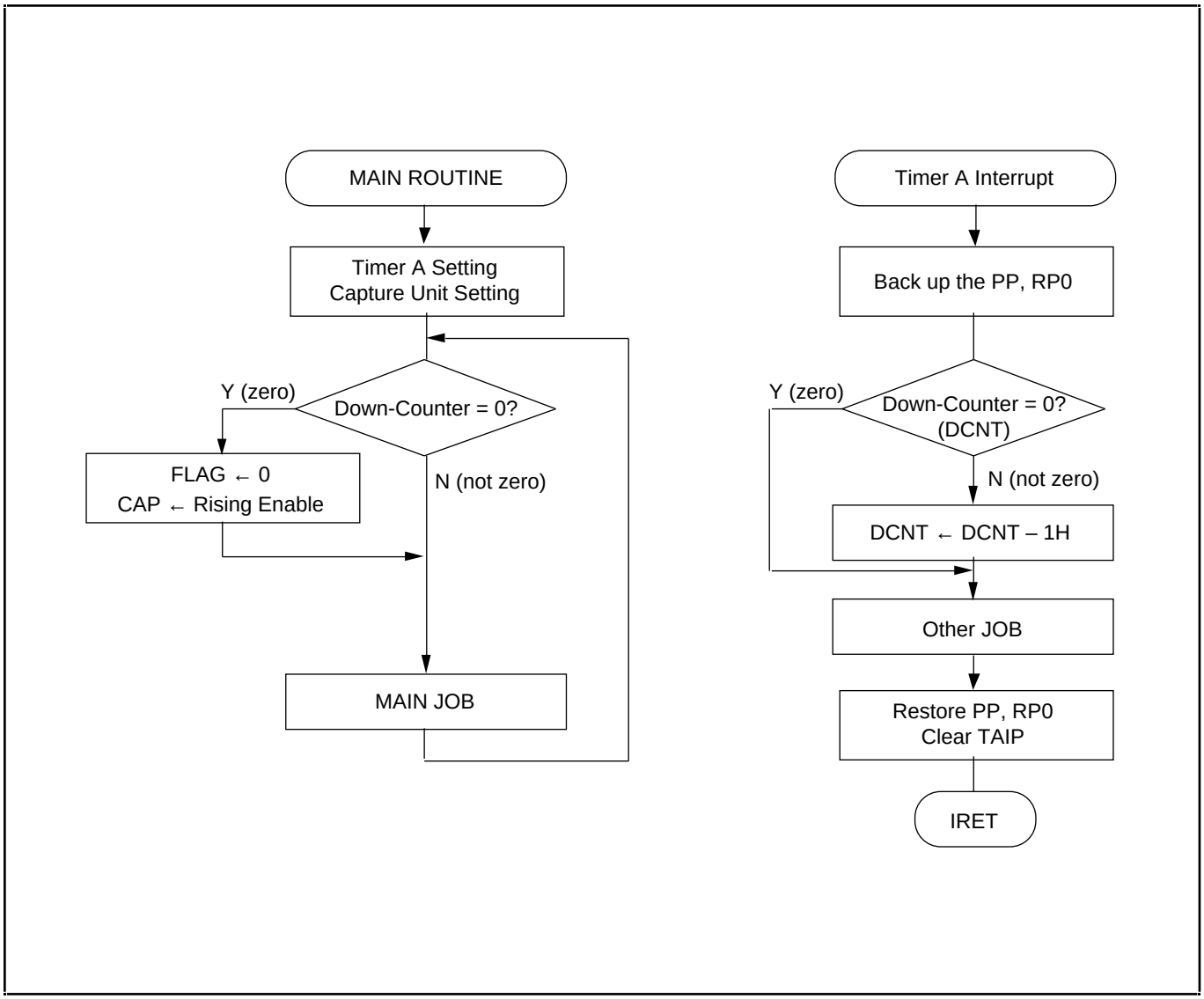


Figure 13–5. Decision Flowchart for Capture Unit Programming Tip

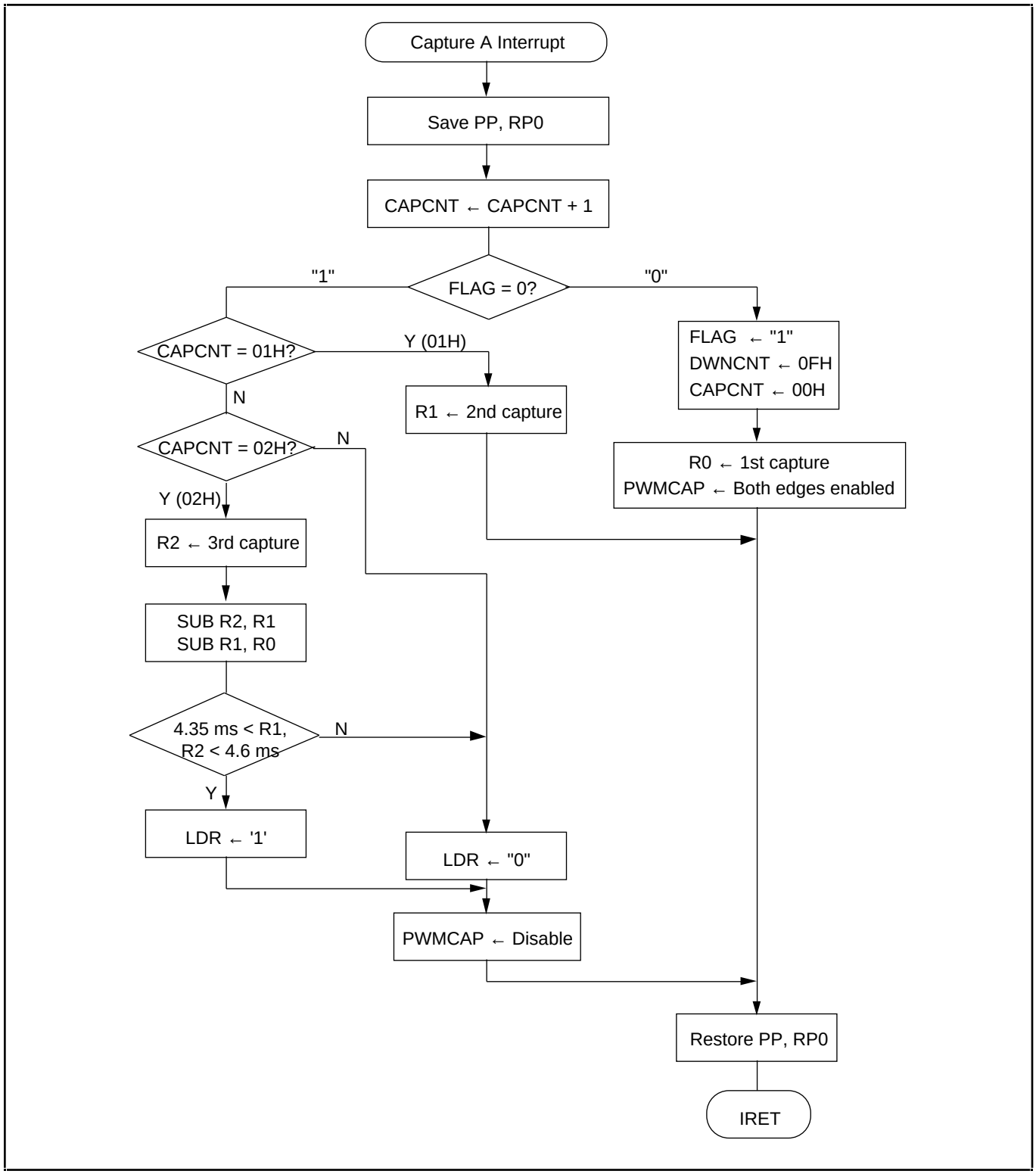


Figure 13–6. Decision Flowchart for Capture Unit Interrupt (Continued)

☞ **PROGRAMMING TIP — (Continued)**

```
                •
                •
                •
LDR     EQU     0
FLAG    EQU     7
CAPCNT  EQU     4
DWNCNT  EQU     3
                •
                •
                •
        CLR     PP              ; Select page 0
        LD      T0CON,#56H      ; PS ← 5, interval mode, enable timer A interrupt
        LD      TADATA,#01H     ; For 2-ms interval (6 MHz /1000 ÷ 6 ÷ 2 = 0.5 kHz
                                ;   = 2 ms)
                •
                •
                •
JOB     SRP0    #70H            ; RP0 ← 70H
        SB1
        CP      RDWNCNT,#00H    ; Down-counter = "0"?
        JP      NE,MAIN         ; If not zero, then jump to MAIN
        BITR    R7.FLAG         ; Clear the 'FLAG'
        LD      PWMCON,#0AAH    ; Capture A ← enable interrupt, trigger on rising edges
        SB0
MAIN                            ; Other JOB...

                •
                •
                •
        JP      T,JOB           ; For looping
                •
                •
                •
TAINT   PUSH    PP              ; Save page pointer
        PUSH    RP0             ; Save register pointer 0
        SRP0    #70H            ; RP0 ← 70H
        CP      RDWNCNT,#00H    ; R3 (down-counter) = "0"?
        JP      EQ,TA1
        DEC     RDWNCNT         ; If not zero, then decrement R3 by 1
TA1             •
                •
                •
        POP     RP0             ; Restore register pointer 0
        POP     PP              ; Restore page pointer
        IRET                    ; Return from timer A interrupt service routine
                •
                •
                •
```

(Continued on next page)

☞ **PROGRAMMING TIP — (Concluded)**

```
CAPINT      PUSH    PP
            PUSH    RP0                 ;  Back up the PP and RP0
            SRP0    #70H                ;  RP0 ← 70H
            SB1
            INC     RCAPCNT             ;  Increment the capture counter
            BTJRT   CAPTURE1,R7.FLAG
            BITS    R7.FLAG
            CLR     RCAPCNT             ;  Clear capture counter
            LD      RDWNCNT,#0FH        ;  Down-counter ← 15 (for counting 30 ms)
            LD      R0,PWMCAP           ;  R0 ← First captured count value
                                        ;  PWMCAP = FFH, set 1, bank 1)
            LD      PWMCON,#0A9H        ;  Enable trigger on both rising and falling edges
            SB0
CAPRTN      POP     RP0                 ;
            POP     PP                  ;  Restore the PP and RP0 values
            IRET
CAPTURE1    CP      RCAPCNT,#01H
            JP      NE,CAPTURE2
            LD      R1,PWMCAP           ;  R1 ← Second captured count value
            JR      T,CAPRTN
CAPTURE2    CP      RCAPCNT,#02H        ;  CAPCNT = 02H?
            JP      EQ,CAPTURE3
CAPTURE4    BITR    R7.LDR              ;  Clear the LDR bit in R7
CAPTURE5    LD      PWMCON,#0A0H        ;  Disable the capture unit
            JR      T,CAPRTN
CAPTURE3    LD      R2,PWMCAP           ;  R2 ← Third capture count value
            SUB     R2,R1               ;  R2 ← (Third capture value – second capture value)
            SUB     R1,R0               ;  R1 ← (Second capture value – first capture value)
            CP      R1,#24H             ;  24H = 4.6 ms
            JP      UGT,CAPTURE4        ;  If high signal period > 4.6 ms, then go to CAPTURE4
            CP      R2,#24H
            JP      UGT,CAPTURE4        ;  If low signal period > 4.6 ms, then go to CAPTURE4
            CP      R1,#22H             ;  22H = 4.35 ms
            JP      ULT,CAPTURE4        ;  If high signal period < 4.35 ms, then go to CAPTURE4
            CP      R2,#22H
            JP      ULT,CAPTURE4        ;  If low signal period < 4.35 ms, then go to CAPTURE4
            BITS    R7.LDR              ;  Set bit 'LDR'
            JP      T,CAPTURE5          ;  Jump to CAPTURE5 unconditionally
        •
        •
        •
```

note

14 Analog-to-Digital Converter

OVERVIEW

The 8-bit A/D converter (ADC) module uses successive approximation logic to convert analog levels entering at one of the eight input channels to equivalent 8-bit digital values. The analog input level must lie between the AV_{REF} and AV_{SS} values. The A/D converter has the following components:

— Analog comparator
— Successive approximation register
— D/A converter logic (resistor ladder type)
— ADC control register (ADCON)
— Eight multiplexed analog data input pins (ADC0–ADC7)
— 8-bit A/D conversion data output register (ADOUT)
— 8-bit digital input register (ADIN; alternately, input port 7)
— AV_{REF} and AV_{SS} input pins

To initiate an analog-to-digital conversion procedure, you write the SCH bits in the A/D converter control register ADCON to select one of the eight analog input pins (ADCn, n = 0–7). The read-write ADCON register is located in set 1, bank 1, at address FBH.

During a normal conversion, you initially set the successive approximation register to 80H (the approximate half-way point of an 8-bit register). This register is then updated automatically during each conversion step. The KS88C4116 performs 8-bit conversions for one input channel at a time. You can dynamically select different channels by manipulating the SCH bit value (SCH0–SCH2) in the ADCON register.

The A/D conversion process requires 24 steps (24 clock edges) to convert each bit. Therefore, a total of 192 clocks are required to complete an 8-bit conversion: With an 8 MHz CPU clock frequency, one clock cyce is125 ns. If each bit conversion requires 24 clocks, the conversion rate is calculated as follows:

125 ns × 24 clocks × 8 bits = 192 clocks, or 24 µs at 8 MHz

The digital result is then dumped into the output register ADOUT (set 1, bank 1, FAH) and the A/D converter unit enters an idle state. Since the A/D unit does not generate an interrupt to signal a completed conversion, you must first read out the contents of ADOUT before another conversion is initiated. Otherwise, the previous result will be overwritten.

NOTE

Because the A/D converter has no sample-and-hold circuitry, it is very important that fluctuation in the analog level at the ADC0–ADC7 input pins during a conversion procedure be kept to an absolute minimum. Any change in the input level, perhaps due to noise, will invalidate the result.

Using A/D Pins for Standard digital Input

You can also use the A/D unit as a standard digital input port, port 7 (see Figure 14–1). The ADC0–ADC7 share pin names are P7.0–P7.7, respectively. Incoming port 7 data values are read directly from the 8-bit digital input register ADIN, located in set 1, bank 1 at address F9H.

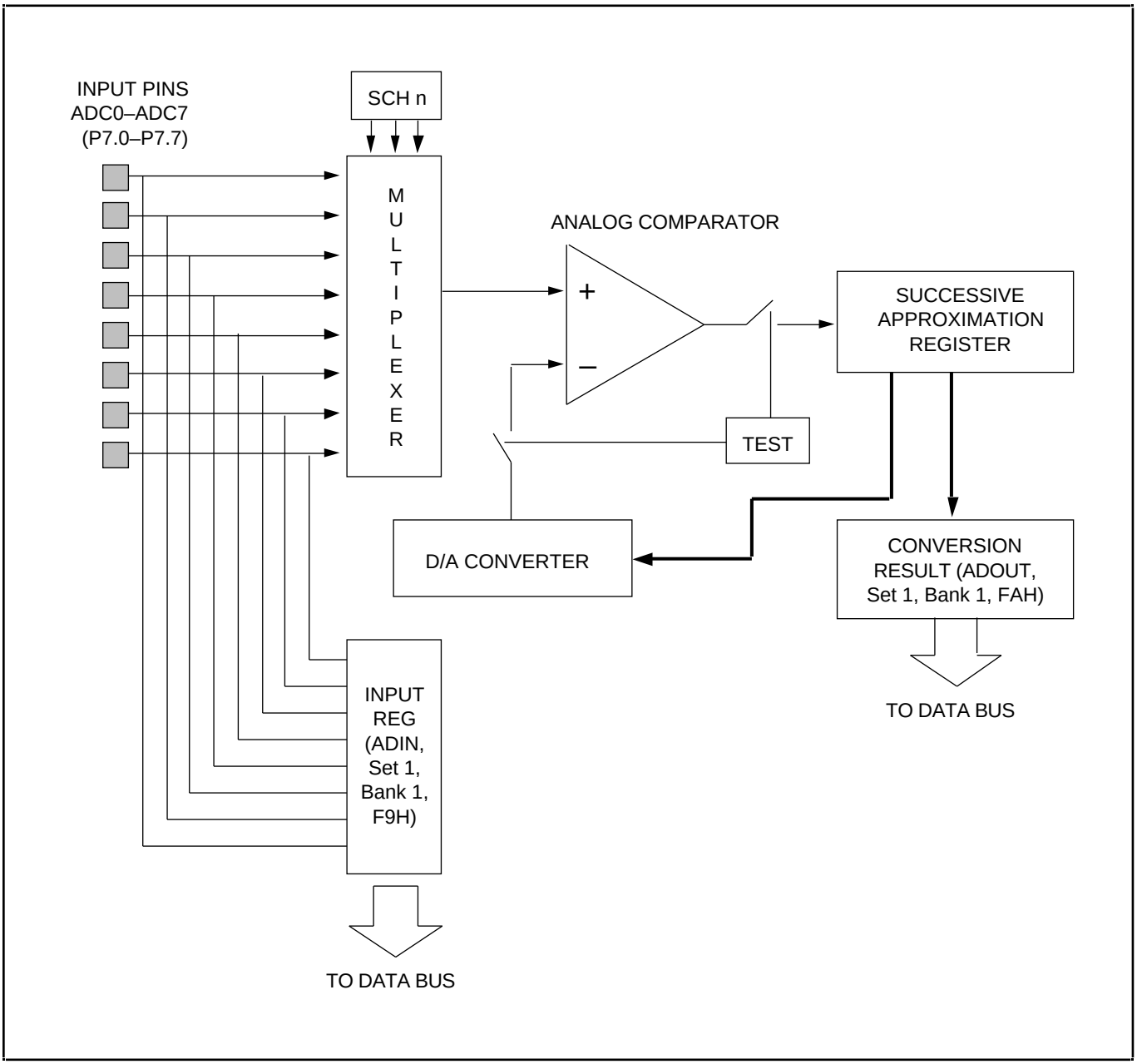


Figure 14–1. A/D Converter Functional Block Diagram

A/D Converter Control Register (ADCON)

The A/D converter control register, ADCON, is located at address FBH in set 1, bank 1. Only bits 4–7 are used in the KS88C4400 implementation. ADCON is read-write addressable using 1-bit or 8-bit instructions. It has two functions:

— Bits 4, 5, and 6 (SCH0–SCH2) are used to select the analog data input pin

— Bit 7 is a test bit for factory use only

After a reset, the ADC0 pin (pin number 41) is automatically selected as the analog data input pin, and the test bit is turned off. (The test bit should always remain cleared to "0".)

You can select only one analog input channel at a time. Other analog input pins (ADC0–ADC7, pins 41, 43, 44, and 46–49) can be selected dynamically by manipulating the SCH bits.

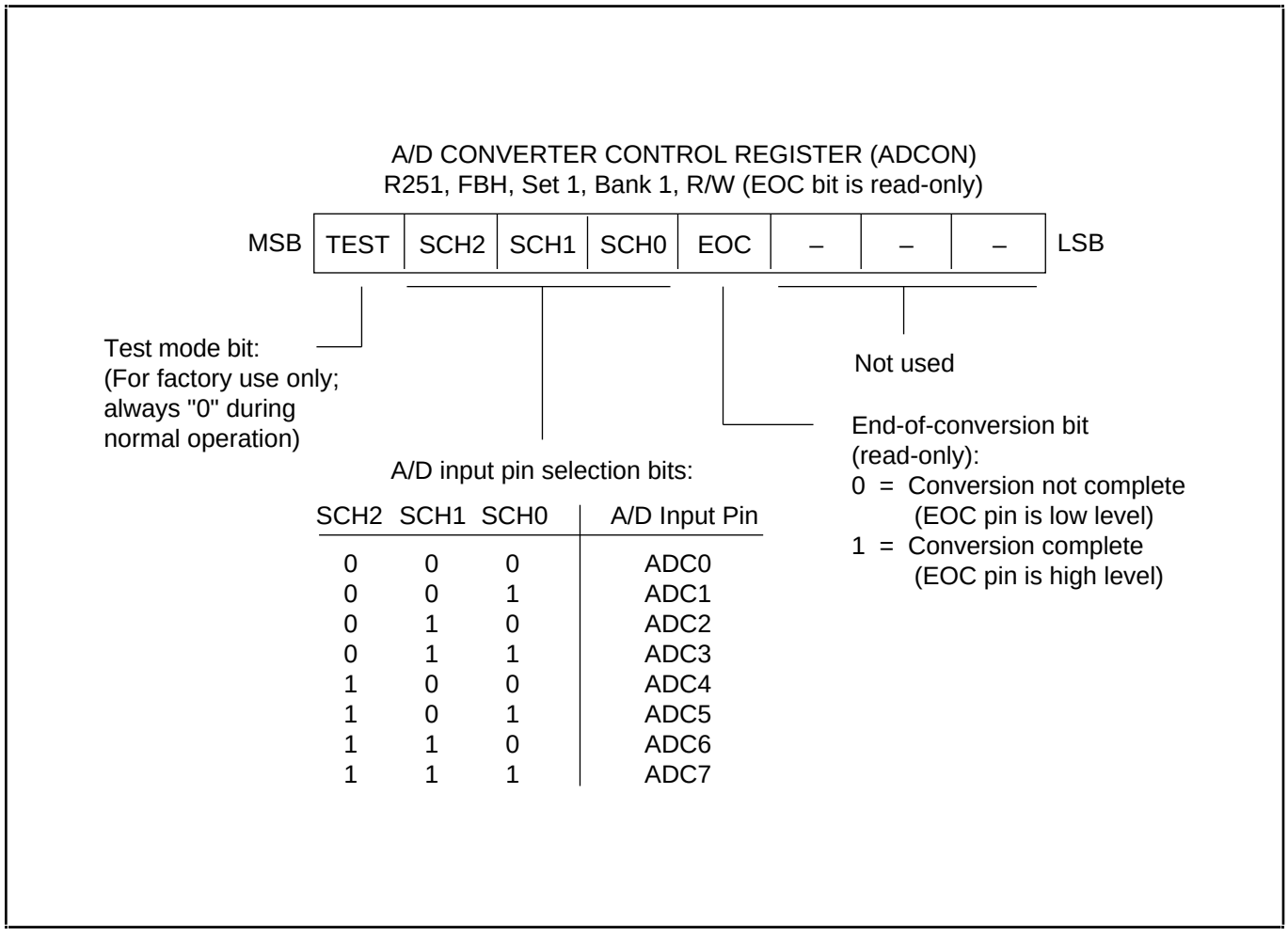


Figure 14–2. A/D Converter Control Register (ADCON)

Internal A/D Conversion Procedure

1. To enable the A/D converter for incoming analog data, you select one of the eight analog data input pins (ADC0–AD7) by writing the appropriate value to the ADCON register bits SCH0–SCH2.

2. Analog data can then be input within the acceptable voltage range (between AV_{SS} and AV_{REF}).

3. If input voltage > first reference voltage (1/2 AV_{REF}), the first conversion output is "1";
   If input voltage < first reference voltage (1/2 AV_{REF}), the first conversion output is "0".

4. The operation described in step 3 is then repeated eight times.
5. After 192 clocks (24 μs with an 8-MHz CPU clock) have elapsed, the converted digital values are loaded to the lower nibble of the output buffer ADOUT (set 1, bank 1, FAH) and the ADC module goes into an idle state.

NOTE

During the 192-clock conversion time, the EOC bit value in the ADCON register is "0". When the 8-bit conversion is completed, the EOC bit value is automatically set to "1". This makes it possible to monitor the progress of A/D conversions internally by software.

6. You can now read the converted digital value from the ADOUT register.
7. To perform another conversion, execute steps 1–7 again.

INTERNAL REFERENCE VOLTAGE LEVELS

In the ADC function block, the analog input voltage level is logically compared to a reference voltage. For the KS88C4116, the analog input level must be within the range AV_{SS} to AV_{REF}, where $AV_{REF} = V_{DD}$. Different reference voltage levels are generated internally along the resistor tree during the analog conversion process for each conversion step. The reference voltage level for the first bit conversion is always 1/2 AV_{REF}.

☞ Programming Tip — Sample A/D Conversion Program

This example show you how to program the A/D converter module. The program specifications are as follows:

— An A/D conversion operation occurs for each of the eight analog input channels. The value at the selected channel will be converted four times. The maximum and minimum conversion values are ignored; the result is the average of the remaining two values.

— The conversion result for each input channel will be loaded to ADC_0 through ADC_7.

The program conditions are:

- CPU clock = 8 MHz (conversion time is 24 μs /bit)
- V_{DD} = 5 V, AV_{REF} = V_{DD}, AV_{SS} = GND
- ADC input = 0 V to 5 V
- Use the register ADC_REG+AD_CHNL (57H) to select a specific input channel before calling the conversion subroutine for that channel.

For a decision flow diagram of this sample program, see Figure 14–3.

☞ **Programming Tip — Sample A/D Conversion Program (Continued)**

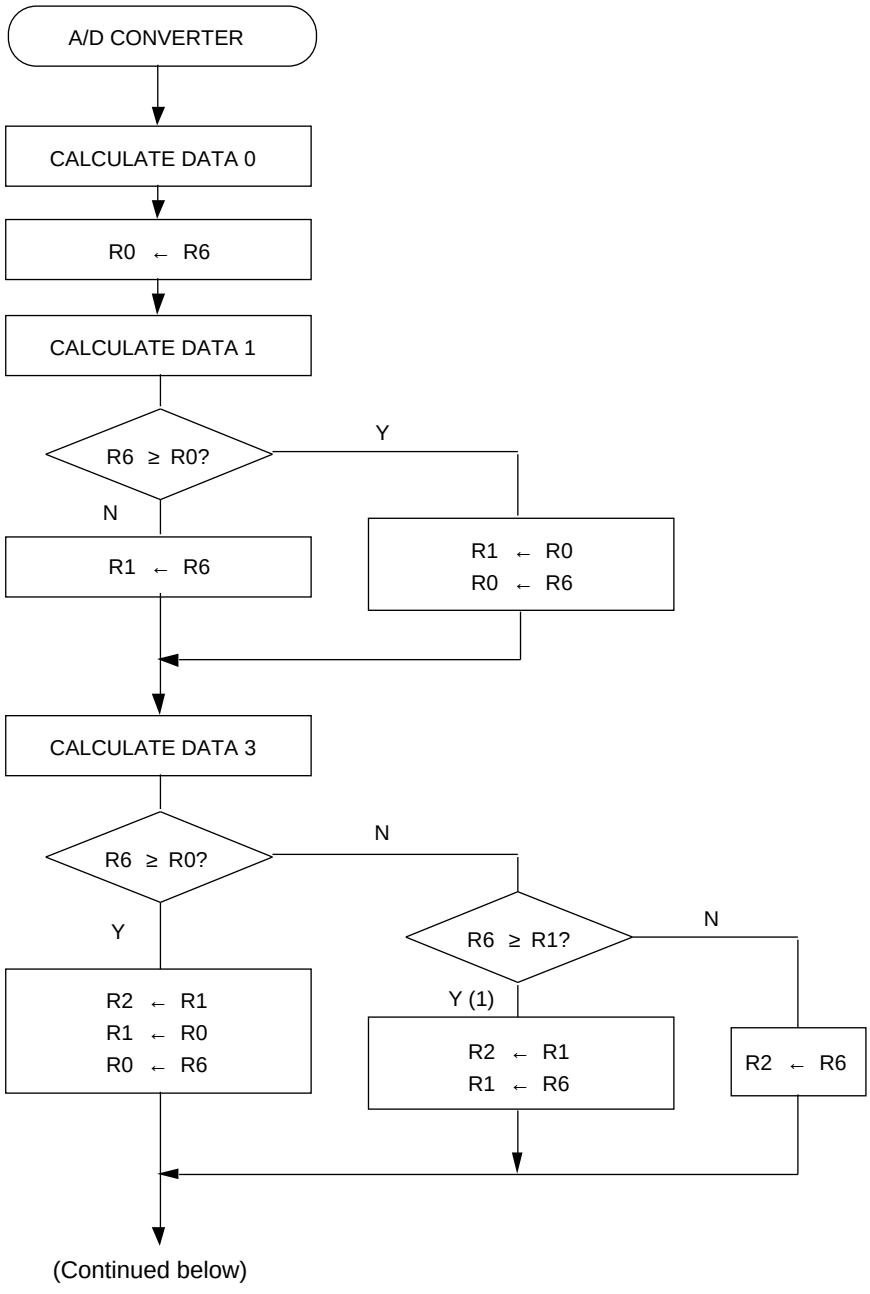


Figure 14–3. Decision Flow for A/D Converter Programming Tip

☞ **Programming Tip — Sample A/D Conversion Program (Continued)**

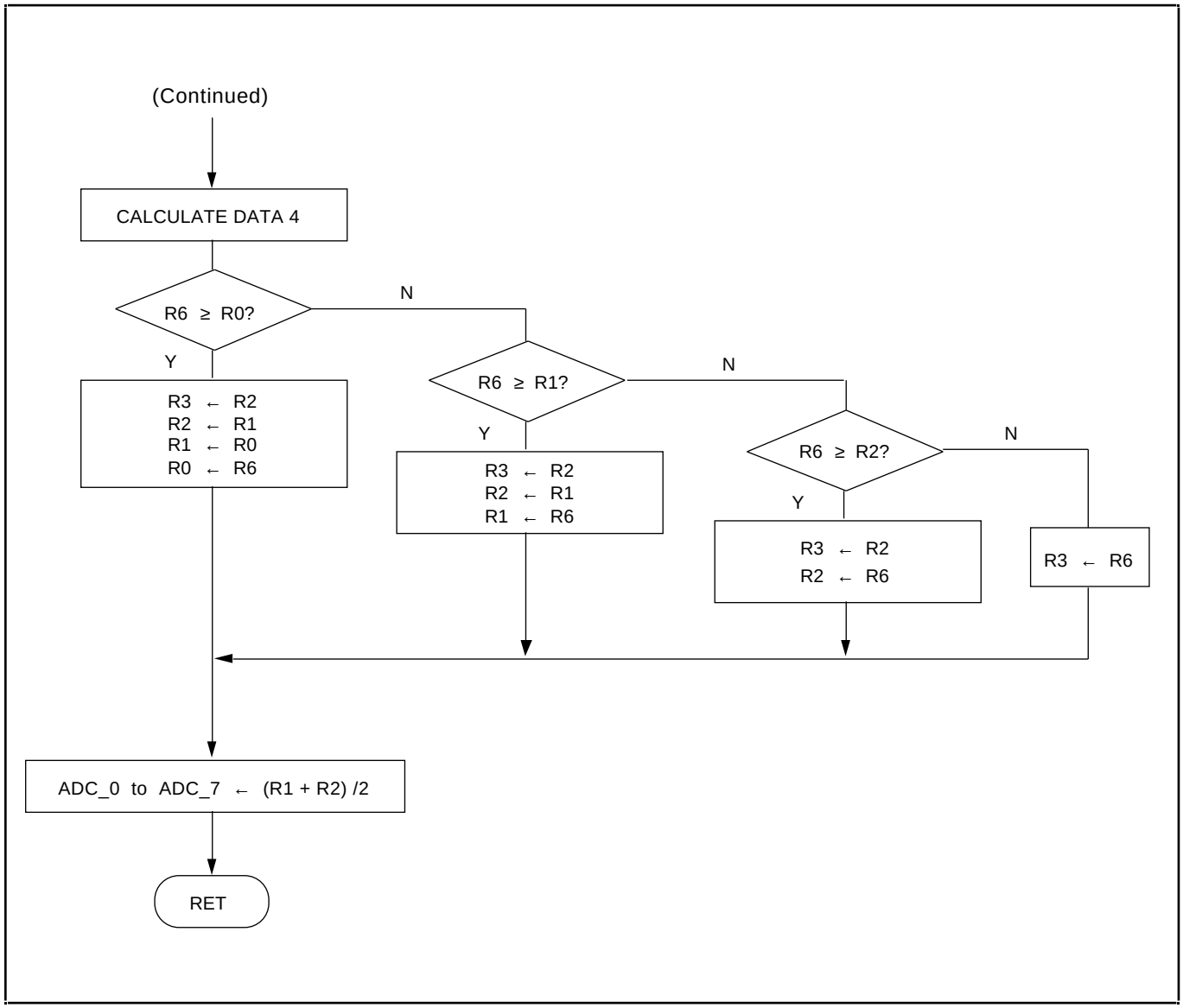


Figure 14–3. Decision Flow for A/D Converter Programming Tip (Continued)

☞ Programming Tip — Sample A/D Conversion Program (Continued)

```
ADC_REG   EQU     50H          ;  Page 0
                               ;  50H–57H is the working register area
AD_CHNL   EQU     7            ;  Target channels are ADC0–ADC7

ADC_0     EQU     58H          ;  A/D conversion result goes to these locations:
ADC_1     EQU     59H
ADC_2     EQU     5AH
ADC_3     EQU     5BH
ADC_4     EQU     5CH
ADC_5     EQU     5DH
ADC_6     EQU     5EH
ADC_7     EQU     5FH

AD_CONVERTER
          PUSH    PP
          PUSH    RP0
          PUSH    RP1

          CLR     PP           ;  Select page 0
          SRP     #ADC_REG
          SB1                  ;  Select bank 1

          CALL    AD_CONV      ;  Start 1st conversion
          LD      R0,R6

          CALL    AD_CONV      ;  Start 2nd conversion
          CP      R6,R0
          JR      ult,AD_00
          LD      R1,R0        ;  R6 ≥ R0
          LD      R0,R6
          JR      t,AD_10

AD_00:    LD      R1,R6        ;  R6 < R0

AD_10:    CALL    AD_CONV      ;  Start 3rd conversion
          CP      R6,R0
          JR      ult,AD_11
          LD      R2,R1        ;  R6 ≥ R0
          LD      R1,R0
          LD      R0,R6
          JR      t,AD_20
AD_11:    CP      R6,R1
          JR      ult,AD_12
          LD      R2,R1        ;  R6 ≥ R0
          LD      R1,R6
          JR      t,AD_20
AD_12:    LD      R2,R6        ;  R6 < R1
```

(Continued on the next page)

☞ **Programming Tip — Sample A/D Conversion Program (Concluded)**

```
AD_20:  CALL    AD_CONV                 ;  Start 4th conversion
        CP      R6,R0
        JR      ult,AD_21
        LD      R3,R2                   ;  R6 ≥ R1
        LD      R2,R1
        LD      R1,R0
        LD      R0,R6
        JR      t,AD_30
AD_21:  CP      R6,R1
        JR      ult,AD_22
        LD      R3,R2                   ;  R6 ≥ R1
        LD      R2,R1
        LD      R1,R6
        JR      t,AD_30
AD_22:  CP      R6,R2
        JR      ult,AD_23
        LD      R3,R2                   ;  R6 ≥ R2
        LD      R2,R6
        JR      t,AD_30
AD_23:  LD      R3,R6                   ;  R6 < R2

AD_30:  CLR     R0
        ADD     R1,R2
        ADC     R0,#0H
        DIV     RR0,#2H

        LD      #ADC_0[AD_CHNL],R1      ;  Final result
        SB0
        RET                             ;  End of AD_CONVERTER routine

AD_CONV: LD     ADCON,R7                ;  Start A/D conversion!
AD_WAIT: LD     R6,ADCON
        BTJRF   AD_WAIT,R6.3            ;  Is the conversion finished?
        NOP                             ;  Yes.
        LD      R6,ADOUT                ;  Load converted data to output register
        RET
```

note

15 EXTERNAL INTERFACE

OVERVIEW

The SAM8 architecture supports an external memory interface. Program and data memory areas in external devices can be accessed over the 16-bit multiplexed address/data bus.

Instruction code can be fetched, or data read, from external program memory space. If the external program memory is implemented in a RAM-type device, program code or data can also be written to external program memory.

The KS88C4400 microcontroller has a total of 80 pins. Of these 80 pins, up to 22 can be configured as an external interface that supports access to external memory and other peripheral devices.

Since memory addresses that are carried over the address/data bus are 16 bits long, up to 65536 bytes of memory space can be addressed. The entire 65536 bytes of data memory can be implemented externally using the ROM-less operating mode.

External data memory can be separated from the external program memory address space by defining a data memory (DM) and program memory (PM) signal line. PM output goes low when instructions are being fetched or when accessing the external program memory address space; DM output goes low whenever an external data memory location is addressed. In this way, external accesses to program memory and data memory are handled separately by the 16-bit address/data bus.

Address and data traffic is controlled by the address strobe (AS), data strobe (DS), read signal (DR) and write signal (DW). In addition, a port 3 pin (P3.7) can be used as an input line for WAIT signals being generated by an external device over the external bus.

You program the external interface by manipulating port control registers, and two system control registers: the system mode register (SYM) and the external memory timing register (EMT).

5 V must be applied to the EA pin at initialization and must remain at the 5-volt level during operation.

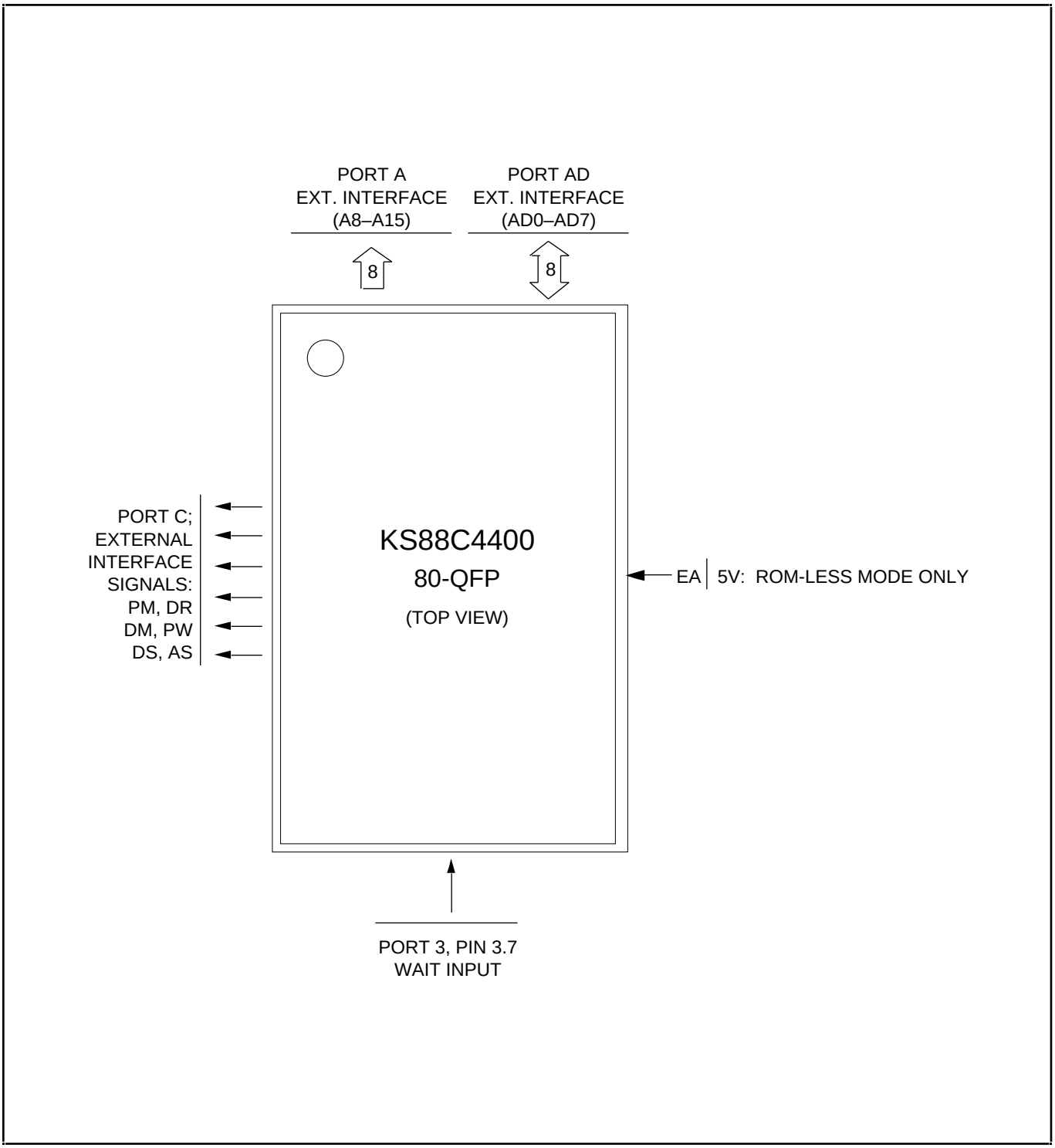


Figure 15–1. KS88C4400 Pin Functions for the External Interface

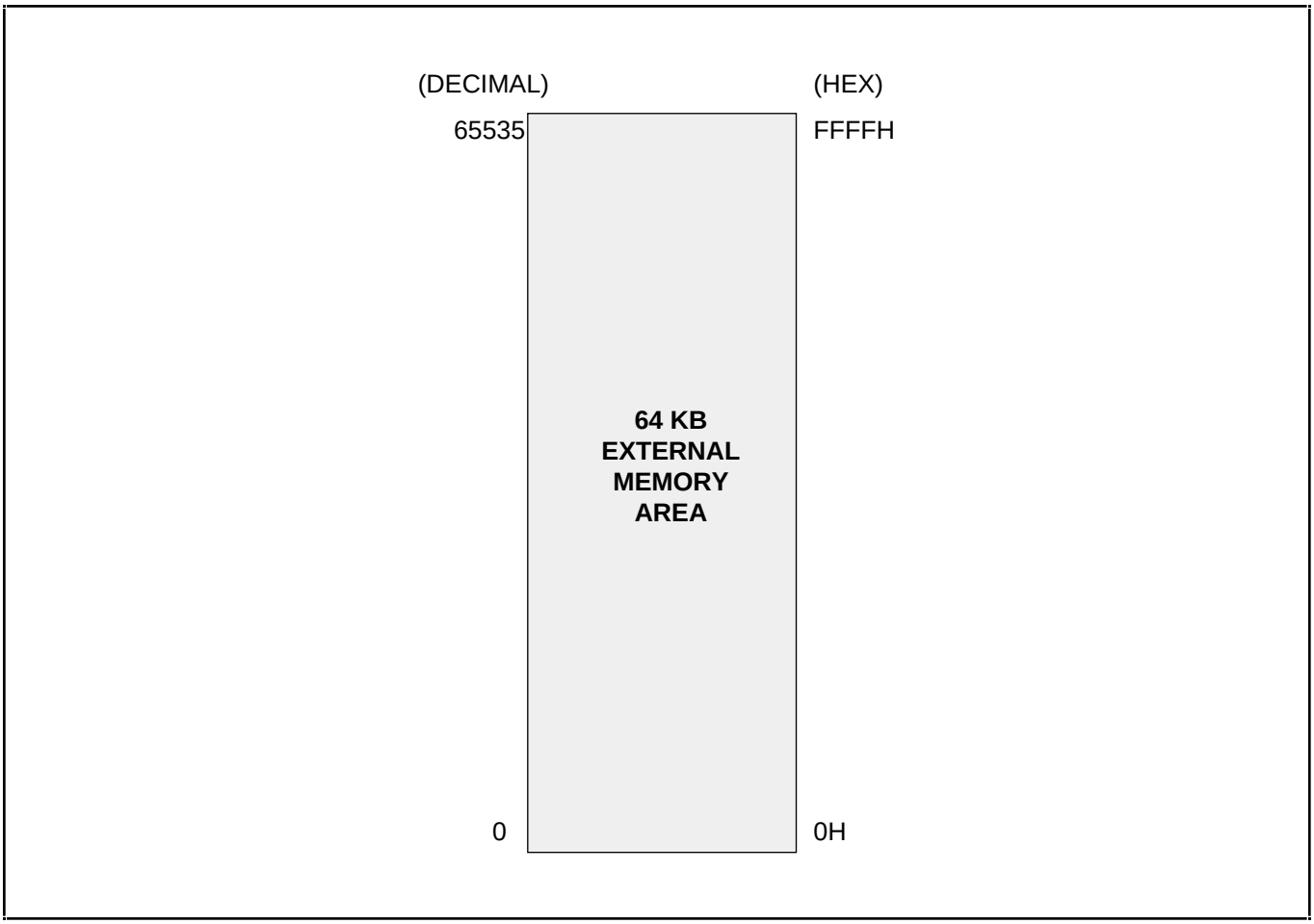


Figure 15–2. KS88C4400 Program Memory (ROM-less Operating Mode)

EXTERNAL INTERFACE CONTROL REGISTERS

This subsection presents an overview of the KS88C4400 system registers which are used to configure and control the external peripheral interface:

— System mode register (SYM, R222, DEH, set 1)

— External memory timing register (EMT, R254, FEH, set 1, bank 0)

Detailed descriptions of each of these registers can be found in Part I, Section 4, "Control Registers."

SYSTEM MODE REGISTER (SYM)

The system mode register SYM controls interrupt processing and also contains the enable bit (bit 7) for the 3-state external memory interface.

SYM is located at address DEH in set 1, and can be read or written by 1-bit and 8-bit instructions. When 3-stating is enabled, the lines of the external memory interface 'float' in a high-impedance state. 3-stating is commonly used multiprocessing applications that require a shared external bus.

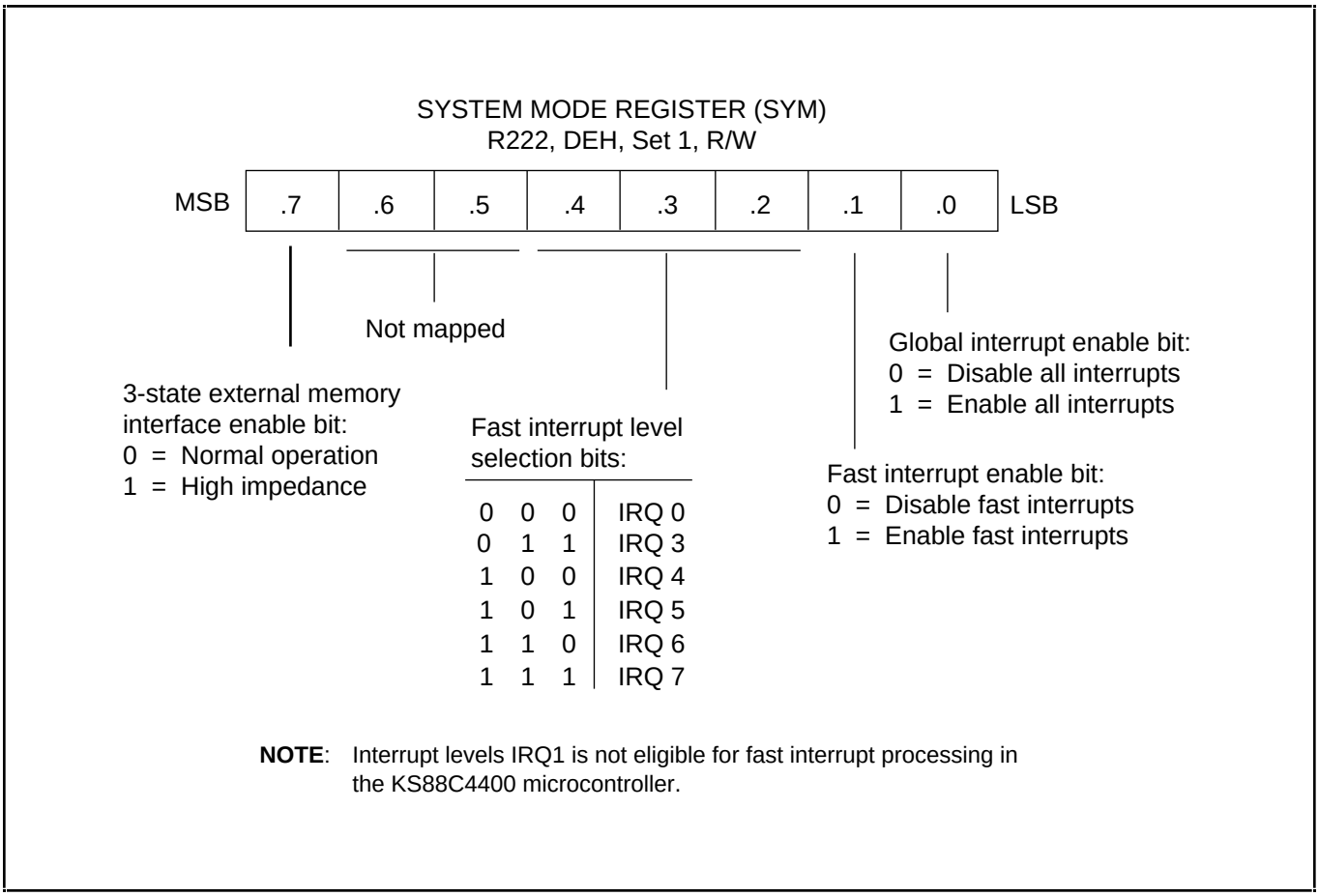


Figure 15–3. System Mode Register (SYM)

EXTERNAL MEMORY TIMING REGISTER (EMT)

The external memory timing register EMT is used to control bus-related operations for the external interface. Functions that are controlled by EMT bit settings are as follows:

EMT Bit(s)	Control Function
.7	External hardware WAIT function
.6	Slow memory timing enable
.5 and .4	Program memory wait cycle function
.3 and .2	Data memory wait cycle function
.1	Stack area selection (internal/external)

A reset clears the external WAIT function and stack area selection bits to "0". This disables the external peripheral WAIT function and selects the internal register file as the system stack area. Additionally, the program memory and data memory wait functions are both set to 3 wait cycles, which means that program execution time is slowed by a factor of two.

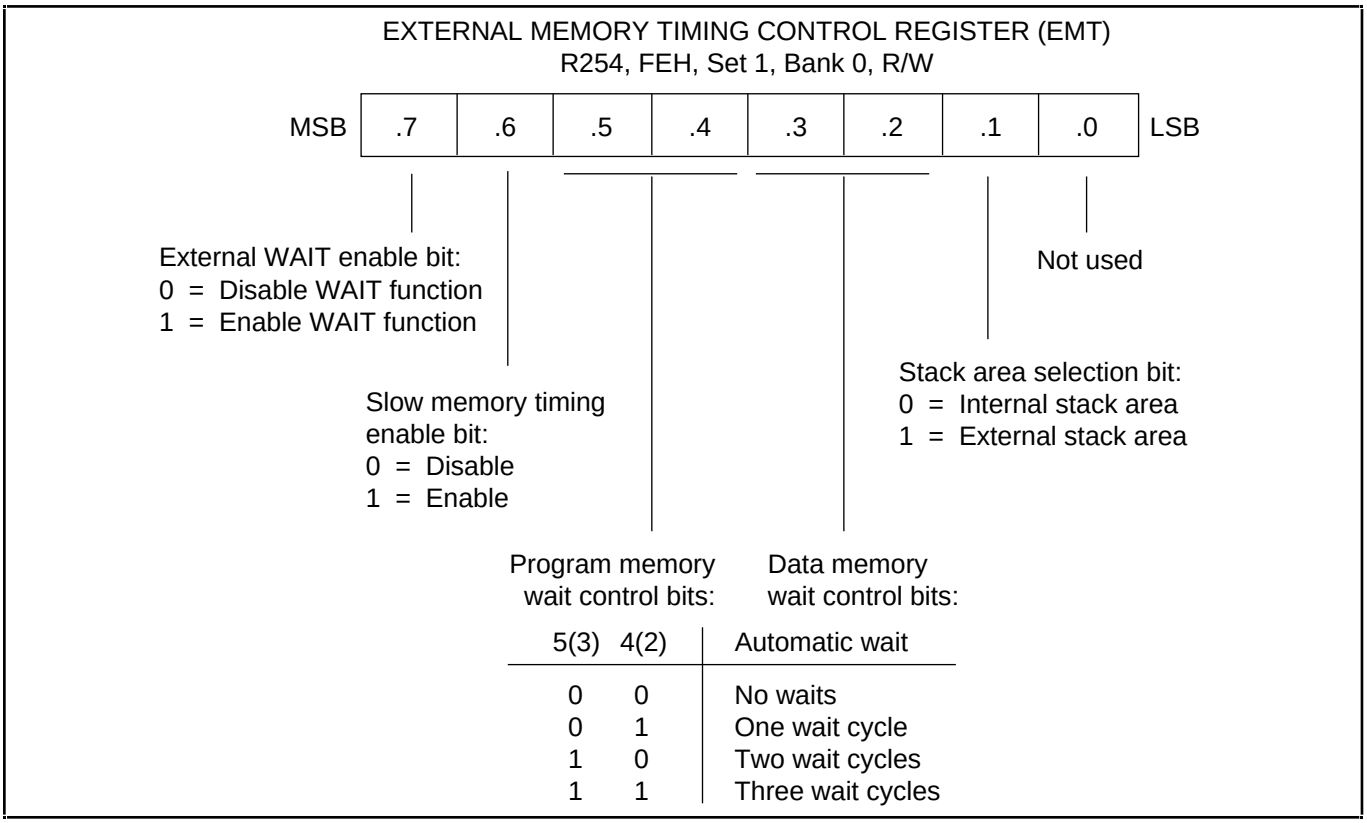


Figure 15–4. External Memory Timing Control Register (EMT)

HOW TO CONFIGURE THE EXTERNAL INTERFACE

The 3-state external memory interface is enabled or disabled by manipulating bit 7 of the system mode register SYM (R222, DEH). A reset clears SYM.7 to logic zero, disabling the high impedance levels of the interface bus lines and enabling the external interface.

CONFIGURING SEPARATE EXTERNAL PROGRAM AND DATA MEMORY AREAS

External program and data memory can be addressed either as a single combined memory space or as two separate spaces. If the program and data memory spaces are to be separated, this must be implemented logically by configuring the data memory select signal (DM) output pin. The DM pin's state goes active low to select data memory when one of the following instructions is executed:

— LDE (Load external data memory)
— LDED (Load external data memory and decrement)
— LDEI (Load external data memory and increment)
— LDEPD (Load external data memory with pre-decrement)
— LDEPI (Load external data memory with pre-increment)

If the stack area select bit in the EMT register (EMT.1) is set to "1", the system stack area is configured externally to the KS88C4400. In this case, the DM signal also goes active low whenever a CALL, POP, PUSH, RET, or IRET instruction is executed.

USING AN EXTERNAL SYSTEM STACK

SAM8 microcontrollers use the system stack to implement subroutine calls and returns, for interrupt processing, and for dynamic data storage. Stack operations are supported in either the internal register file or in externally configured data memory.

The PUSH and POP instructions support external system stack operations. The instructions PUSHUI, PUSHUD, POPUI, and POPUD support user-defined stack operations in the register file only.

After a reset, the stack pointer value is undetermined. For external stack operations, a 16-bit stack pointer value (in other words, both SPL and SPH) must be used.

An external stack holds return addresses for procedure calls and interrupts, as well as dynamically-generated data. The contents of the PC are saved on the external stack during a CALL instruction and restored during a RET instruction. During interrupts, the contents of the PC and the FLAGS register are saved on the external stack and then restored by the IRET instruction.

To select the external stack area option, bit 1 in the external memory timing register (EMT, FEH, set 1, bank 0) must be set to logic one. The instruction used to change the stack selection bit in the EMT register should not be immediately followed by an instruction that uses the stack, since this will cause indeterminate program flow. Also, interrupts should be disabled with a DI instruction before changing the stack selection bit.

Table 15–1. Control Register Overview for the External Memory Interface

Register	Location	Bit (s)	Description
SYM	DEH	7	External 3-state interface enable bit
EMT	FEH	1	External/internal stack selection control bit
		3, 2	Data memory wait time control bits
		5, 4	Program memory wait time control bits
		6	Slow memory timing control bit
		7	External hardware WAIT signal input enable bit
P3CONH	F4H	7, 6	External WAIT signal function when P3.7 is set to input mode

Table 15–2. External Interface Control Register Values After a RESET **(Normal Mode)**

Register Name	Mnemonic	Address		Bit Values After RESET (EA Pin is Low)							
		Dec	Hex	7	6	5	4	3	2	1	0
System Mode Register	SYM	R222	DEH	0	–	–	x	x	x	0	0
Port 3 Control Register (High Byte)	P3CONH	R244	F4H	0	0	0	0	0	0	0	0
External Memory Timing Register	EMT	R254	FEH	0	1	1	1	1	1	0	0

NOTE: A dash (–) indicates that the bit is not mapped; an 'x' means that the value is undefined after a RESET.

EXTERNAL BUS OPERATIONS

Typical data transfer timings for read and write cycles between the KS88C4400 and an external memory location are shown in Figures 15–5 and 15–6. The number of machine cycles required for external memory operations can vary from 6 to 12 external clock cycles, depending on the type of operation being performed.

The notation used to describe basic timing periods in these figures are machine cycles (Mn), timing states (Tn), and clock periods. All timing references are made with respect to the address strobe (AS) and the data strobe (DS). The clock waveform is shown for clarification only and does not have a specific timing relationship to the other signals.

Shared Bus Feature

Port A, port AD, and port C pins (if configured as additional address lines), can be placed in a high impedance state, allowing the KS88C4400 to share common resources with other bus masters. This feature is necessary for multiprocessor or related applications which required two or more devices to share the same external bus.

The 3-state memory interface enable bit in the system mode register (SYM.7) controls this function. When SYM.7 = "1", the 3-state function is enabled, the external interface lines are all set to high impedance, and control of the external bus is put under software control.

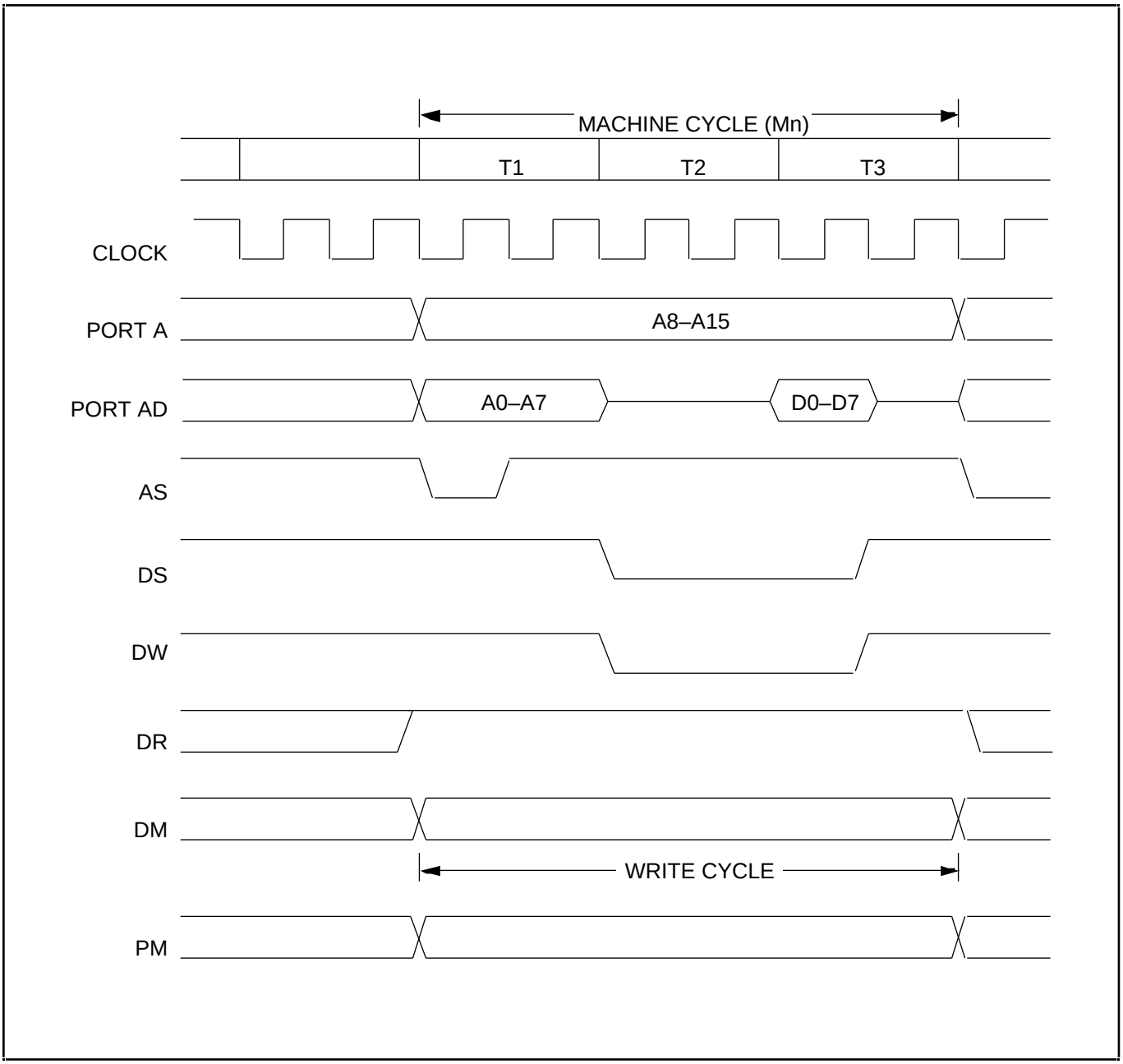


Figure 15–5. KS88C4400 External Bus Write Cycle Timing Diagram

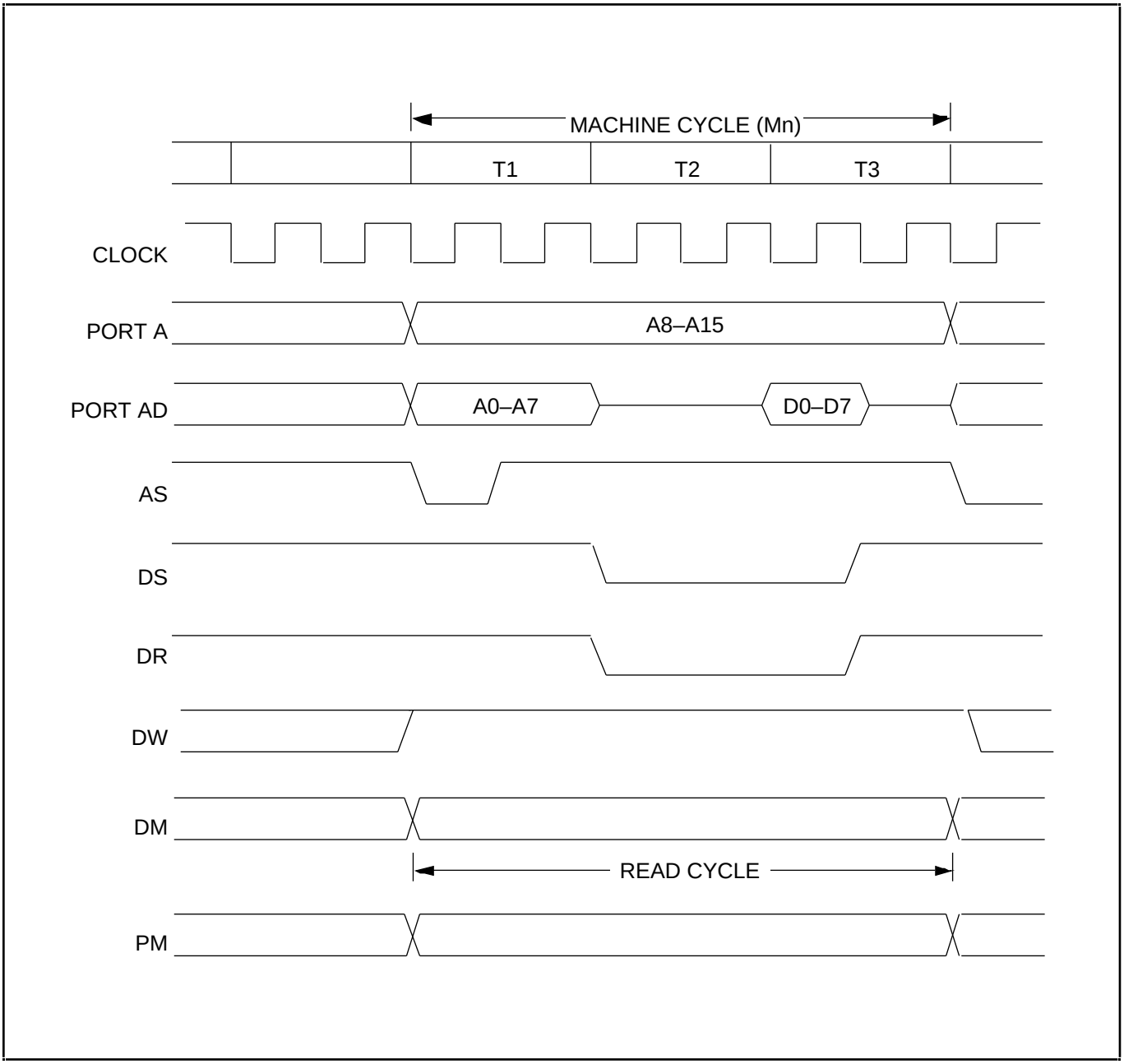


Figure 15–6. KS88C4400 External Bus Read Cycle Timing Diagram

EXTENDED BUS TIMING FEATURES

The SAM8 can accommodate slow memory access and cycle times by three different methods that give the user much flexibility in the types of memory available.

Software Programmable Wait States

The SAM8 can stretch the Data Strobe (DS) timing automatically by adding one, two, or three control and applies only to external memory cycles. Internal memory cycles still operate at the maximum rate.

The software has independent control over stretched Data Strobe for external memory (i.e., the software can set up one timing for program memory and a different timing for data memory). Thus, program and data memory may be made up of different kinds of hardware chips, each requiring its own timing.

Slow Memory Timing

Another feature of the SAM8 that is useful in interfacing with slow memories is the Slow Memory Timing option. When this option is enabled, the normal external memory timing is slowed by a factor of two (bus clock = CPU clock divided by two). All memory times for set-up, duration, hold, and access times are essentially doubled.

This feature can also be used with the programmed automatic wait states can still be used to stretch the Data Strobe time by one, two, or three internal clock times (not two, four, or six) when Slow Memory Timing is enabled.

Hardware Wait States

Still another SAM8 feature is an optional external WAIT input using port pin P3.7. The WAIT input function can be used with either or both of the above two features. Thus the Data Strobe width will have a minimum value determined by the number of programmed wait states selected and/ or by Slow Memory Timing.

The WAIT input provides the means to stretch it even further. The WAIT input is sampled each internal clock time and, if held low, can stretch the Data Strobe by adding one internal clock period to the Data Strobe time for an indefinite period of time.

All of the extended bus timing features are programmed by writing the appropriate bits in the External Memory Timing register.

Table 15–3. KS88C4400 External Memory Interface Signal Descriptions

Signal Name	Symbol	Pin	Active Level	Description
Address strobe	AS	19	Low	An address strobe is pulsed once at the start of each machine cycle. Addresses for external program memory or data memory transfers are output at ports A and AD at the trailing edge of AS.
Data strobe	DS	18	Low	The data strobe provides the timing signal for each external memory data transfer to or from port 1.
Read	DR	15	Low	DR determines the data transfer direction for external memory operations.
Write	DW	17	Low	DW is low when writing to external program memory or data memory locations, and is high for all other operations.
Memory select	DM	16	Low	When it is low, DM selects data memory.
	PM	14	Low	When it is low, PM selects program memory.
External hardware wait signal	WAIT	31	Low	This pin is sampled at each rising edge of the CPU clock. If it is held low, it can "stretch" the data strobe indefinitely by adding one clock period to the DS valid time.

NOTE: If bit 7 of the SYM register is high level, and assuming the external memory interface is configured, the AS, DS, DR, DW and DM signals, as well as port A and port AD, will be set to high impedance state. This causes the external interface signals to 'float.'

ADDRESS AND DATA STROBES

Address Strobe (AS)

All external memory transactions start when the KS88C4400 drives the address strobe (AS) active low and then sends it high. The leading edge of the address strobe validates the read and write line (DW), the data memory line (DM), and addresses that are output at ports A and AD.

Addresses output at port AD remain valid only during the T1 phase of a machine cycle and typically need to be latched using AS. Port A address outputs remain stable throughout the machine cycle (T1–T3).

Data Strobe (DS)

The KS88C4400 uses the data strobe (DS) to time the actual transfer of data over the external memory interface.

For a write operation (DW = "0"), a DS low signal indicates that valid data is on the port AD AD0–AD7 lines. For a read operation (DR = "0") the address/data bus is placed in a high impedance state before driving the data strobe active low so that the addressed device can put its data on the bus. The KS88C4400 then samples this data before it sends the data strobe high.

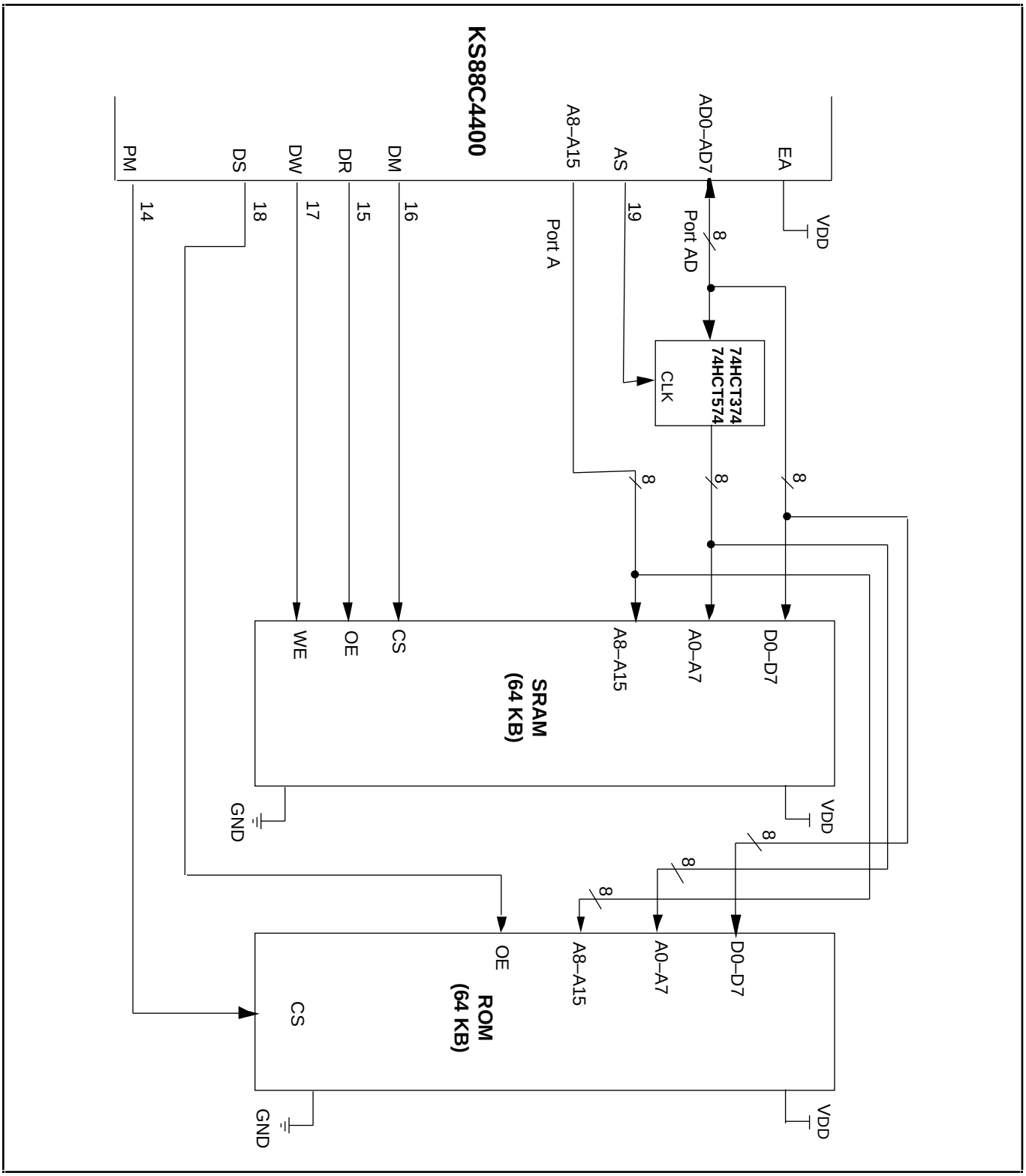


Figure 15–7. External Interface Function Diagram (KS88C4400, SRAM, EPROM, EEPROM)

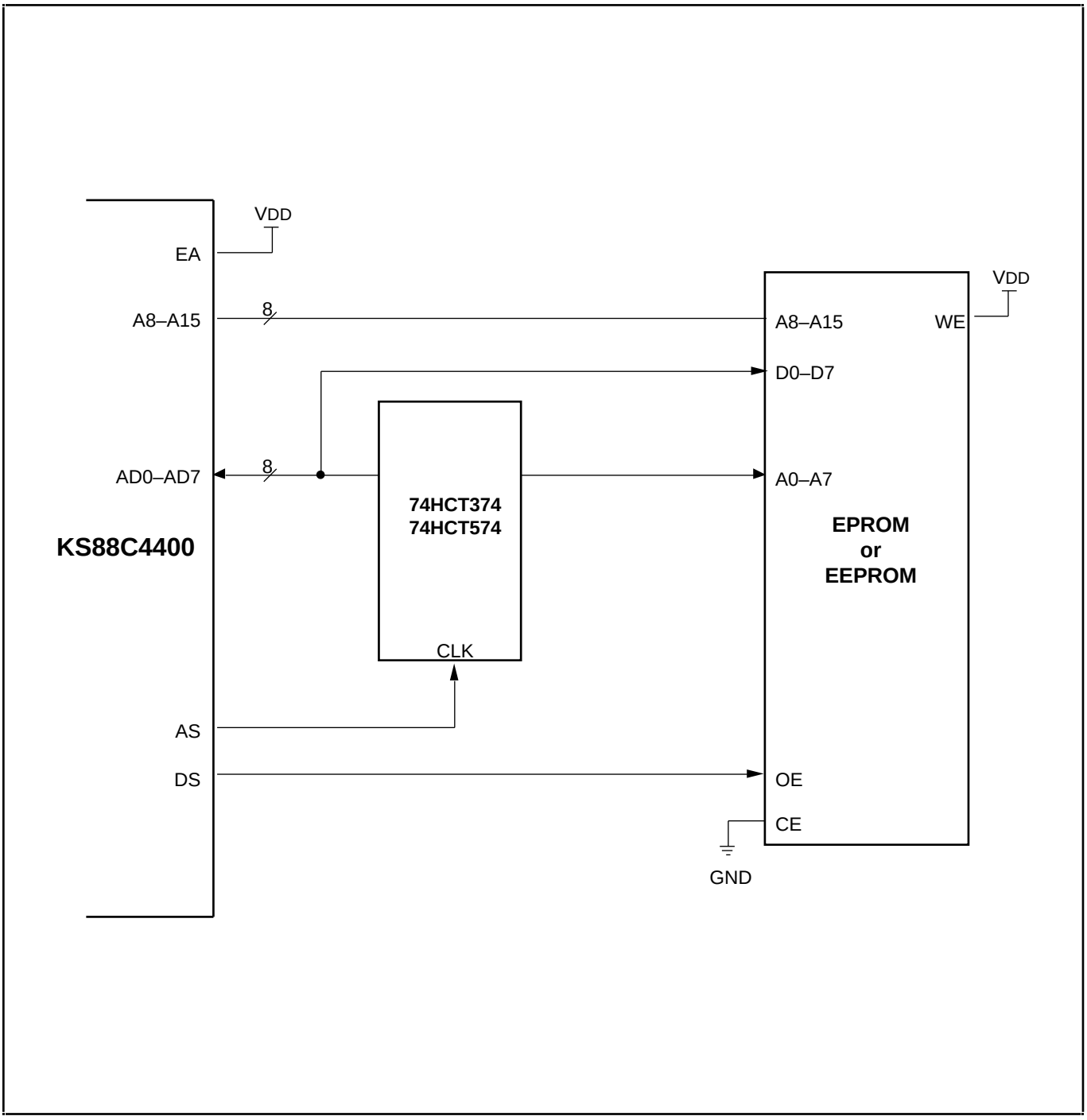


Figure 15–8. External Interface Function Diagram (External ROM Only)

SAM8 INSTRUCTION EXECUTION TIMING DIAGRAMS

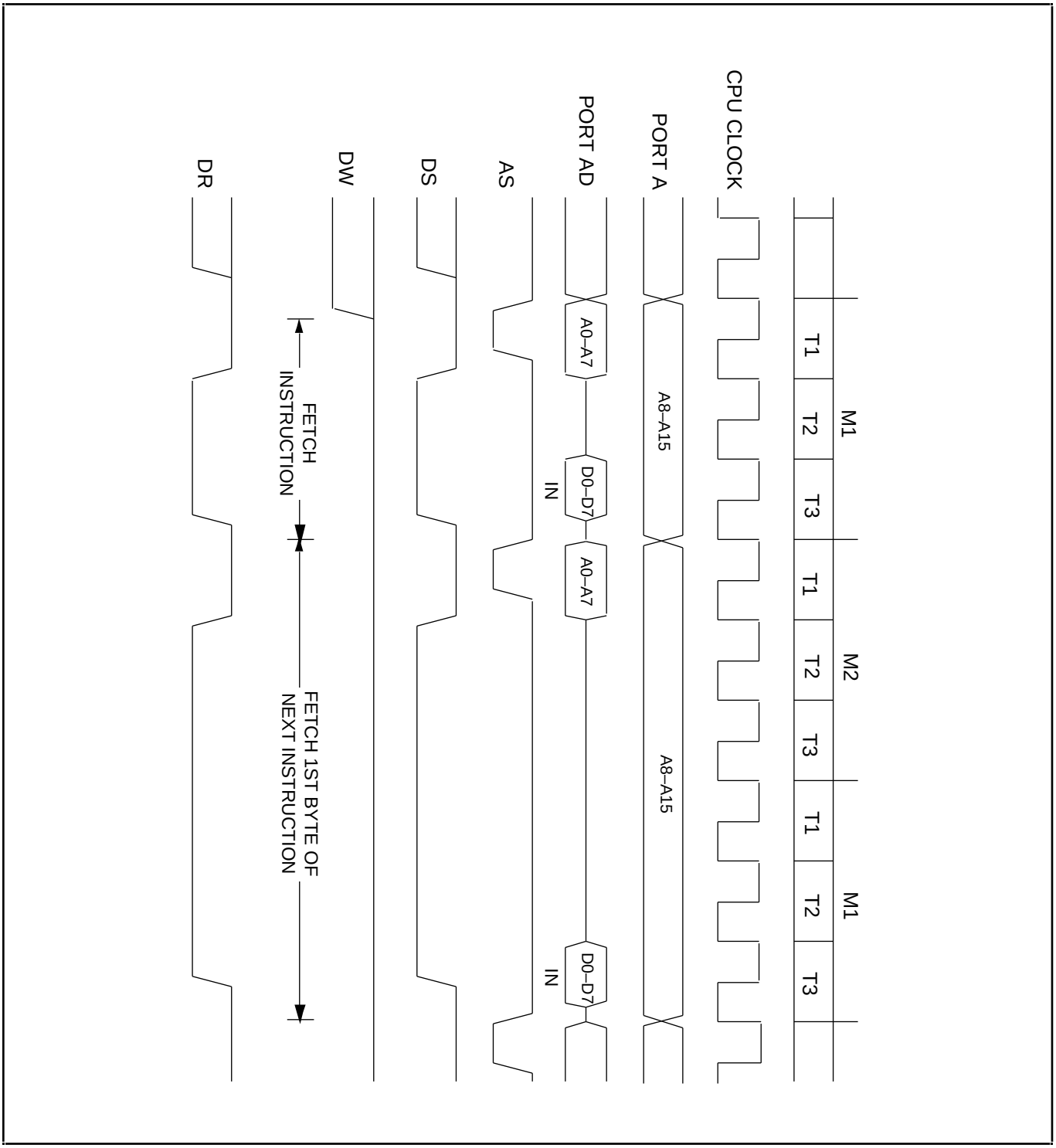


Figure 15–9. External Bus Timing Diagram for 1-Byte fetch Instructions

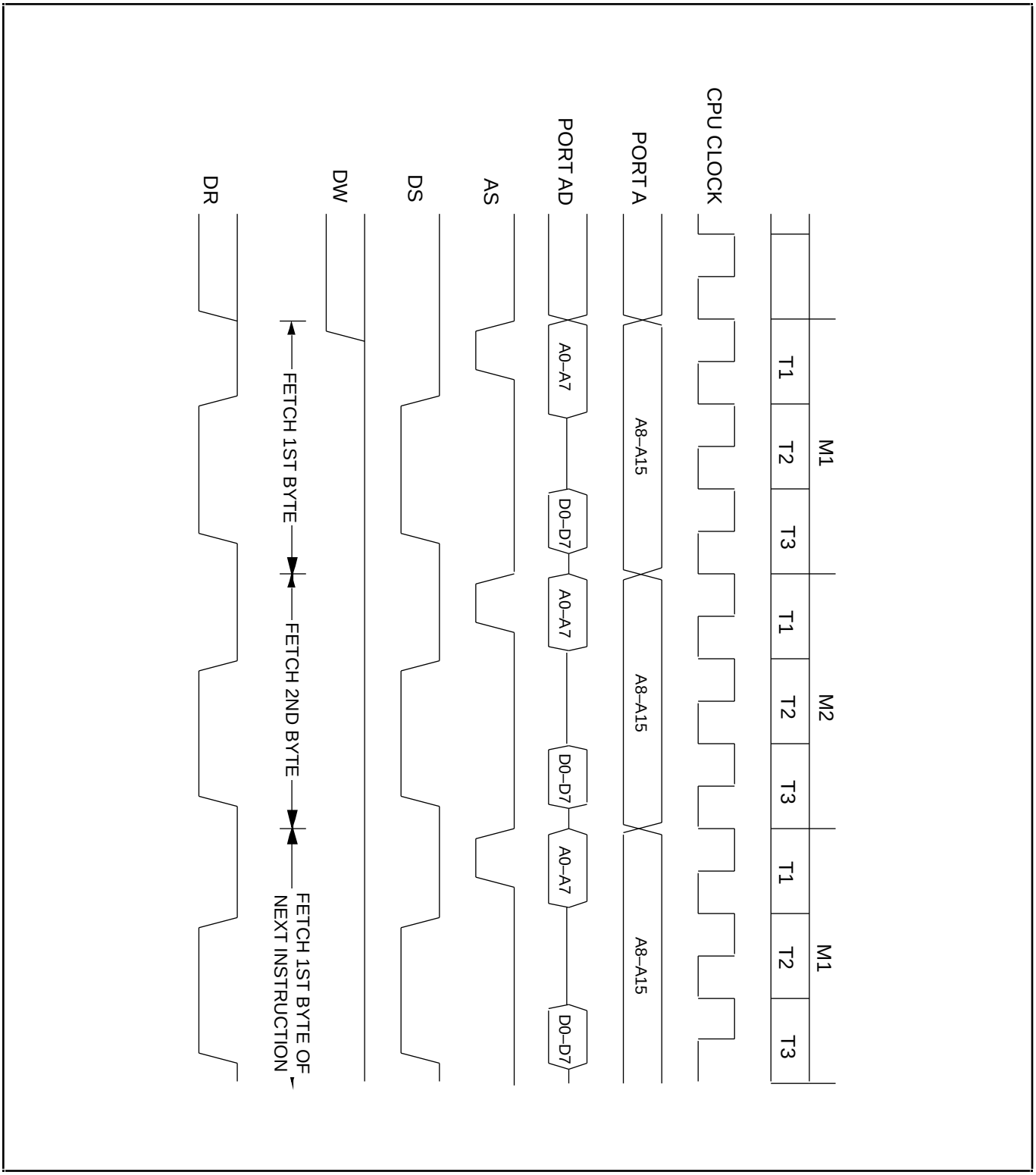


Figure 15–10. External Bus Timing Diagram for 2-Byte fetch Instructions

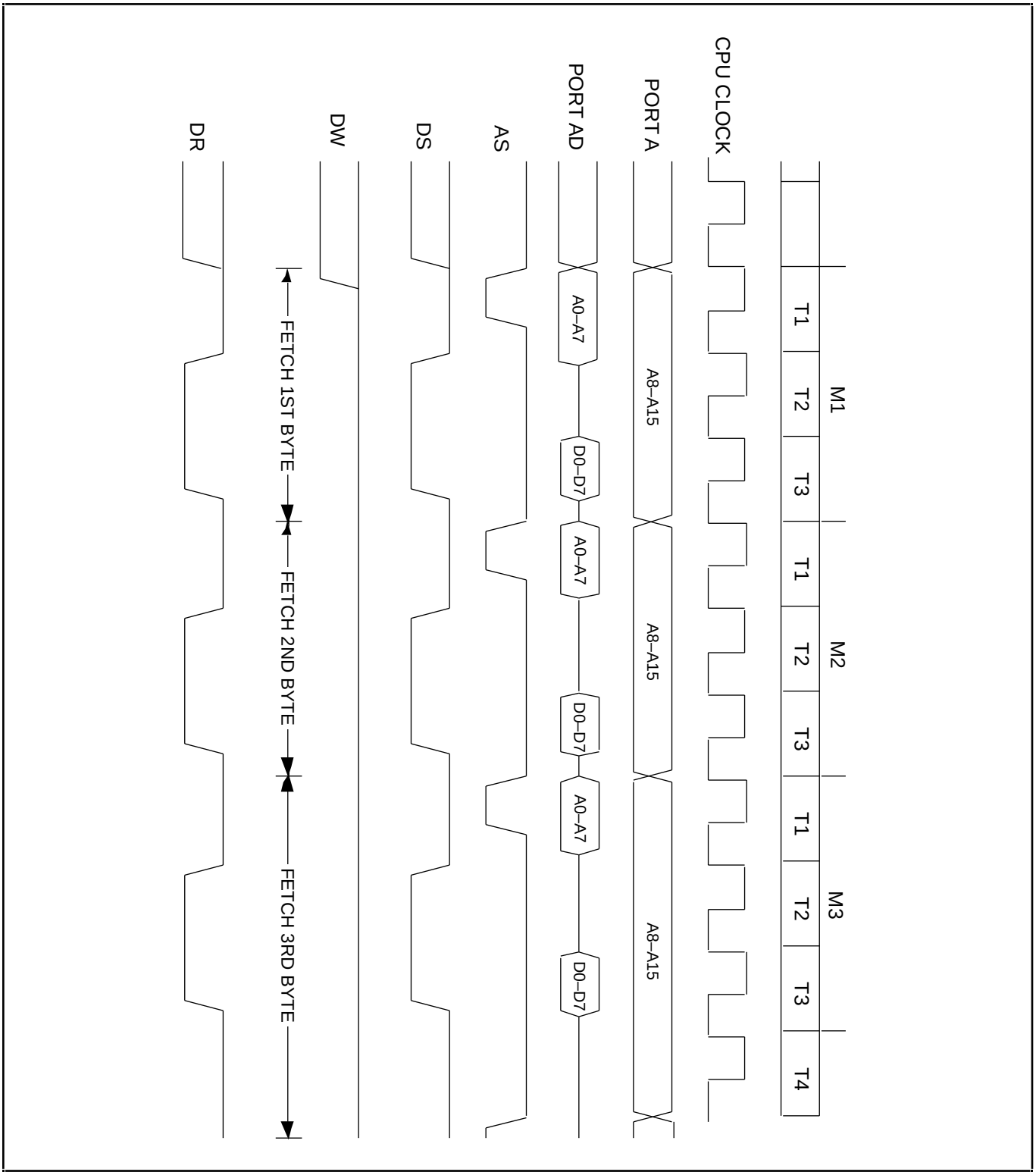


Figure 15–11. External Bus Timing Diagram for 3-Byte fetch Instructions

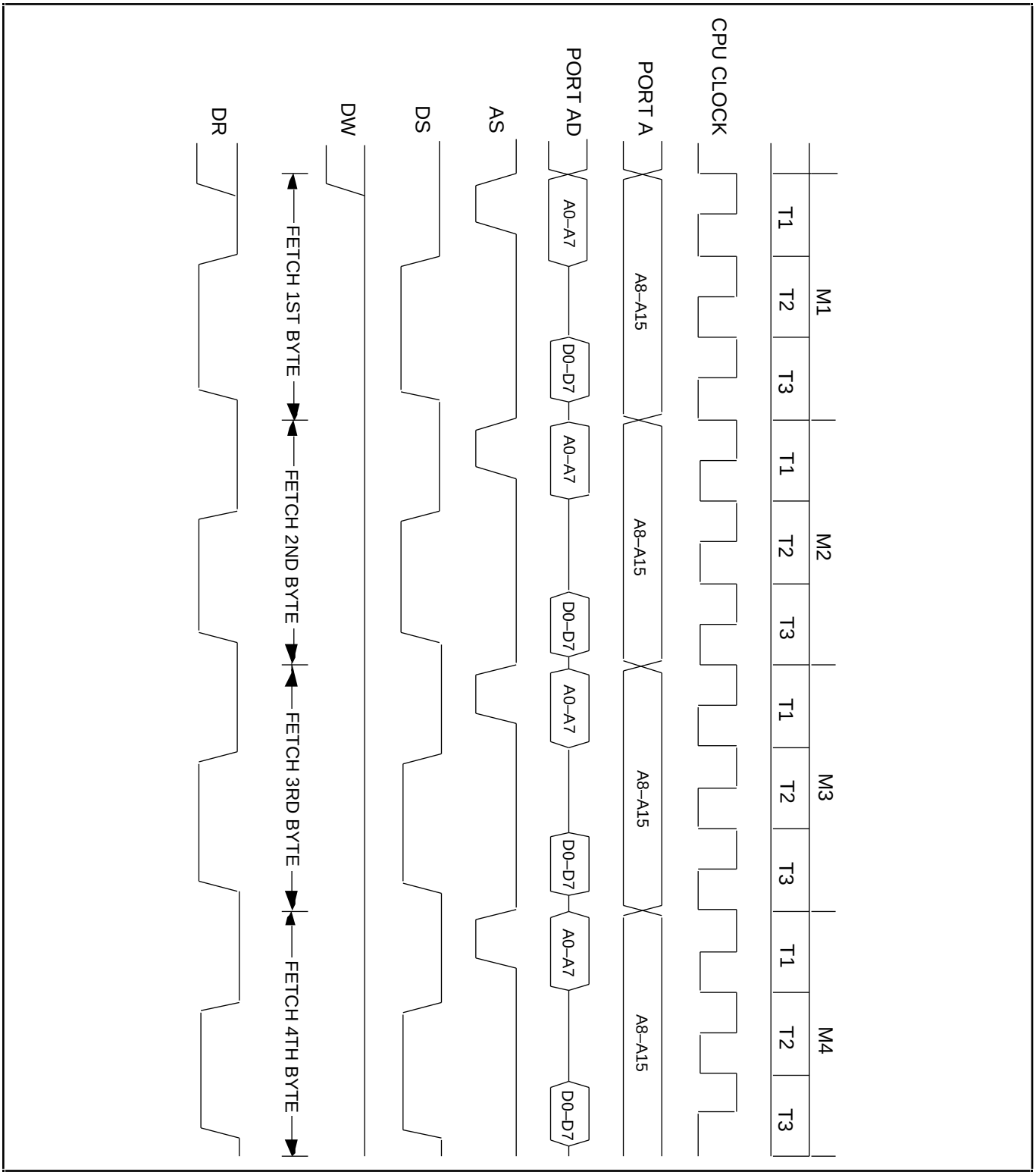


Figure 15–12. External Bus Timing Diagram for 4-Byte fetch Instructions

NOTE

16 Electrical Data

In this section, KS88C4400 electrical characteristics are presented in tables and graphs. The information is arranged in the following order:

— Absolute maximum ratings
— DC electrical characteristics
— AC electrical characteristics
— Input timing for external interrupts (ports 3 and 4)
— Input timing for RESET
— I/O capacitance
— Data retention supply voltage in Stop mode
— Stop mode release timing initiated by RESET
— A./D Converter Electrical Characteristics
— Serial port timing characteristics in mode 0 (10 MHz)
— Serial clock waveform
— Serial port timing in mode 0 (shift register mode)
— External memory timing characteristics (10 MHz)
— External memory read and write timing
— Recommended A/D converter circuit for highest absolute accuracy
— Main oscillator frequency (f_{OSC1})
— Main oscillator clock stabilization time (t_{ST1})
— Clock timing measurement at X_{IN}
— Suboscillator clock stabilization time (t_{ST2})
— Characteristic curves

Table 16–1. Absolute Maximum Ratings

(T_A = 25°C)

Parameter	Symbol	Conditions	Rating	Unit
Supply voltage	V_{DD}		– 0.3 to + 7.0	V
Input voltage	V_{I1}	Port 6 only (open-drain)	– 0.3 to + 10	V
	V_{I2}	All ports except port 6	– 0.3 to V_{DD} + 0.3	
Output voltage	V_O		– 0.3 to V_{DD} + 0.3	V
Output current high	I_{OH}	One I/O pin active	– 18	mA
		All I/O pins active	– 60	
Output current low	I_{OL}	One I/O pin active	+ 30	mA
		Total pin current for ports 0, 2, 3, 4, 6	+ 100	
		Total pin current for ports 1 and 5	+ 200	
Operating temperature	T_A		– 20 to + 85	°C
Storage temperature	T_{STG}		– 65 to + 150	°C

Table 16–2. D.C. Electrical Characteristics

(T_A = – 20°C to + 85°C, V_{DD} = 4.5 V to 6.0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Input high voltage	V_{IH1}	All input pins except V_{IH2}	0.8 V_{DD}	–	V_{DD}	V
	V_{IH2}	X_{IN}	V_{DD} – 0.5			
Input low voltage	V_{IL1}	All input pins except V_{IL2}	–	–	0.2 V_{DD}	V
	V_{IL2}	X_{IN}			0.4	
Output high voltage	V_{OH1}	V_{DD} = 4.5 V to 6.0 V I_{OH} = – 1 mA Port AD only	V_{DD} – 1.0	–	–	V
	V_{OH2}	V_{DD} = 4.5 V to 6.0V I_{OH} = – 200 µA All output pins except port AD	V_{DD} – 1.0			

Table 16–2. D.C. Electrical Characteristics (Continued)

(T_A = – 20°C to + 85°C, V_{DD} = 4.5 V to 6.0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Output low voltage	V_{OL1}	V_{DD} = 4.5 V to 6.0 V I_{OL} = 2 mA All output pins except port 5	–	–	0.4	V
	V_{OL2}	V_{DD} = 4.5 V to 6.0 V I_{OL} = 1.5 mA Port 5				
Input high leakage current	I_{LIH1}	V_{IN} = V_{DD} All input pins except X_{IN}	–	–	3	µA
	I_{LIH2}	V_{IN} = V_{DD} X_{IN}			20	
Input low leakage current	I_{LIL1}	V_{IN} = 0 V All input pins except X_{IN}, and RESET	–	–	– 3	µA
	I_{LIL2}	V_{IN} = 0 V X_{IN}			– 20	
Output high leakage current	I_{LOH1}	V_{OUT} = V_{DD} All output pins except for port 6	–	–	5	µA
	I_{LOH2}	Port 6 (open-drain) V_{OUT} = 9 V			20	
Output low leakage current	I_{LOL}	V_{OUT} = 0 V	–	–	– 5	µA
Pull-up resistor	R_{L1}	V_{IN} = 0 V; V_{DD} = 5 V ± 10% Ports 0, 1, 4, 5, and RxD	30	56	80	KΩ
	R_{L2}	V_{IN} = 0 V; V_{DD} = 5 V ± 10% RESET only	120	220	320	
Supply current (1)	I_{DD1}	V_{DD} = 5 V ± 10% 18 MHz crystal oscillator	–	32	50	mA
		V_{DD} = 5 V ± 10% 12 MHz crystal oscillator		22		
	I_{DD2}	Idle mode: V_{DD} = 5 V ± 10% 18 MHz crystal oscillator		12	25	
		Idle mode: V_{DD} = 5 V ± 10% 12 MHz crystal oscillator		10		
	I_{DD3}	Stop mode; V_{DD} = 5 V ± 10%		18	50	µA

NOTE: Supply current does not include current drawn through internal pull-up resistors or external output current loads.

Table 16–3. A.C. Electrical Characteristics

(T_A = – 20°C to + 85°C, V_{DD} = 4.5 V to 6.0V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Interrupt input high, low width	t_{INTH}, t_{INTL}	P3.0–P3.3, P4.0–P4.7	3	–	–	t_{CPU}
RESET input low width	t_{RSL}	Input	22	–	–	t_{CPU}

NOTES:
1. The unit t_{CPU} means one CPU clock period.
2. The oscillator frequency is the same as CPU clock frequency.

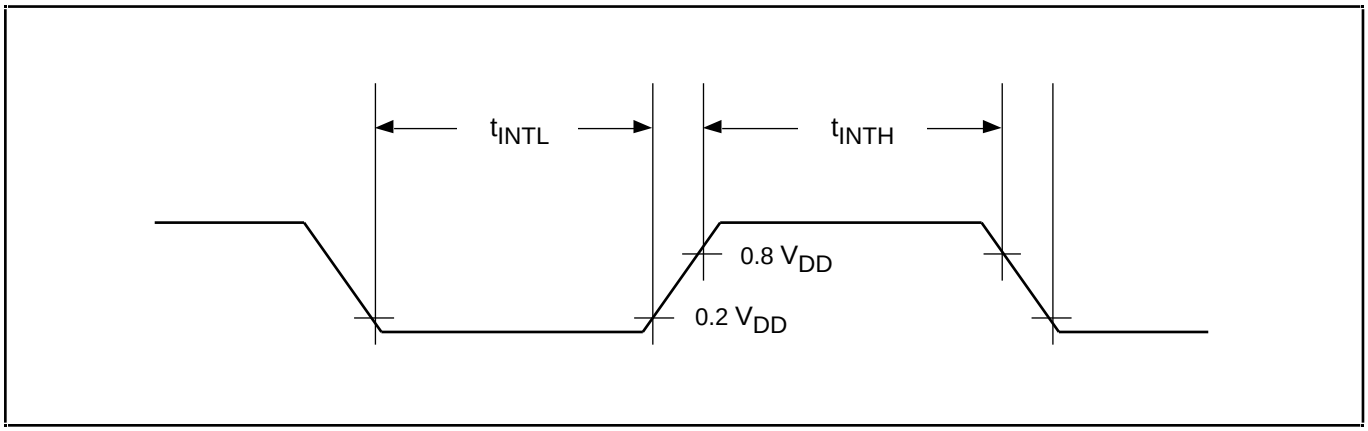


Figure 16–1. Input Timing for External Interrupts (Ports 3 and 4)

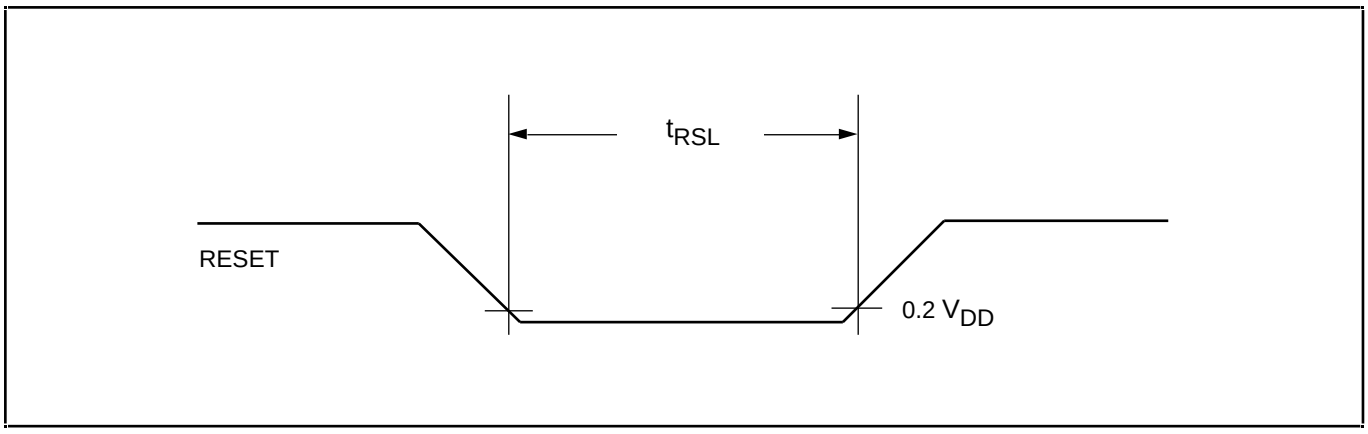


Figure 16–2. Input Timing for RESET

Table 16–4. Input/Output Capacitance

(T_A = – 20°C to + 85°C, V_{DD} = 0 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 16–5. Data Retention Supply Voltage in Stop Mode

(T_A = – 20°C to + 85°C)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Data retention supply voltage	V_{DDDR}		2	–	6	V
Data retention supply current	I_{DDDR}	V_{DDDR} = 2 V	–	–	5	µA

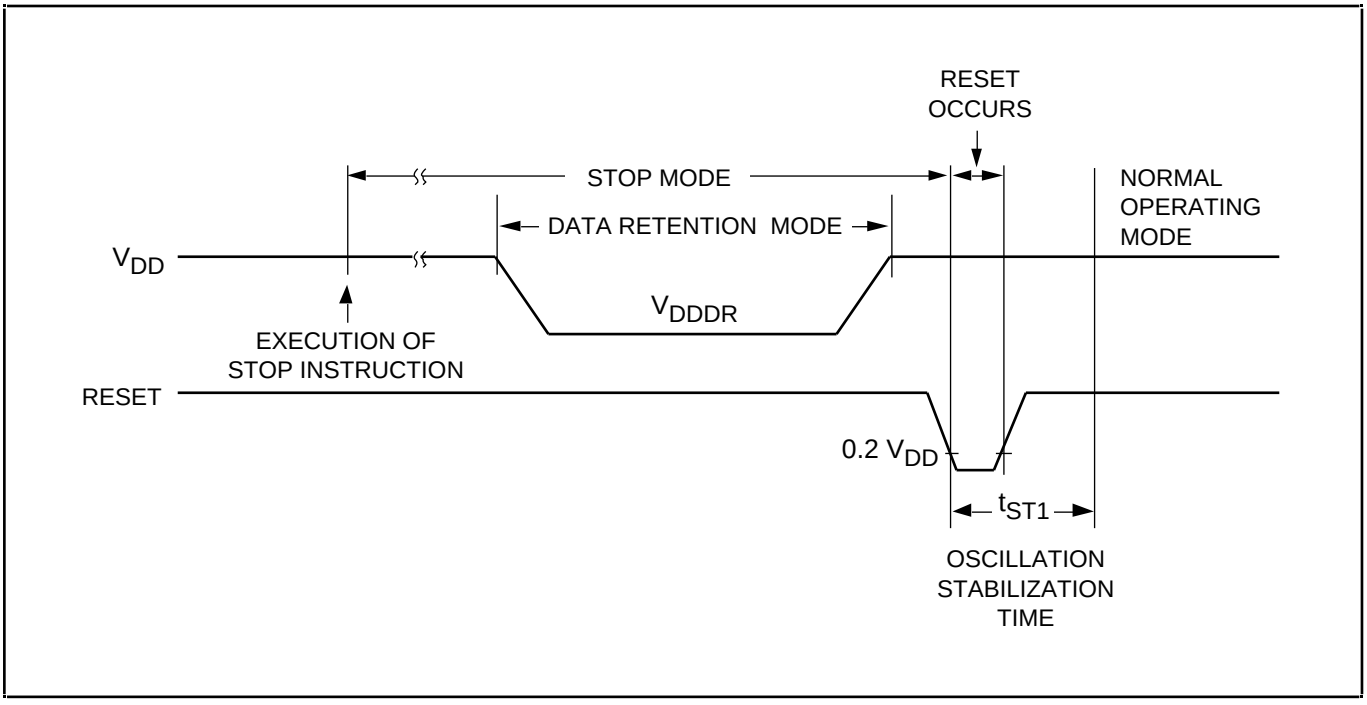


Figure 16–3. Stop Mode Release Timing Initiated by RESET

Table 16–6. A/D Converter Electrical Characteristics

(T_A = – 20°C to + 85°C, V_{DD} = 4.5 V to 6.0 V, V_{SS} = 0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Resolution			8	8	8	bit
Absolute accuracy (1)		V_{DD} = 5.12 V CPU clock = 18 MHz AV_{REF} = 5.12 V AV_{SS} = 0 V	–	–	| 3 |	LSB
Conversion time (2)	t_{CON}		t_{CPU} × 192 (3)	–	–	µs
Analog reference voltage	AV_{REF}		2.56	–	V_{DD}	V
Analog ground	AV_{SS}		V_{SS}	–	–	V
Analog input voltage	V_{IAN}		AV_{SS}	–	AV_{REF}	V
Analog input impedance	R_{AN}		2	–	–	MΩ

NOTES:
1. Excluding quantization error, absolute accuracy equals ± 1/2 LSB.
2. 'Conversion time' is the time required from the moment a conversion operation starts until it ends.
3. t_{CPU} is the CPU clock period.

Table 16–7. Serial Port Timing Characteristics in Mode 0 (10 MHz)

(T_A = – 20°C to + 85°C, V_{DD} = 4.5 V to 6.0V, V_{SS} = 0 V)

Parameter	Symbol	Min	Typ	Max	Unit
Serial port clock cycle time	t_{SCK}	500	t_{CPU} × 6	700	ns
Output data setup to clock rising edge	t_{S1}	300	t_{CPU} × 5	–	
Clock rising edge to input data valid	t_{S2}	–	–	300	
Output data hold after clock rising edge	t_{H1}	50	t_{CPU}	–	
Input data hold after clock rising edge	t_{H2}	0	–	–	
Serial port clock high, low width	t_{HIGH}, t_{LOW}	200	t_{CPU} × 3	400	

NOTES:
1. All times are in ns and assume a 10 MHz input frequency.
2. The unit t_{CPU} means one CPU clock period.
3. The oscillator frequency is identical to the CPU clock frequency.

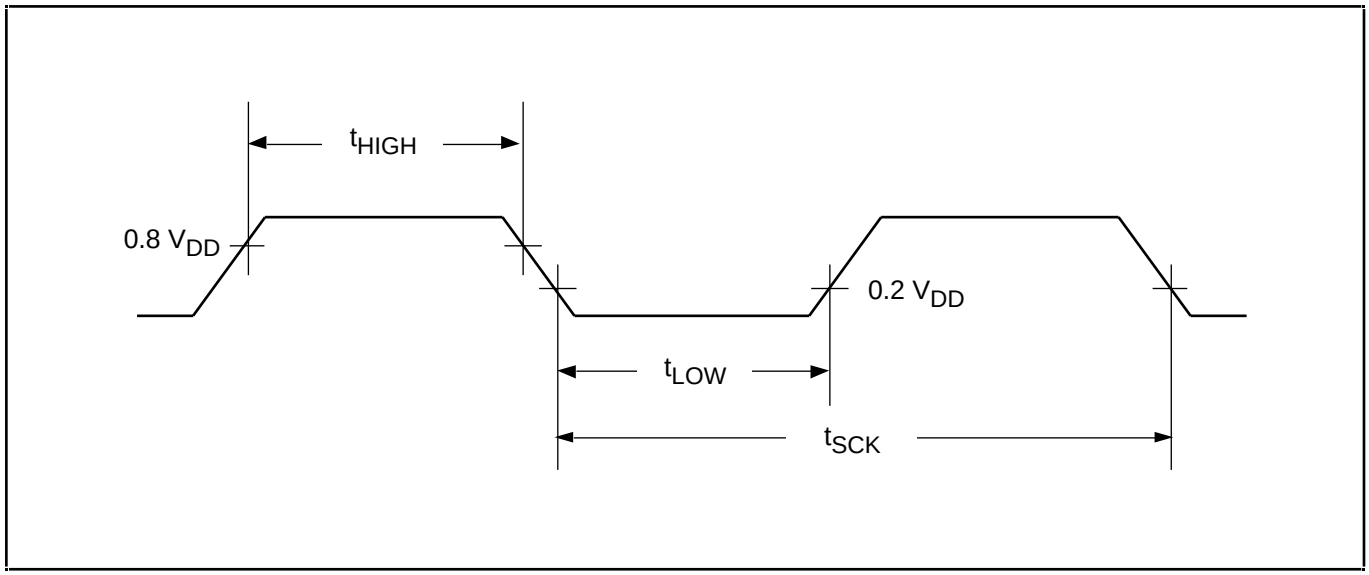
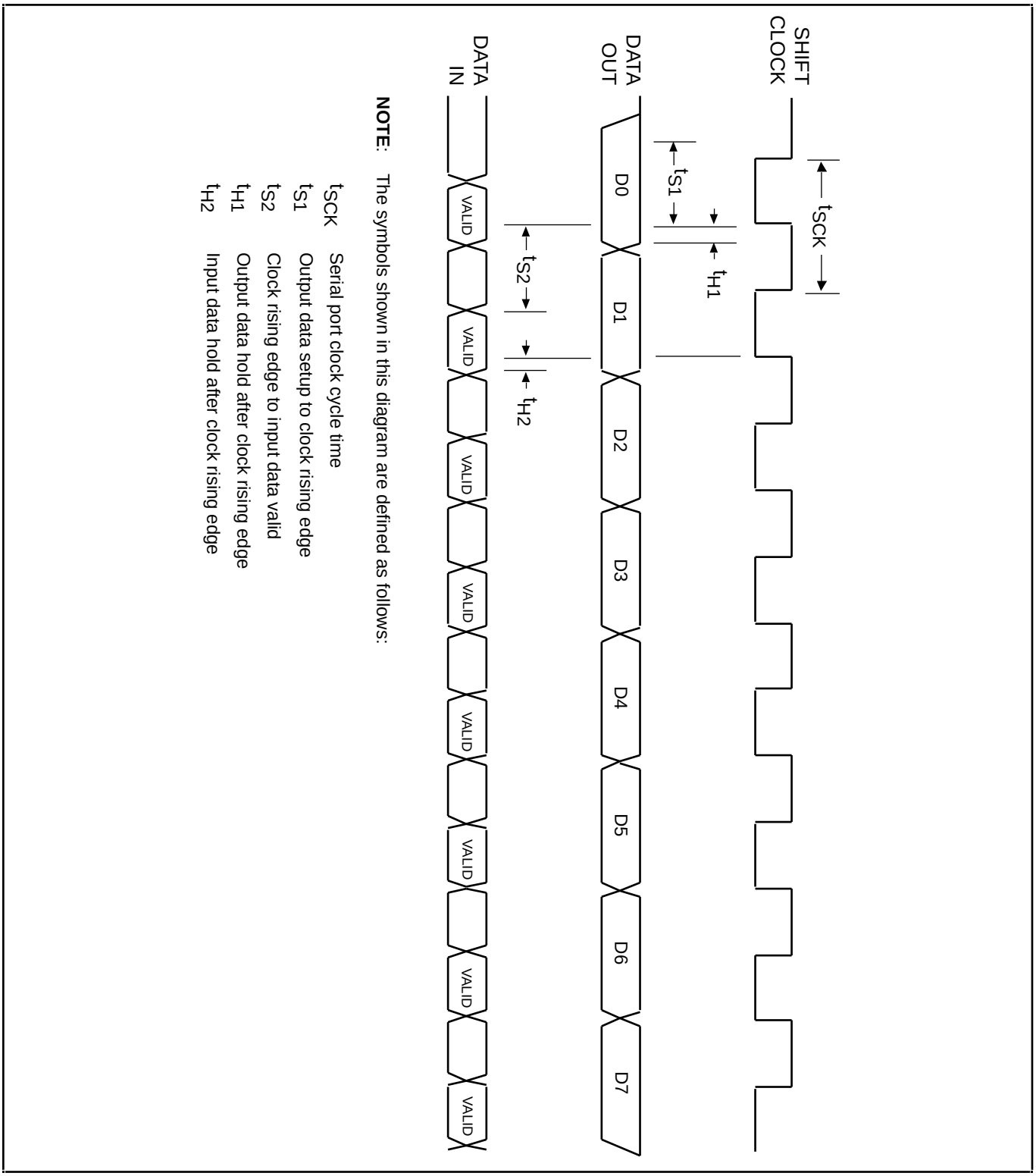


Figure 16–4. Serial Clock Waveform



NOTE: The symbols shown in this diagram are defined as follows:

t_{SCK}	Serial port clock cycle time
t_{S1}	Output data setup to clock rising edge
t_{S2}	Clock rising edge to input data valid
t_{H1}	Output data hold after clock rising edge
t_{H2}	Input data hold after clock rising edge

Figure 16–5. Serial Port Timing in Mode 0 (Shift Register Mode)

Table 16–8. External Memory Timing Characteristics (10 MHz)

(T_A = – 20°C to + 85°C, V_{DD} = 4.5 V to 6.0 V)

Number	Symbol	Parameter	Normal Timing		Extended Timing	
			Min	Max	Min	Max
1	t_{dA} (AS)	Address valid to AS ↑ delay	10	–	50	–
2	t_{dAS} (A)	AS ↑ to address float delay	35	–	85	–
3	t_{dAS} (DR)	AS ↑ to read data required valid	–	140	–	335
4	t_{wAS}	AS low width	35	–	85	–
5	t_{dA} (DS)	Address float to DS ↓	0	–	0	–
6a	t_{wDS} (read)	DS (read) low width	125	–	275	–
6b	t_{wDS} (write)	DS (write) low width	65	–	165	–
7	t_{dDS} (DR)	DS ↓ to read data required valid	–	80	–	255
8	t_{hDS} (DR)	Read data to DS ↑ hold time	0	–	0	–
9	t_{dDS} (A)	DS ↑ to address active delay	20	–	70	–
10	t_{dDS} (AS)	DS ↑ to AS ↓ delay	30	–	80	–
11	t_{dDO} (DS)	Write data valid to DS (write) ↓ delay	10	–	50	–
12	t_{dAS} (W)	AS ↑ to wait delay	–	90	–	335
13	t_{hDS} (W)	DS ↑ to wait hold time	0	–	0	–
14	t_{dRW} (AS)	R/W valid to AS ↑ delay	20	–	70	–
15	t_{dDS} (DW)	DS ↑ to write data not valid delay	20	–	70	–

NOTES:
1. All times are in ns and assume a 10 MHz input frequency.
2. Wait states add 100 ns to the time of numbers 3, 6a, 6b, and 7.
3. Auto-wait states add 100 ns to the time of number 12.

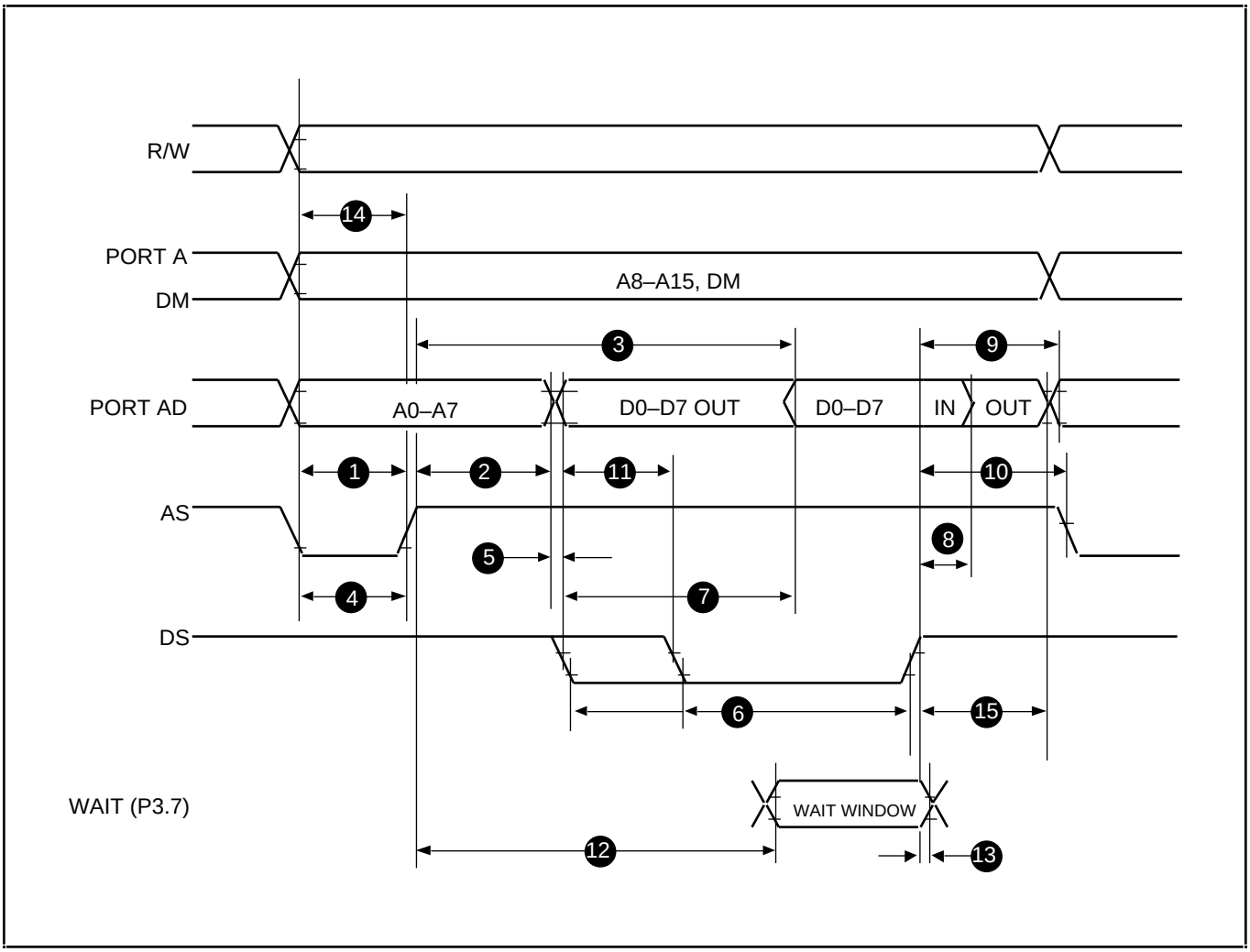
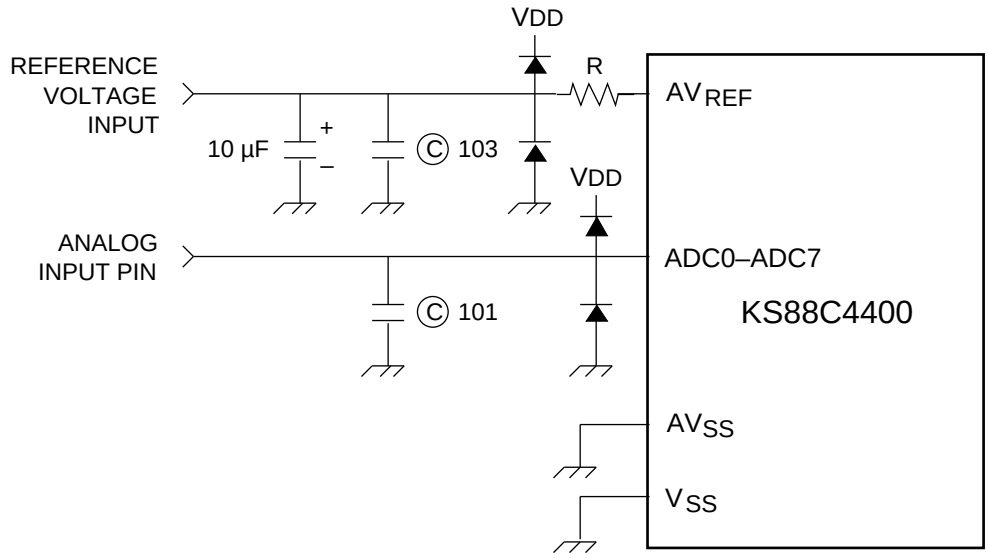


Figure 16–6. External Memory Read and Write Timing

(See Table 15–7 for a description of each timing point.)



NOTE: The symbol 'R' signifies an offset resistor with a value of 50 Ω to 100 Ω. If this resistor is omitted, the absolute accuracy will be maximum of 4 LSBs.

Figure 16–7. Recommended A/D Converter Circuit for Highest Absolute Accuracy

Table 16–9. Main Oscillator Frequency (f_{OSC1})

(T_A = – 20°C + 85°C, V_{DD} = 4.5 V to 6.0 V)

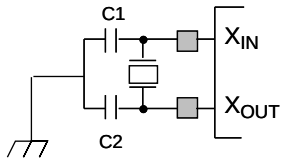
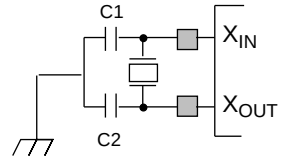
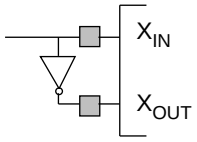
Oscillator	Clock Circuit	Test Condition	Min	Typ	Max	Unit
Crystal		CPU clock oscillation frequency	1	–	18	MHz
Ceramic		CPU clock oscillation frequency	1	–	18	MHz
External clock		X_{IN} input frequency	1	–	18	MHz

Table 16–10. Main Oscillator Clock Stabilization Time (t_{ST1})

(T_A = – 20°C + 85°C, V_{DD} = 4.5 V to 6.0 V)

Oscillator	Test Condition	Min	Typ	Max	Unit
Crystal	V_{DD} = 4.5 V to 6.0 V	–	–	20	ms
Ceramic	Stabilization occurs when V_{DD} is equal to the minimum oscillator voltage range.	–	–	10	ms
External clock	X_{IN} input high and low level width (t_{XH}, t_{XL})	25	–	500	ns

NOTE: Oscillation stabilization time (t_{ST1}) is the time required for the CPU clock to return to its normal oscillation frequency after a power-on occurs, or when Stop mode is ended by a RESET signal. The RESET should therefore be held at low level until the t_{ST1} time has elapsed (see Figure 15–3).

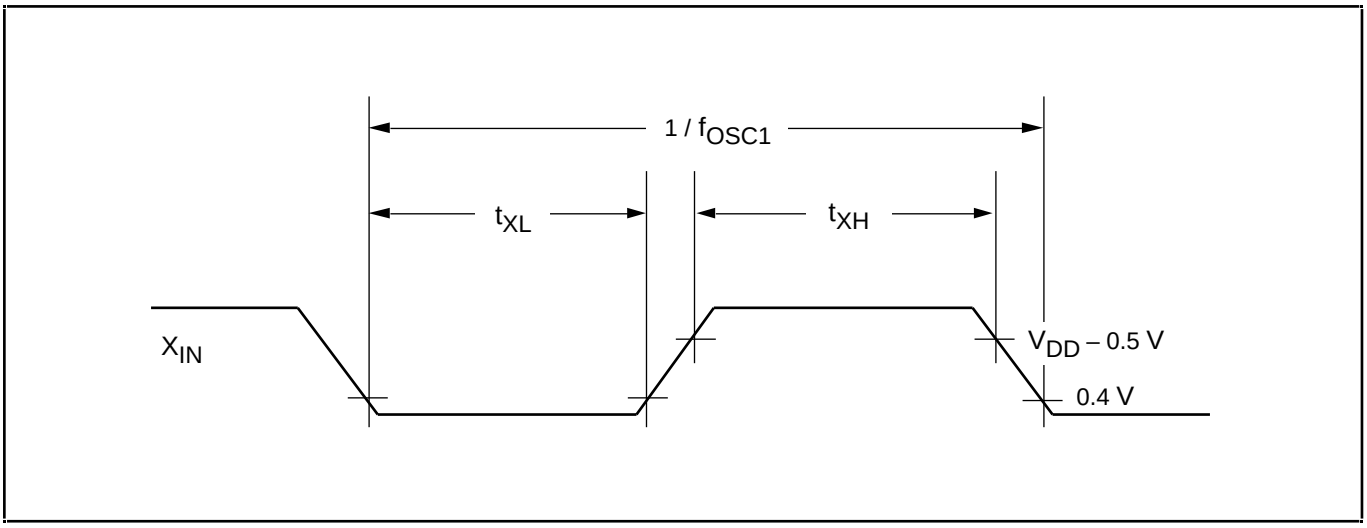


Figure 16–8. Clock Timing Measurement at X_{IN}

Characteristic Curves

NOTE

The characteristic values shown in the following graphs are based on actual test measurements. They do not, however, represent guaranteed operating values.

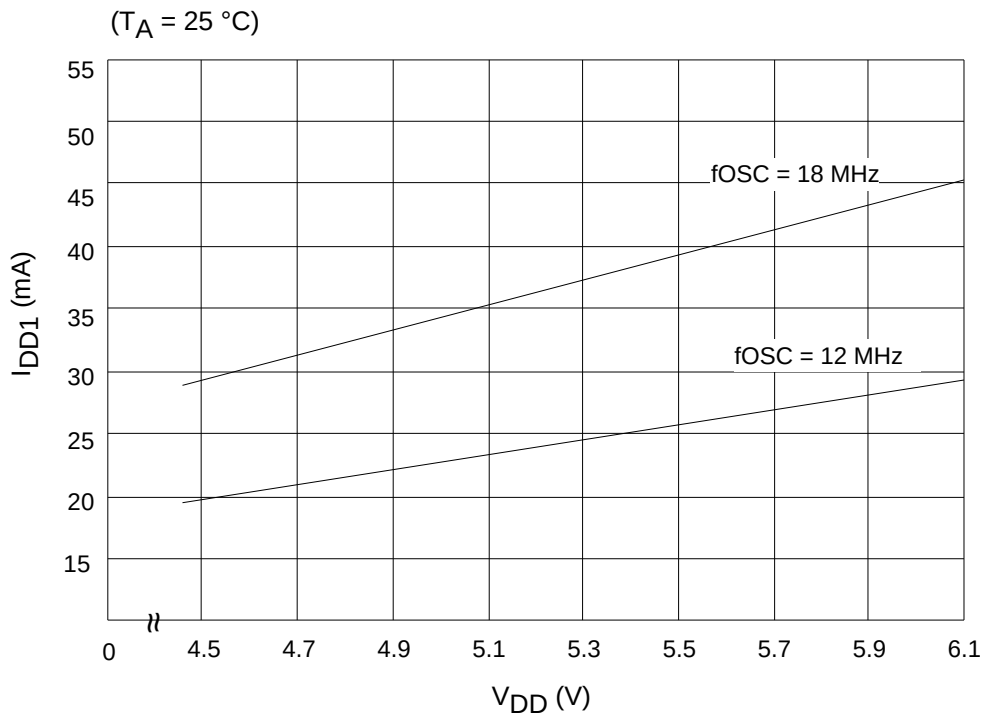


Figure 16–9. IDD1 vs VDD

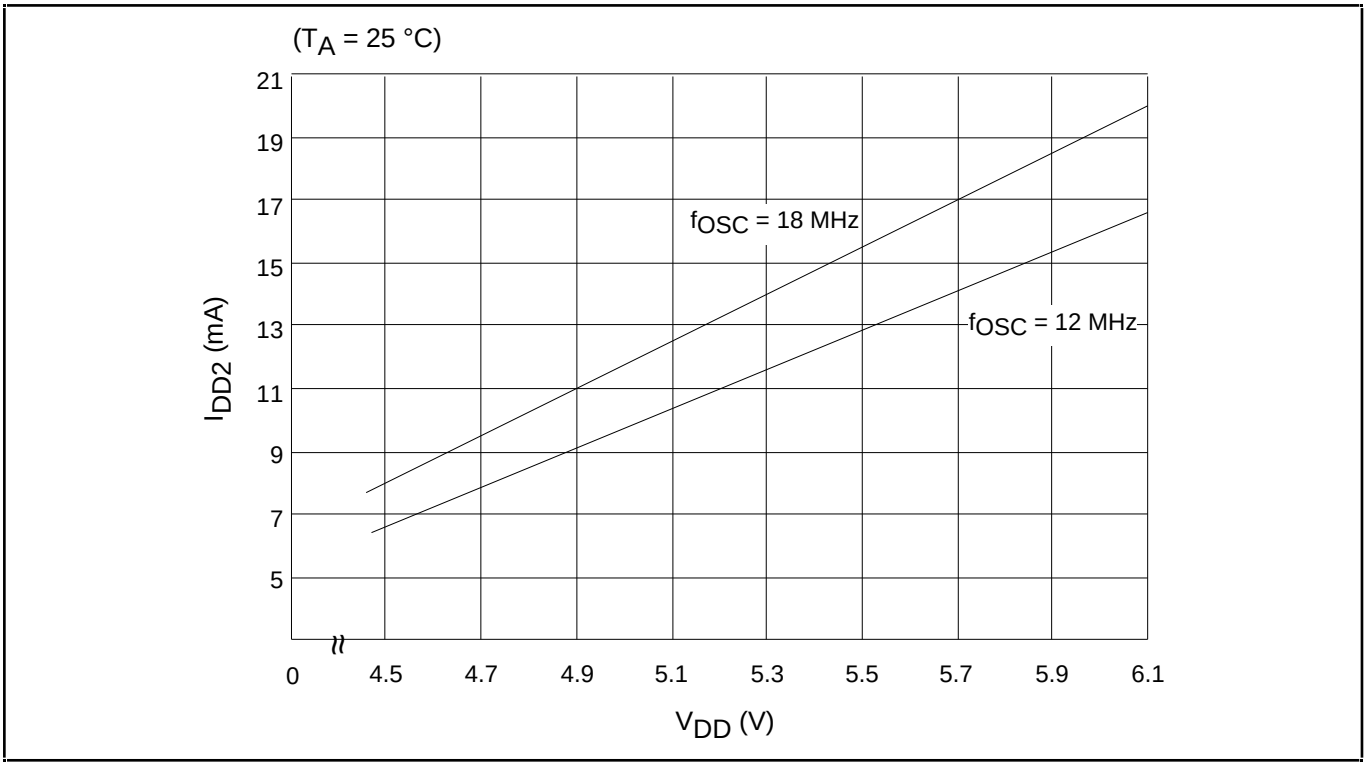


Figure 16–10. IDD2 vs VDD

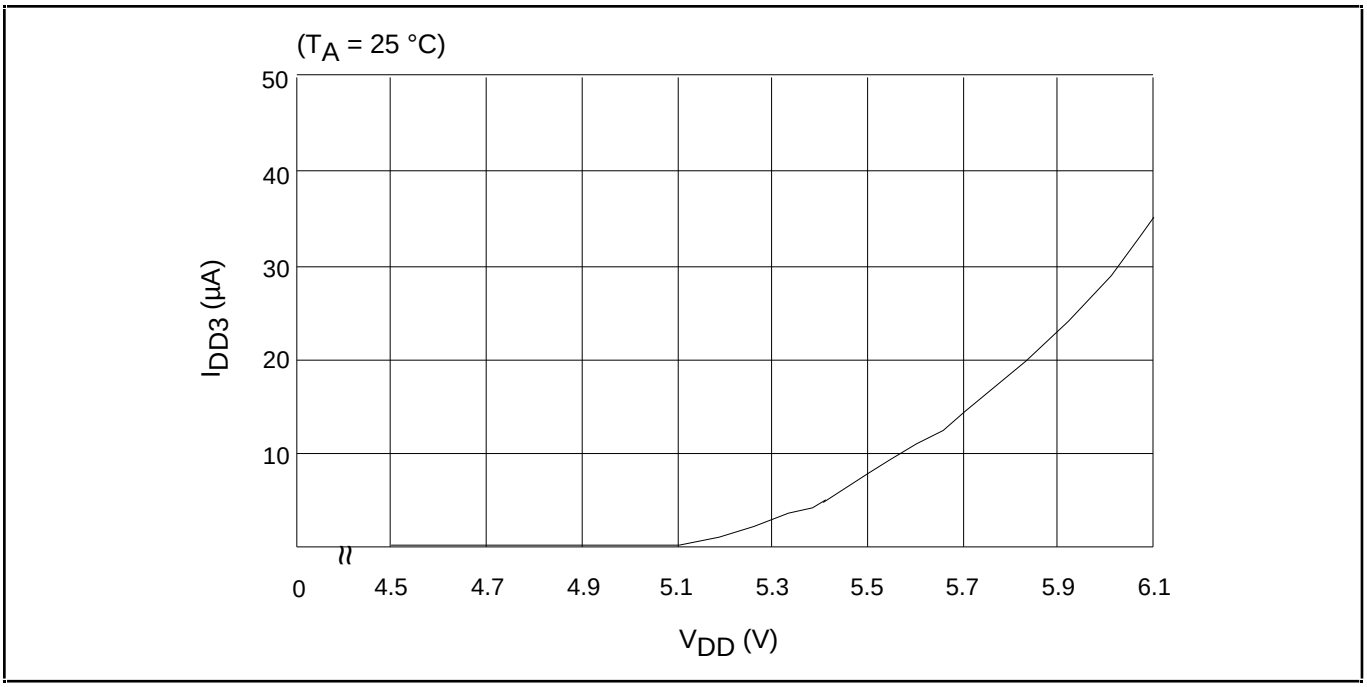


Figure 16–11. IDD3 vs VDD

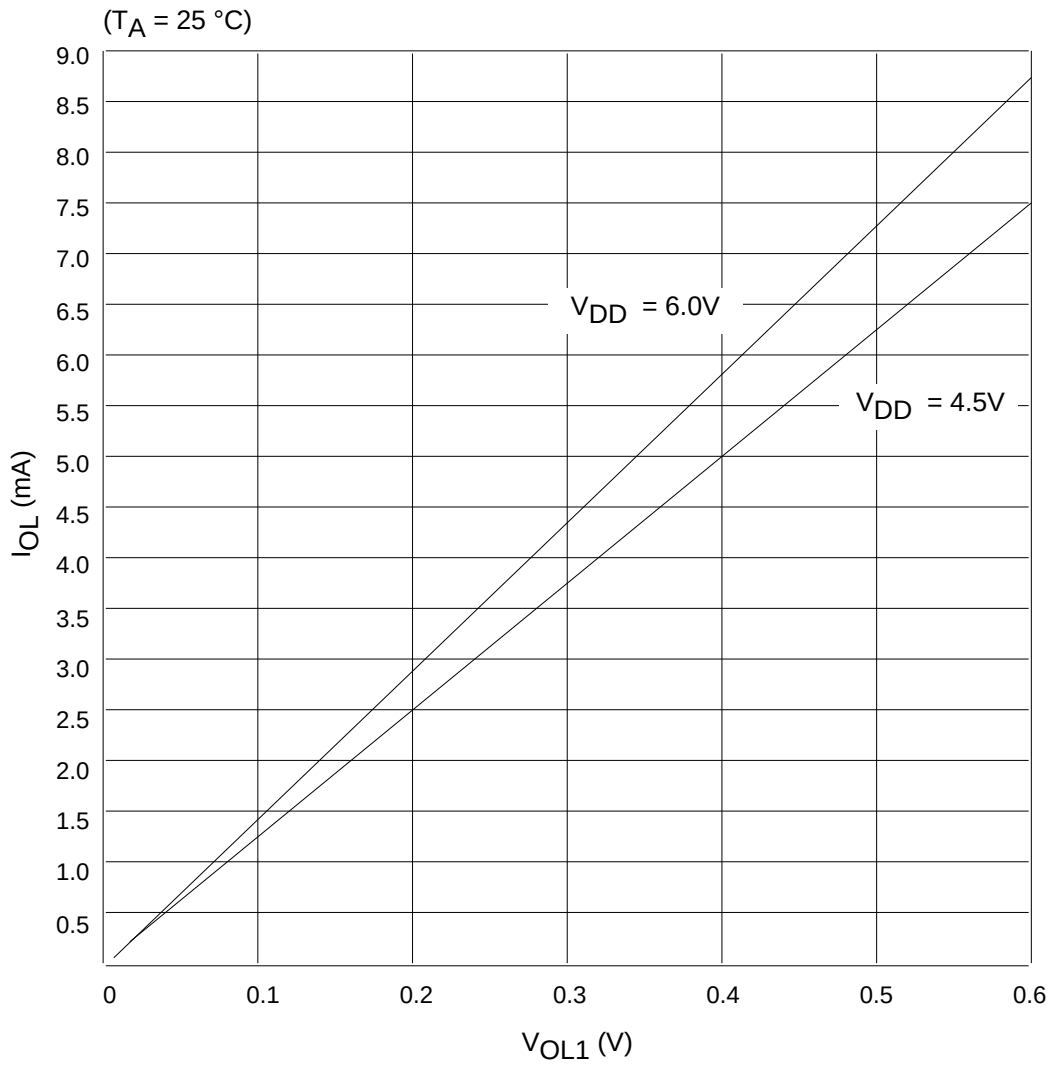


Figure 16–12. IOL vs VOL1

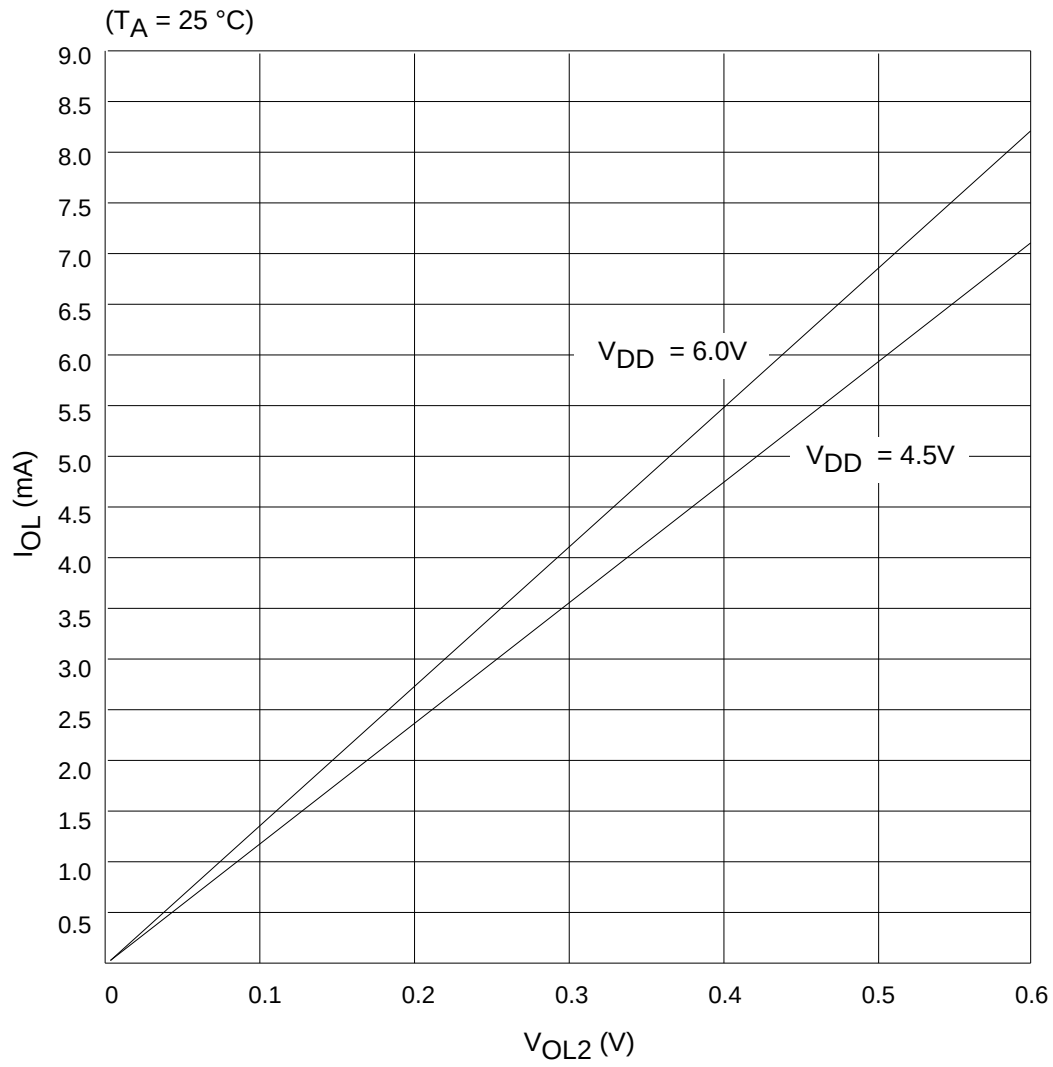


Figure 16–13. IOL vs VOL2

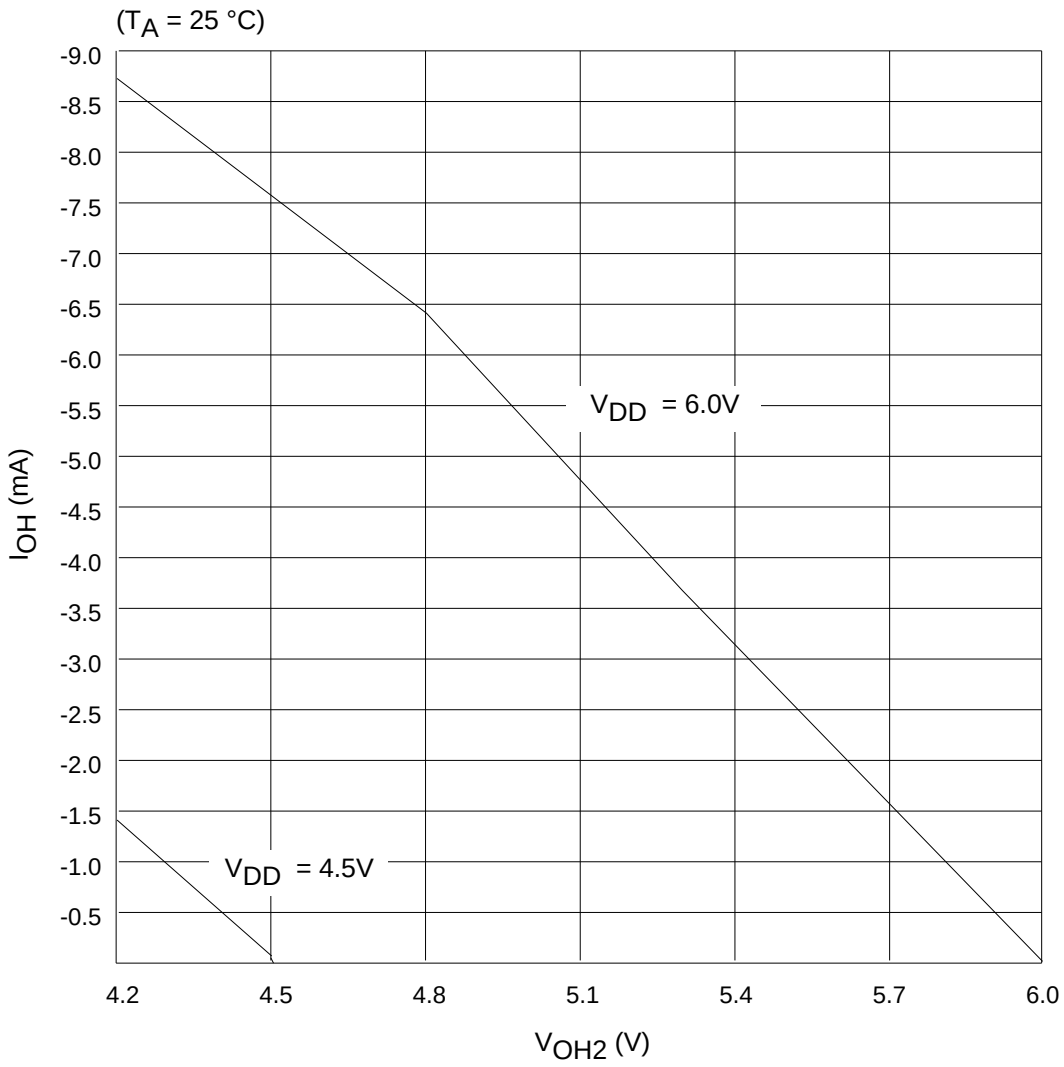
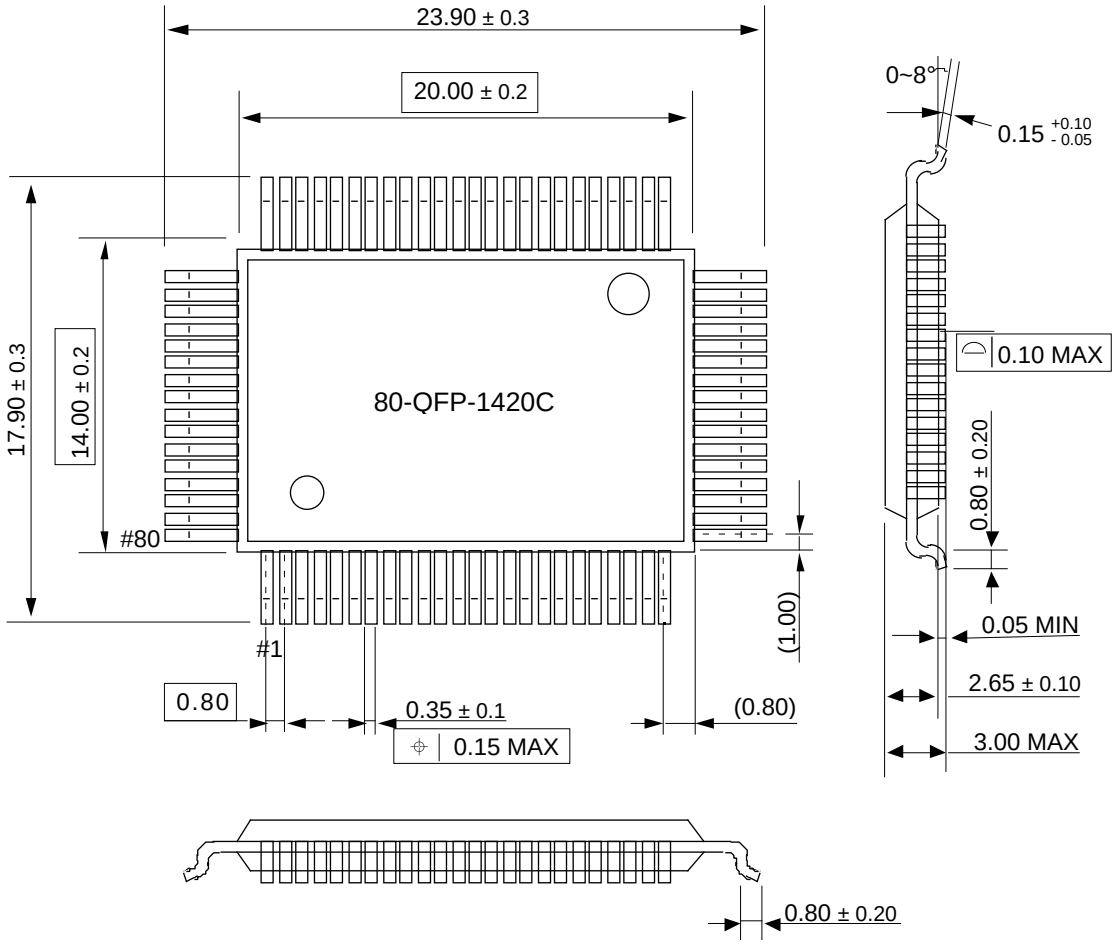


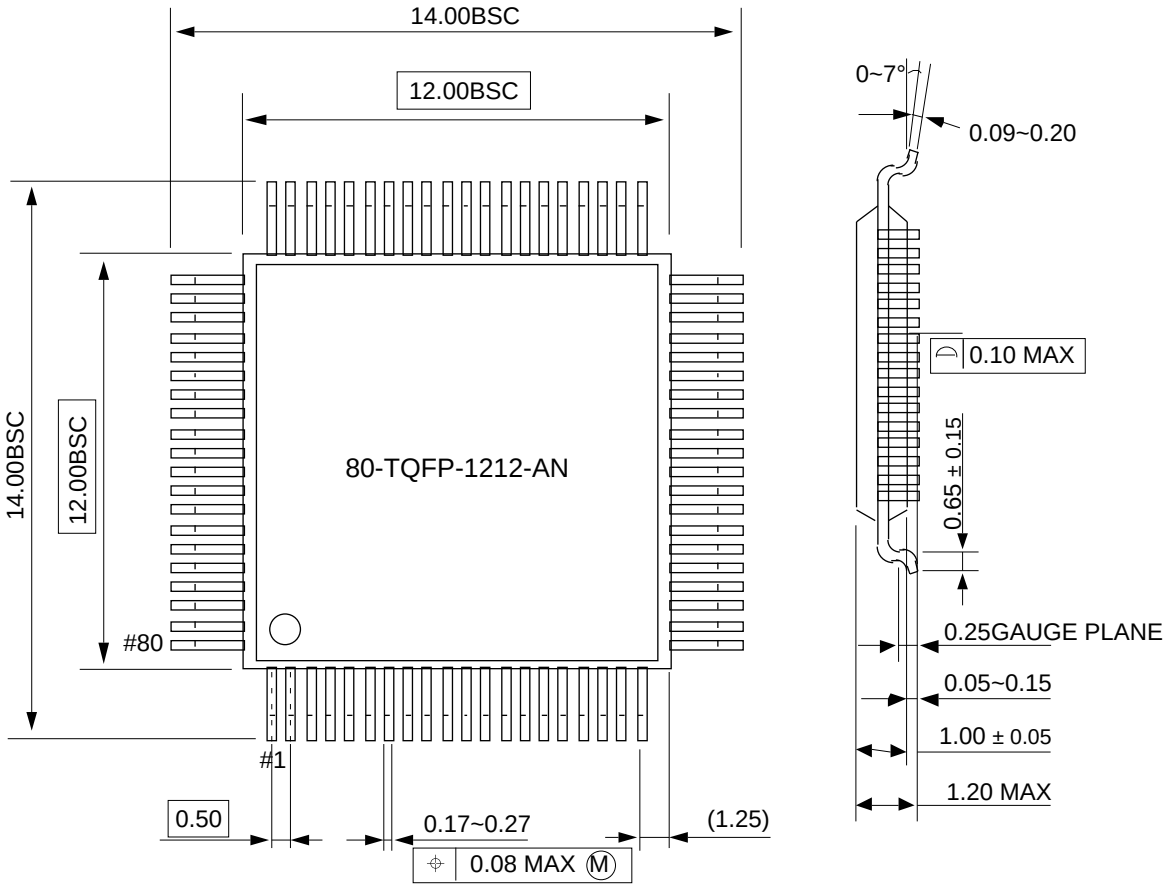
Figure 16–14. IOH vs VOH2

17 Mechanical Data



NOTE: Dimensions are in millimeters.

Figure 17–1. KS88C4400 QFP Standard Package Dimensions (in Millimeters)



NOTE: Dimensions are in millimeters.

Figure 17–2. KS88C4400 TQFP Standard Package Dimensions (in Millimeters)

18 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 as its operating system can be used. Two types of debugging tools including hardware and software are provided: the in-circuit emulator, SMDS2, developed for KS51, KS57, KS88 families of microcontrollers, and even more sophisticated and powerful in-circuit emulator, SMDS2+, for KS57, KS88 families of microcontrollers. The SMDS2+ is a new and improved version of SMDS2. In the future SMDS2+ will replace SMDS2 and eventually SMDS2 will not be supported. Samsung also offers support software that includes debugger, assembler, and a program for setting options.

DEVELOPMENT TOOLS VERSIONS

As of the date of this publication, two versions of the SMDS are being supported:

— SMDS2 Version 5.3 (S/W) and SMDS2 Version 1.3 (H/W); last release: October, 1995.

— SHINE Version 1.0 (S/W) and SMDS2+ Version 1.0 (H/W); last release: January, 1997.

SMDS V5.3

SMDS V5.3 is an assembly level debugger with user-friendly host interfacing that uses in-circuit emulator,SMDS2.

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 an relocatable assembler for Samsung's KS88-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 upto the maximum ROM size of the target device automatically.

TARGET BOARDS

Target boards are available for all KS88-series microcontrollers. All required target system cables and adapters are included with the device-specific target board.

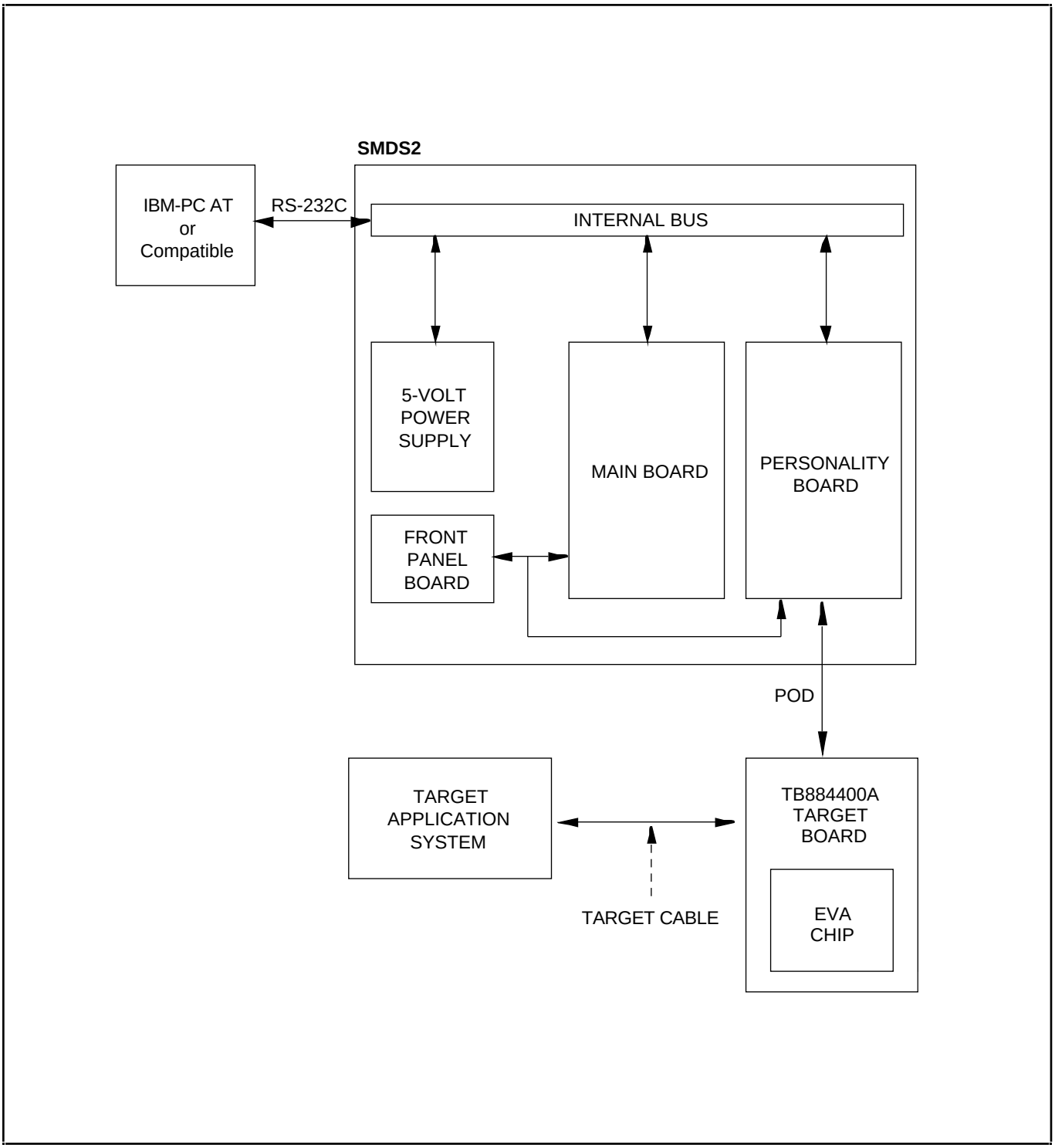


Figure 18–1. SMDS Product Configuration (SMDS2)

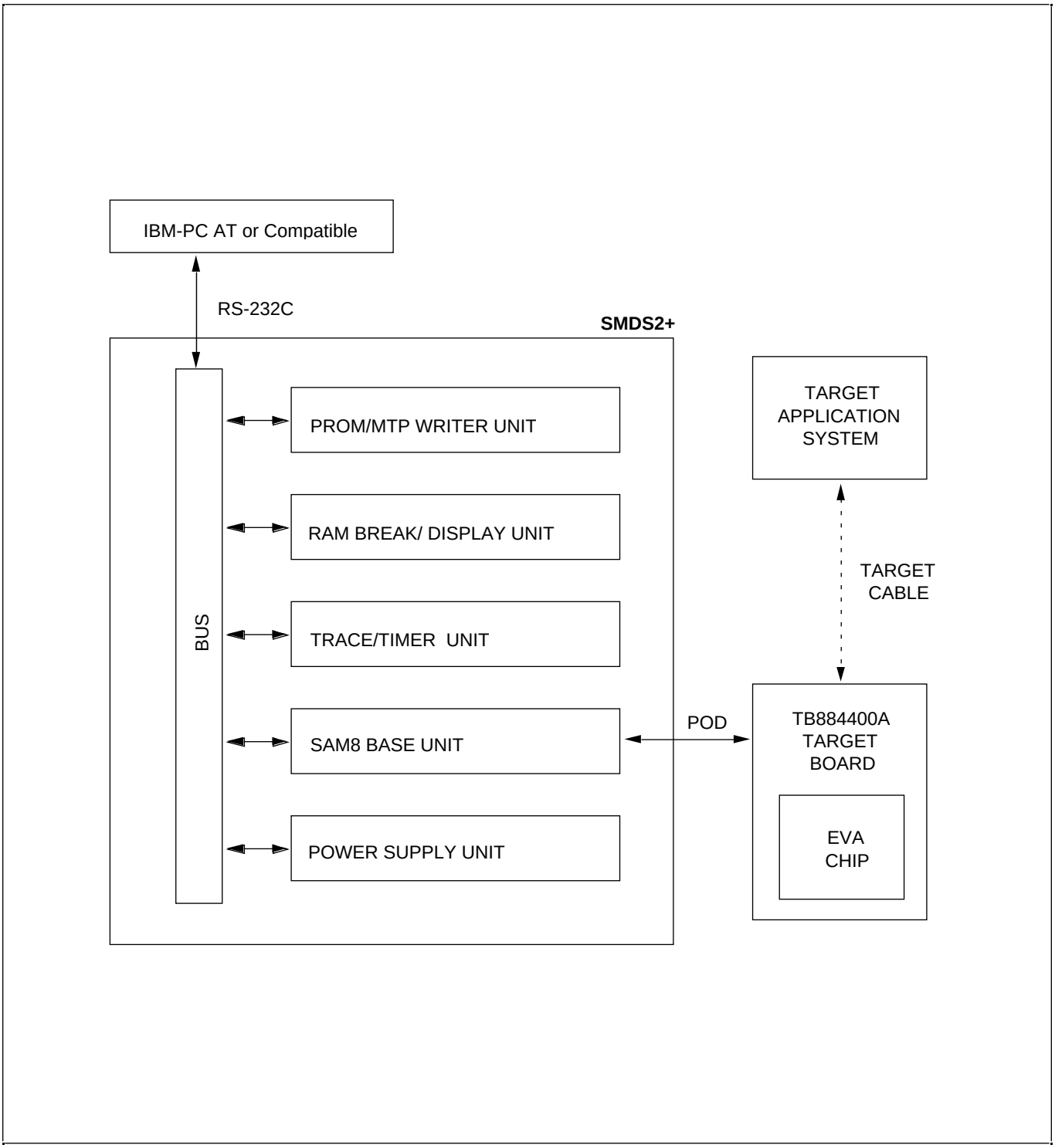


Figure 18–2. SMDS Product Configuration (SMDS2+)

TB884400A Target Board

The TB884400A target board is used for the KS88C4400 microcontroller. It is supported by the SMDS2 or SMDS2+ development system.

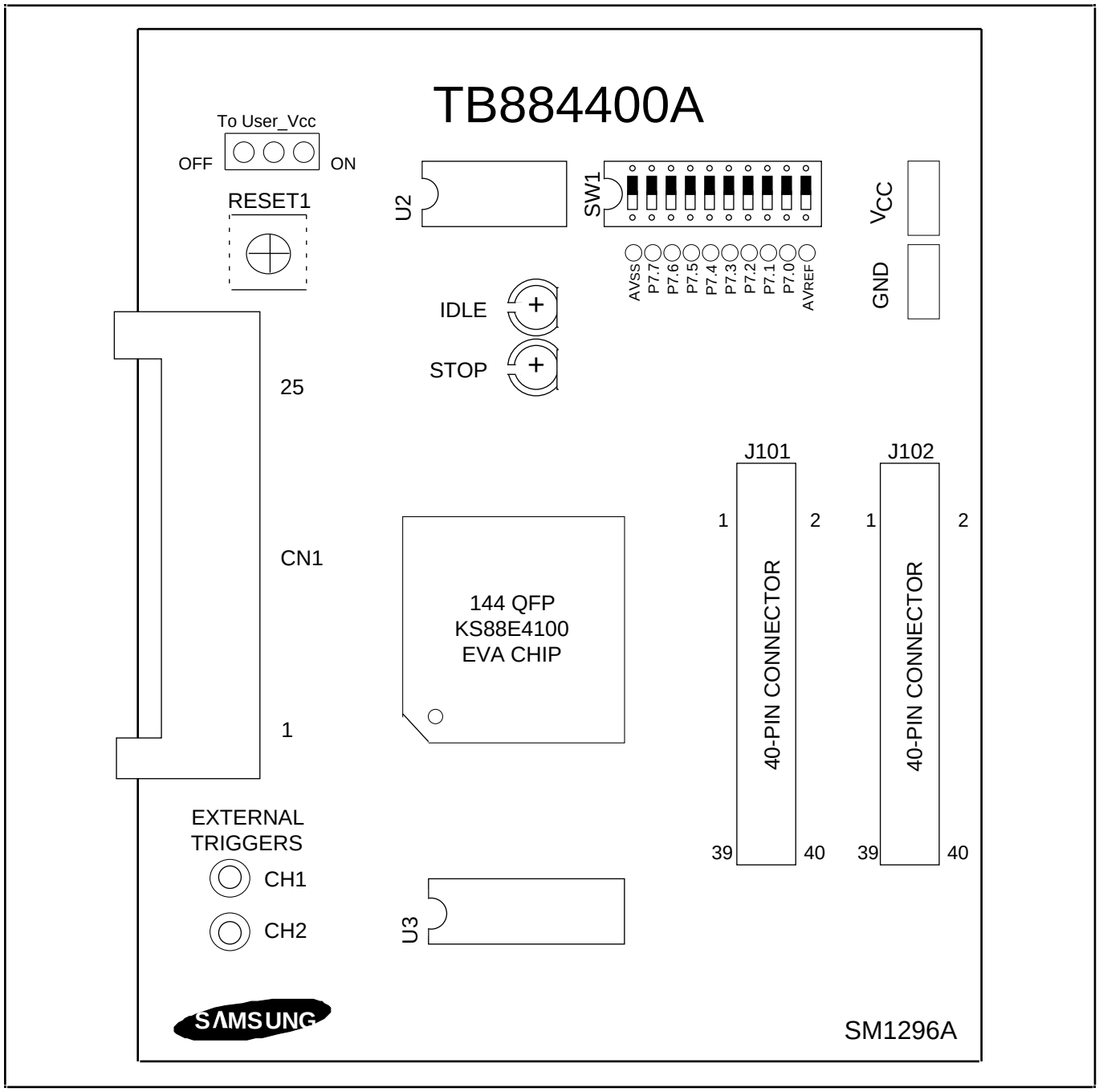
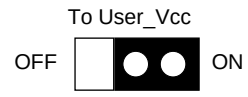
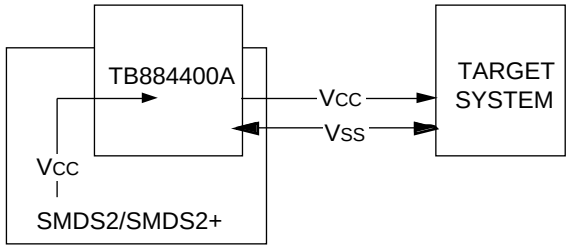
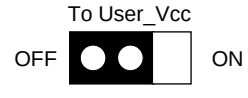
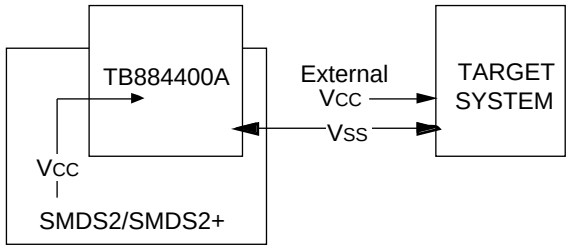


Figure 18–3. TB884400A Target Board Configuration

Table 18–1. Power Selection Settings for TB884400A

'To User_Vcc' Settings	Operating Mode	Comments
		The SMDS2/SMDS2+ main board supplies V_{CC} to the target board (evaluation chip) and the target system.
		The SMDS2/SMDS2+ main board 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 configuration:

Table 18–2. Using Single Header Pins as the Input Path for External Trigger Sources

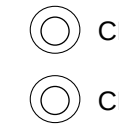
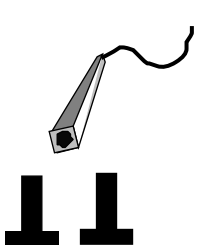
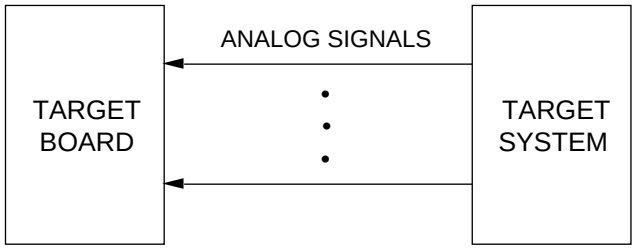
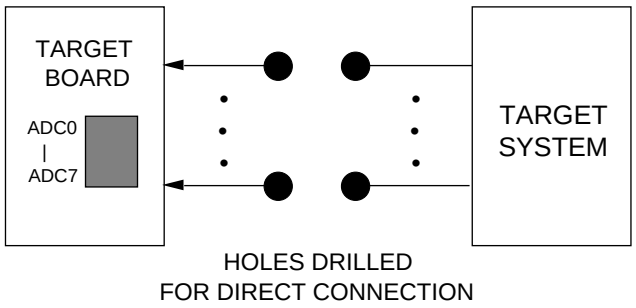
Target Board Part	Comments
	 You can connect an external trigger source to one of the two external trigger channels (CH1 or CH2) for the SMDS2/SMDS2+ breakpoint and trace functions.

Table 18–3. Analog Pin Connection Switch Settings (TB884400A)

Analog Pin Switch	Operating Mode
DIP SW1: ON 	
DIP SW1: OFF 	

NOTE: Analog signals coming into the target board can easily introduce noise into the analog converter circuit. This can cause invalid conversion results. To reduce noise, you can use the analog pin switches to provide the shortest possible path for analog signals. To do this, turn all DIP switches to the OFF position. Then, connect the analog signal lines directly via the holes of the corresponding analog pins.

IDLE LED

The Green LED is ON when the evaluation chip(KS88E4100) is in idle mode.

STOP LED

The Red LED is ON when the evaluation chip(KS88E4100) is in stop mode.

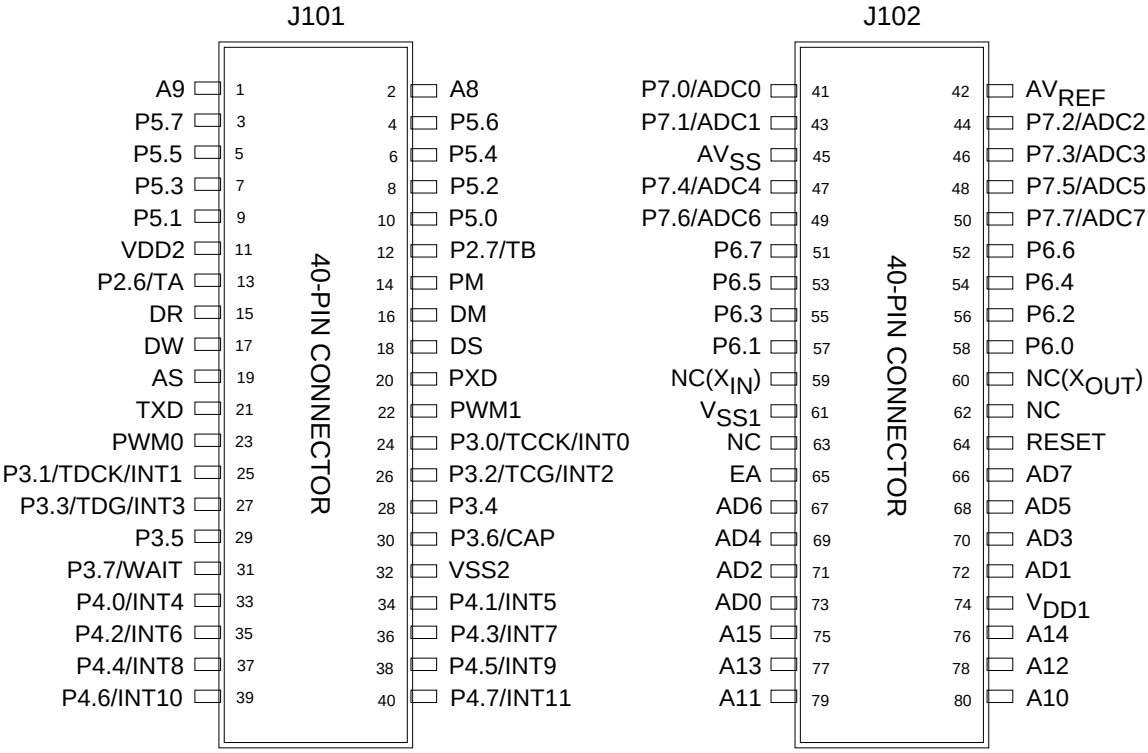
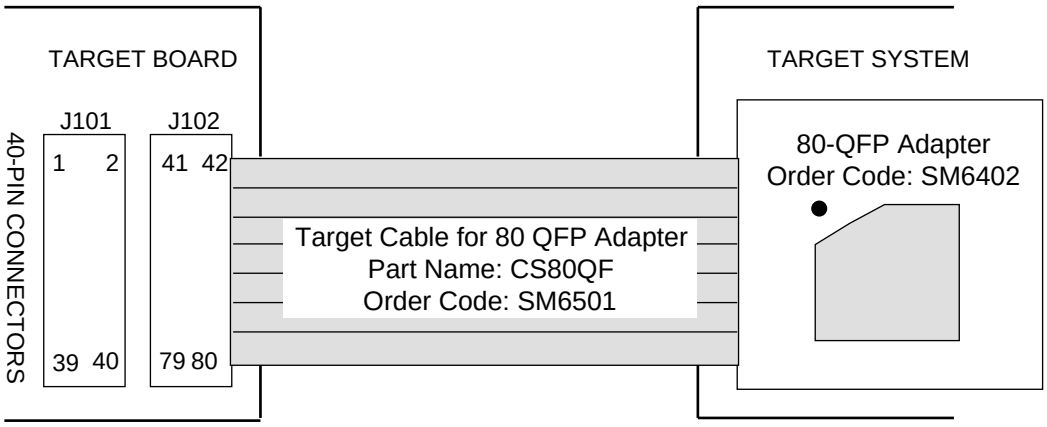


Figure 18–4. 40-Pin Connectors for TB884400A (KS88C4400, 80-QFP Package)



NOTE: Two 40-pin flat cables can be used instead of the target cable and the 80-QFP adapter to connect the target board and the target system.

Figure 18–5. TB884400A Cable for 80-QFP Adapter

note