

Product Overview

Address Spaces

Addressing Modes

Control Registers

Interrupt Structure

Instruction Set

1

PRODUCT OVERVIEW

SAM87RI PRODUCT FAMILY

Samsung's SAM87Ri 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 address/data bus architecture and a large number of bit-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.

KS86C4004/C4104 MICROCONTROLLER

The KS86C4004/C4104 single-chip 8-bit microcontroller is fabricated using an advanced CMOS process. It is built around the powerful SAM87Ri CPU core. The KS86C4004/C4104 is a versatile microcontroller, with its A/D converter and a zero-crossing detection capability it can be used in a wide range of general purpose applications.

Stop and Idle power-down modes were implemented to reduce power consumption. To increase on-chip register space, the size of the internal register file was logically expanded. The KS86C4004/C4104 has 4 K bytes of program memory on-chip (ROM) and 208 bytes of general purpose register area RAM.

Using the SAM87Ri design approach, the following peripherals were integrated with the SAM87Ri core:

- Four configurable I/O ports (KS86C4004: 22 pins, KS86C4104: 16 pins)
- Six interrupt sources with one vector and one interrupt level
- Two 8-bit timer/counter with various operating modes

The KS86C4004/C4104 microcontroller is ideal for use in a wide range of electronic applications requiring simple timer/counter, PWM, ADC, ZCD and capture functions. KS86C4004 is available in a 30-pin SDIP and a 32-pin SOP package. KS86C4104 is available in a 24-pin SDIP and a 24-pin SOP package.

FEATURES

CPU

- SAM87Ri CPU core

Memory

- 4-Kbyte internal program memory (ROM)
- 208-byte general purpose register area (RAM)

Instruction Set

- 41 instructions
- IDLE and STOP instructions added for power-down modes.

Instruction Execution Time

- 600 ns at 10 MHz f_{OSC} (minimum)

Interrupts

- 6 interrupt sources with one vector and one level interrupt structure

Oscillation Frequency

- 1-MHz to 10-MHz external crystal oscillator
- Maximum 10-MHz CPU clock
- 4 MHz RC oscillator

General I/O

- Four I/O ports (22 pins for KS86C4004, 16 pins for KS86C4104)
- Bit programmable ports

A/D Converter

- Eight analog input pins
- 8-bit conversion resolution

Timer/Counter

- One 8-bit basic timer for watchdog function
- One 8-bit timer/counter with three operating modes (10-bit PWM 1ch)
- One 8-bit timer/counter for the zero-crossing detection circuit

Zero-Crossing Detection Circuit

- Zero-crossing detection circuit that generates a digital signal in synchronism with an AC signal input

Buzzer Frequency Range

- 200 Hz to 20 kHz signal can be generated

Operating Temperature Range

- -40°C to $+85^{\circ}\text{C}$

Operating Voltage Range

- 2.7 V to 5.5 V

OTP Interface Protocol Spec

- Serial OTP

Package Types

- 30-pin SDIP, 32-pin SOP for KS86C4004/P4004
- 24-pin SDIP, 24-pin SOP for KS86C4104/P4104

BLOCK DIAGRAM

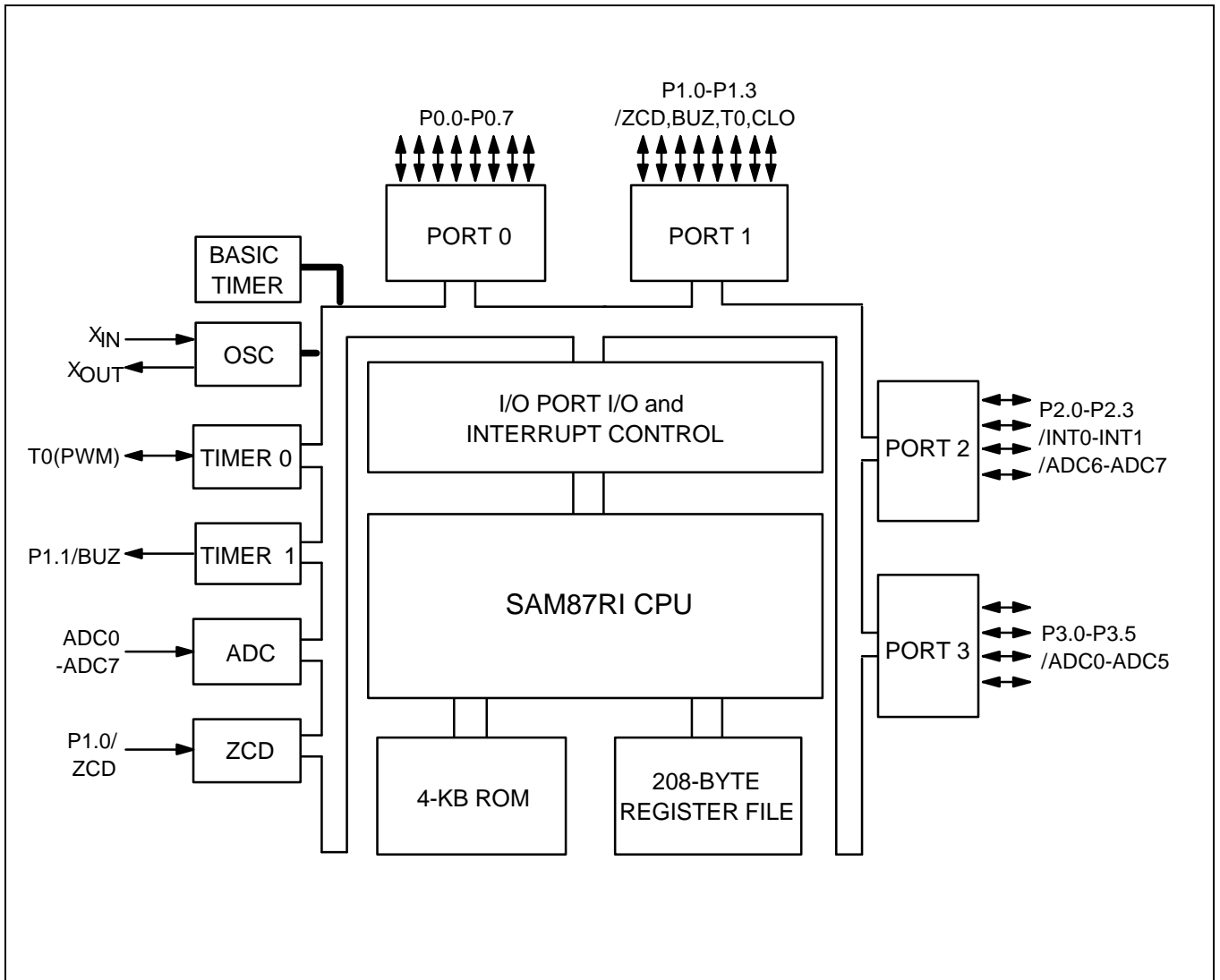


Figure 1-1. Block Diagram

PIN ASSIGNMENTS

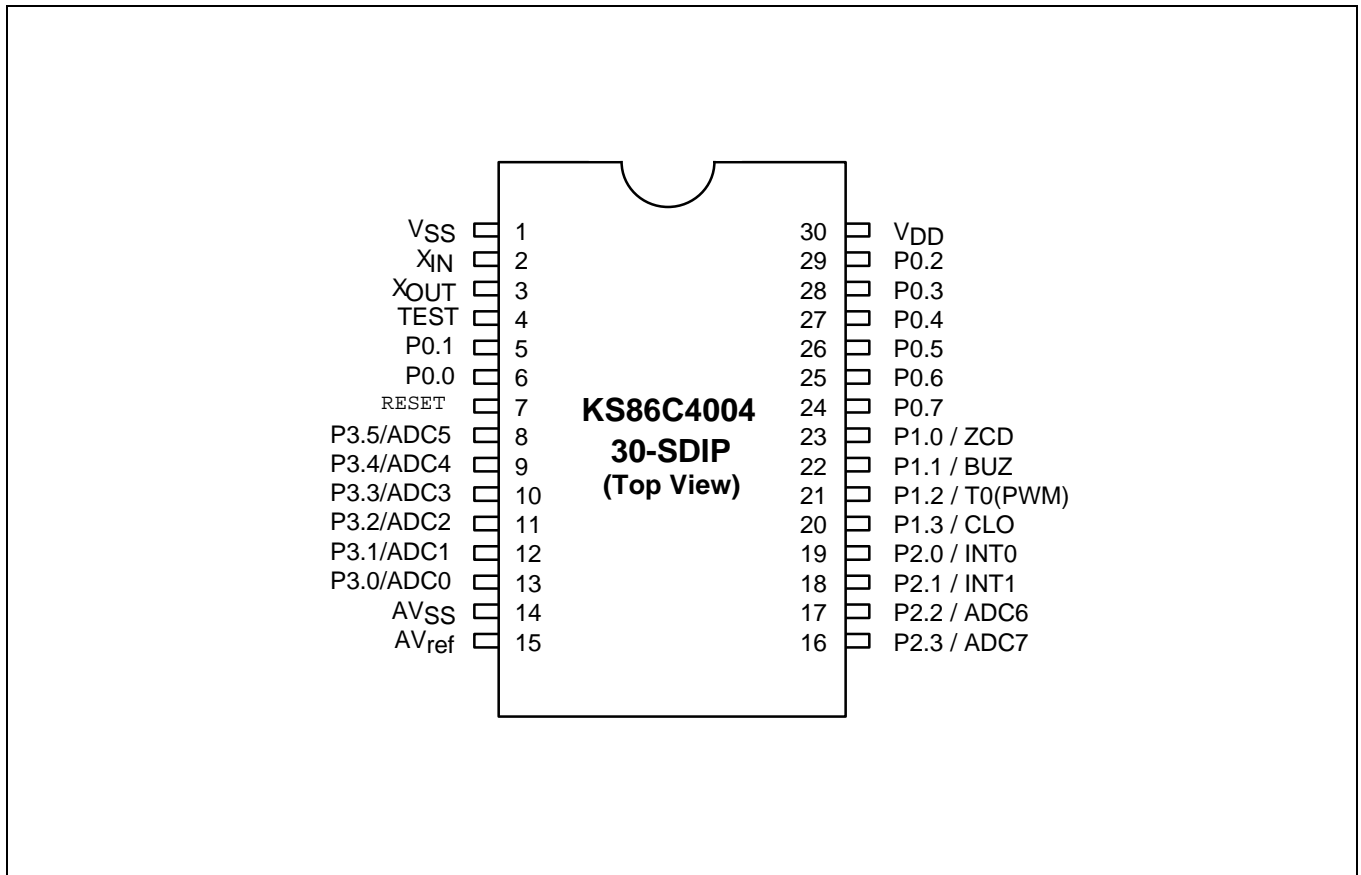


Figure 1-2. Pin Assignment Diagram (30-Pin SDIP Package)

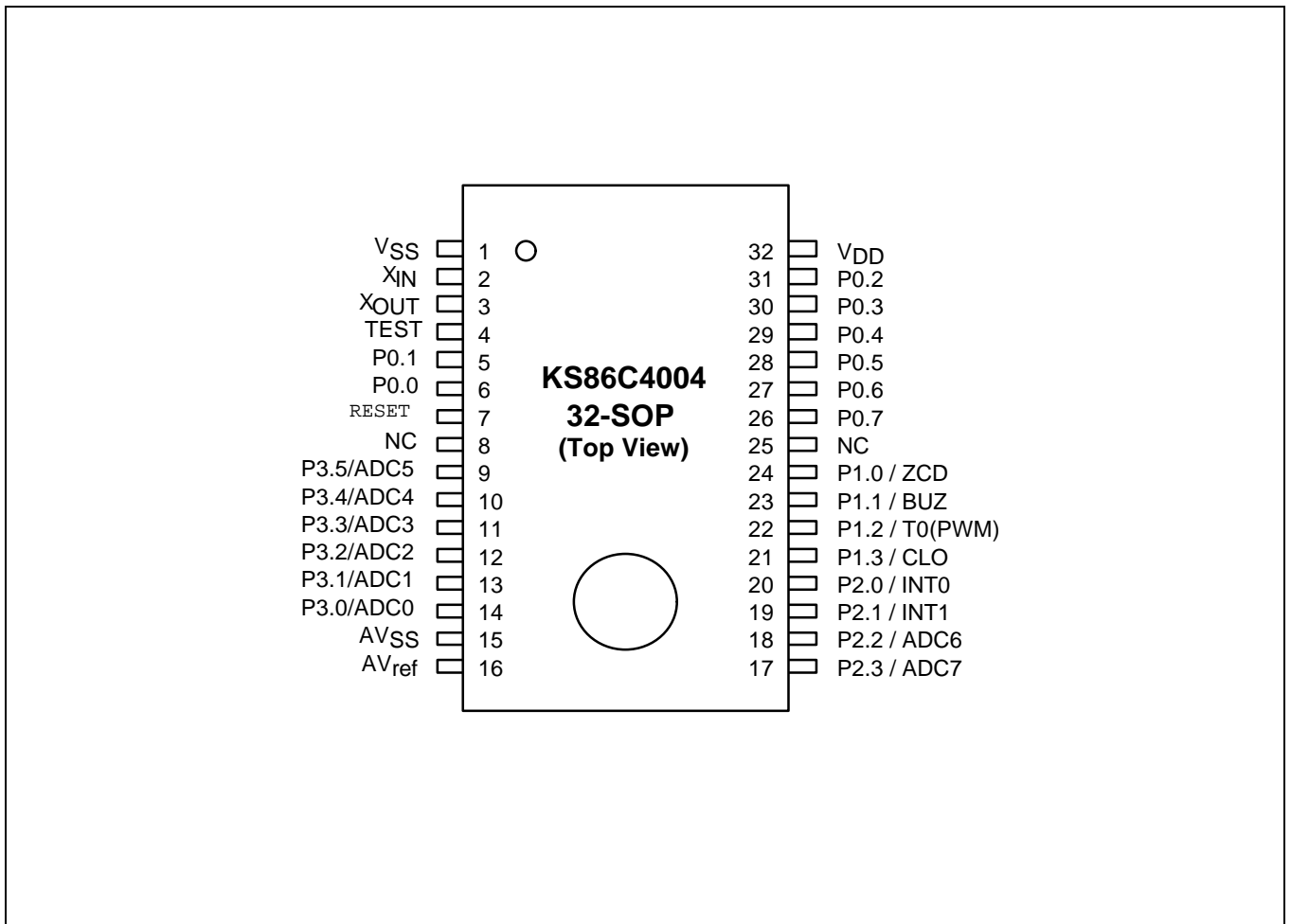


Figure 1-3. Pin Assignment Diagram (32-Pin SOP Package)

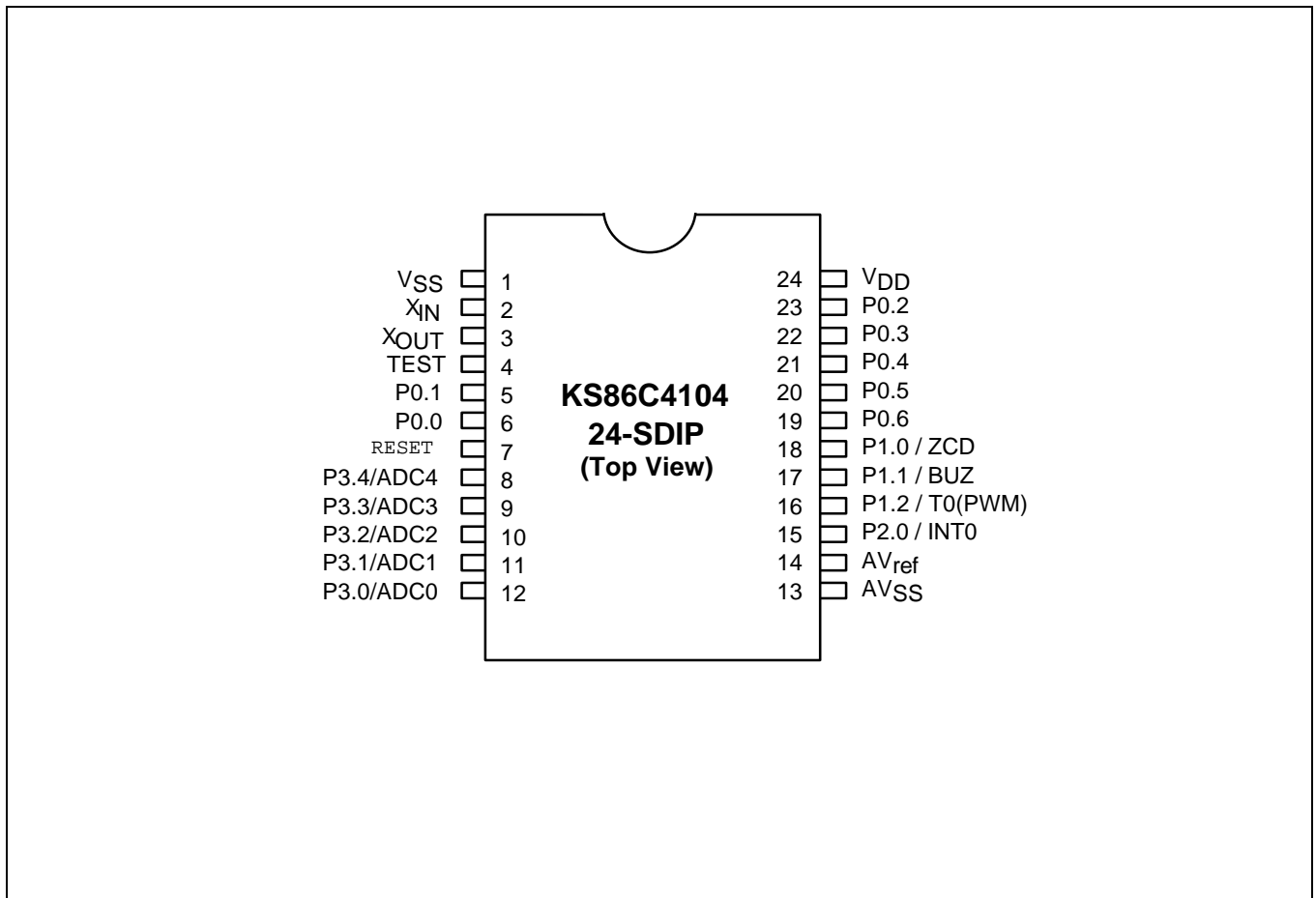


Figure 1-4. Pin Assignment Diagram (24-Pin SDIP Package)

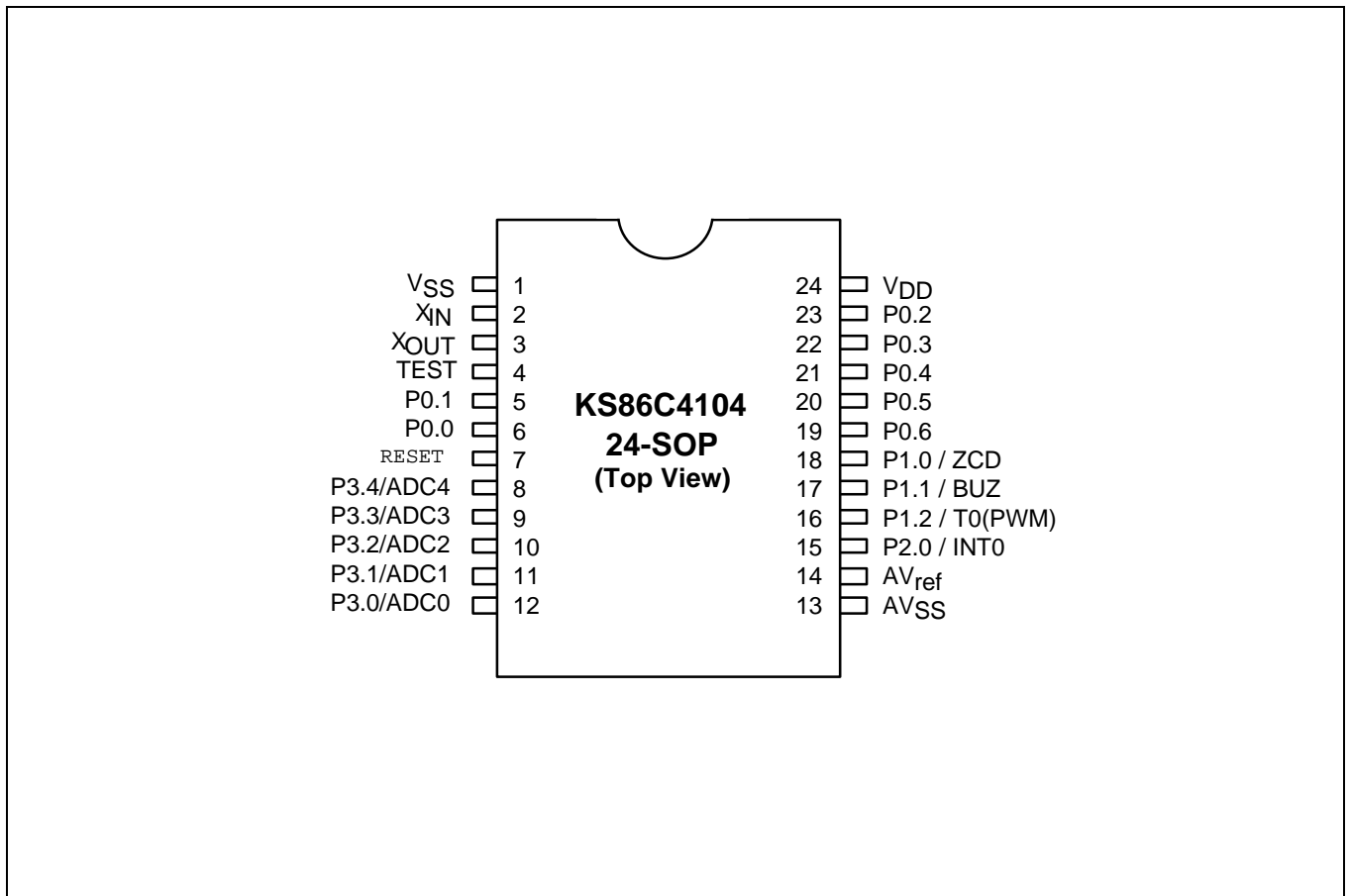


Figure 1-5. Pin Assignment Diagram (24-Pin SOP Package)

PIN DESCRIPTIONS

Table 1–1. KS86C4004/C4104 Pin Descriptions

Pin Names	Pin Type	Pin Description	Circuit Type	Share Pins
P0.0-P0.7	I/O	Bit-programmable I/O port for normal input or push-pull, open-drain output. Pull-up resistors are assignable by software.	E	
P1.0-P1.3	I/O	Bit-programmable I/O port for Schmitt trigger input or push-pull output. Pull-up resistors are assignable by software. Port1 pins can also be used as alternative functions.	F D D D	ZCD BUZ T0(PWM) CLO
P2.0-P2.3	I/O	Bit-programmable I/O port for Schmitt trigger input or push-pull, open drain output. Pull up resistors are assignable by software. Port2 can also be used as external interrupt, A/D input.	E E-1	INT0–INT1 ADC6–ADC7
P3.0-P3.5	I/O	Bit-programmable I/O port for Schmitt trigger input or push-pull output. Pull-up resistors are assignable by software. Port 3 pins can also be used as A/D converter input.	F	ADC0–ADC5
X _{IN} , X _{OUT}	–	Crystal/ceramic, or RC oscillator signal for system clock.	–	–
INT0–INT1	I	External interrupt input.	E	P2.0–P2.1
RESET	I	System RESET signal input pin.	B	–
TEST	I	Test signal input pin (for factory use only: must be connected to V _{SS})	–	–
V _{DD} , V _{SS}	–	Voltage input pin and ground	–	–
A _V _{REF} , A _V _{SS}	–	A/D converter reference voltage input and ground	–	–
ZCD	I	Zero crossing detector input	F	P1.0
BUZ	O	200 Hz–20 kHz frequency output for buzzer sound	D	P1.1
T0	I/O	Timer 0 capture input or 10-bit PWM output	D	P1.2
CLO	O	System clock output port	D	P1.3
ADC0–ADC7	I	A/D converter input	F E-1	P3.0–P3.5 P2.2–P2.3

NOTE: Port 0.7, P1.3, P2.1–P2.3 and P3.5 is not available in KS86C4104/P4104 .

PIN CIRCUITS

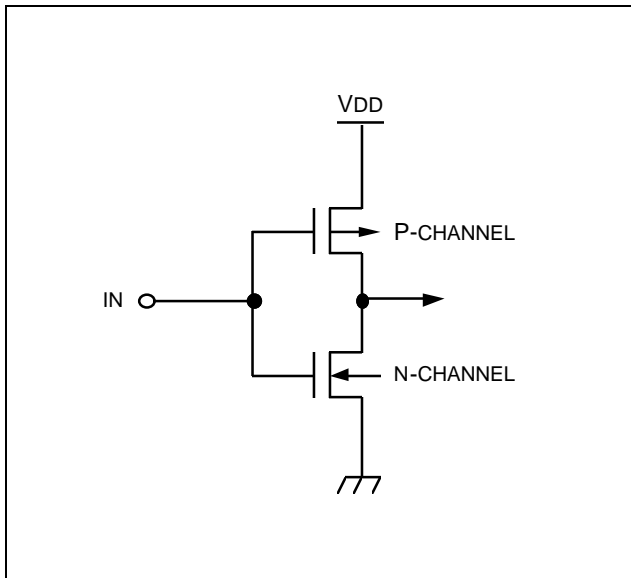


Figure 1-6. Pin Circuit Type A

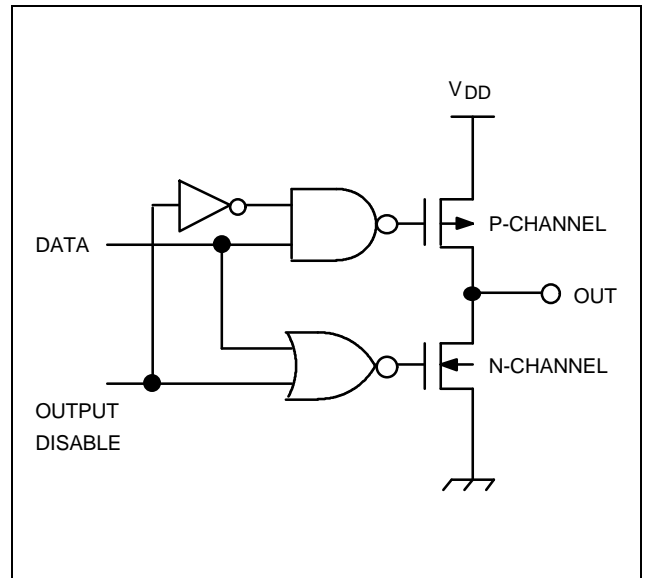


Figure 1-8. Pin Circuit Type C

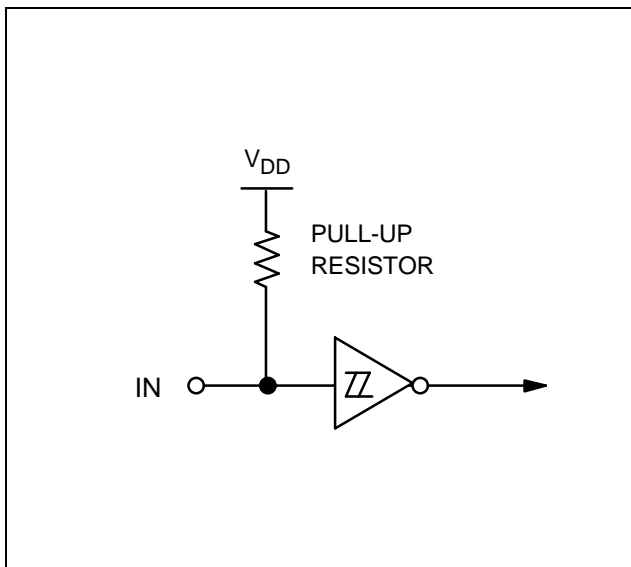


Figure 1-7. Pin Circuit Type B

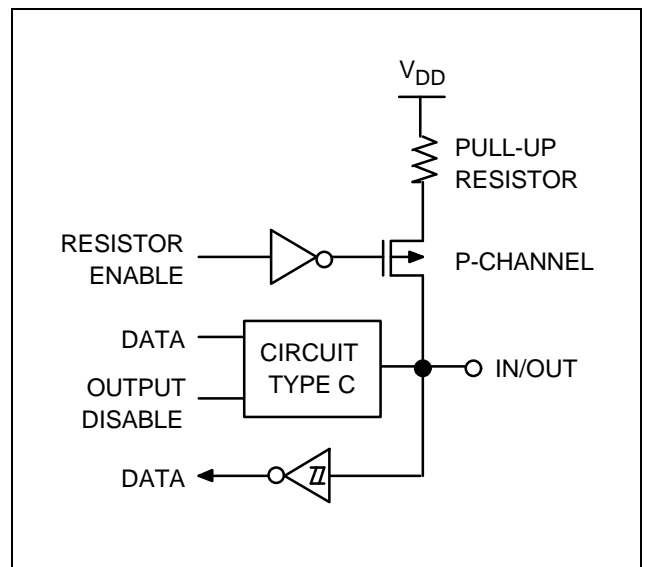


Figure 1-9. Pin Circuit Type D

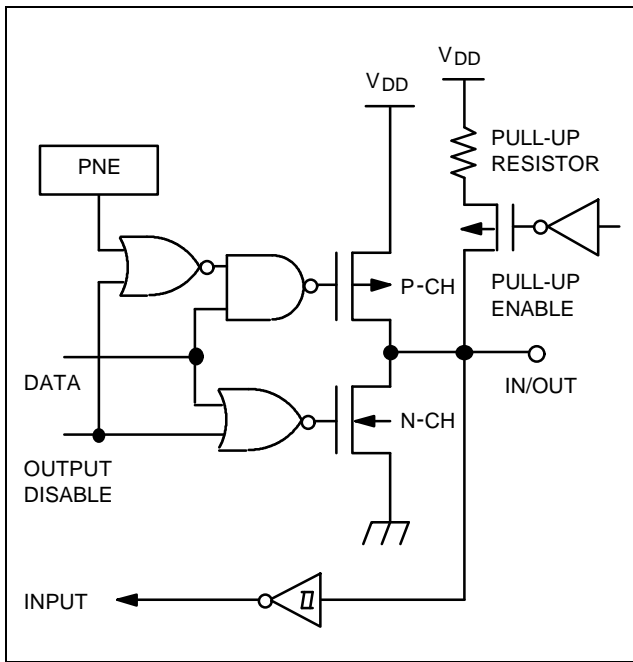


Figure 1-10. Pin Circuit Type E

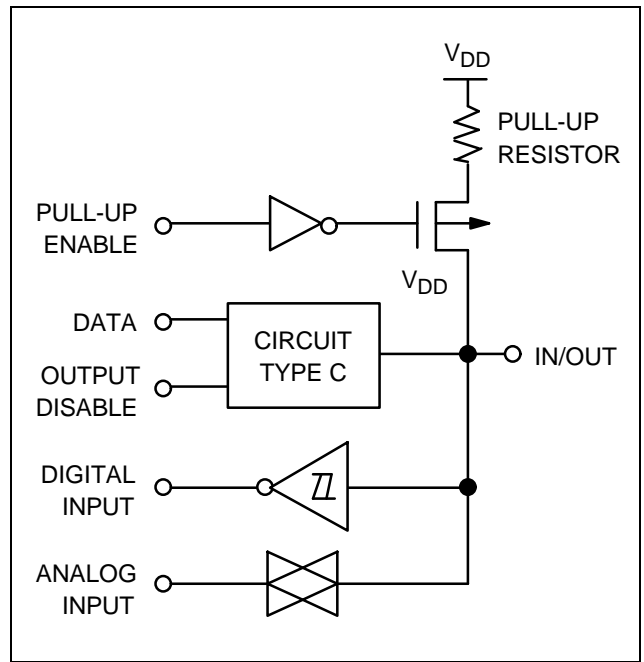


Figure 1-12. Pin Circuit Type F

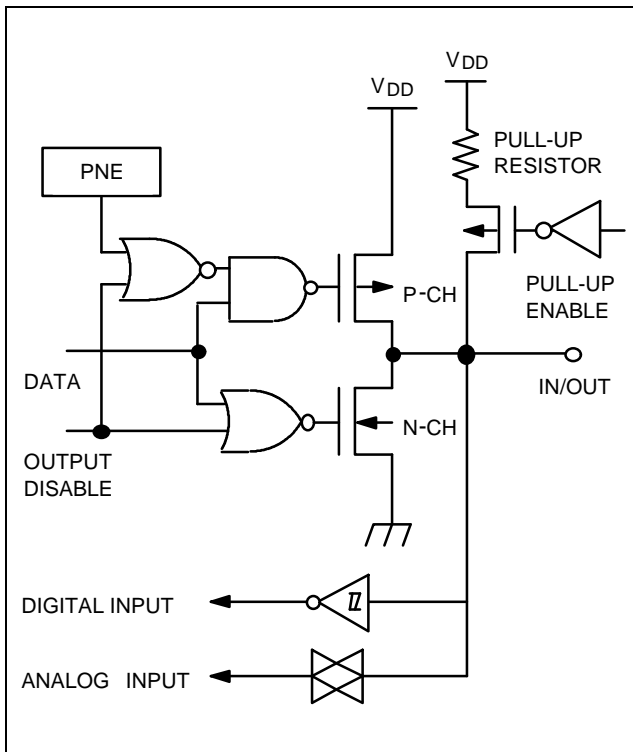


Figure 1-11. Pin Circuit Type E-1

2 ADDRESS SPACES

OVERVIEW

The KS86C4004/C4104 microcontroller has two kinds of address space:

- Internal program memory (ROM)
- Internal register file

A 12-bit address bus supports program memory operations. A separate 8-bit register bus carries addresses and data between the CPU and the internal register file.

The KS86C4004/C4104 has 4-Kbytes of mask-programmable on-chip program memory: which is configured as the Internal ROM mode, all of the 4-Kbyte internal program memory is used.

The KS86C4004/C4104 microcontroller has 208 general-purpose registers in its internal register file. Thirty-two bytes in the register file are mapped for system and peripheral control functions.

PROGRAM MEMORY (ROM)

Normal Operating Mode

The KS86C4004/C4104 has 4 K bytes (locations 0H–0FFFH) of internal mask-programmable program memory.

The first 2 bytes of the ROM (0000H–0001H) are interrupt vector address.

Unused locations (0002H–00FFH) can be used as normal program memory.

The program reset address in the ROM is 0100H.

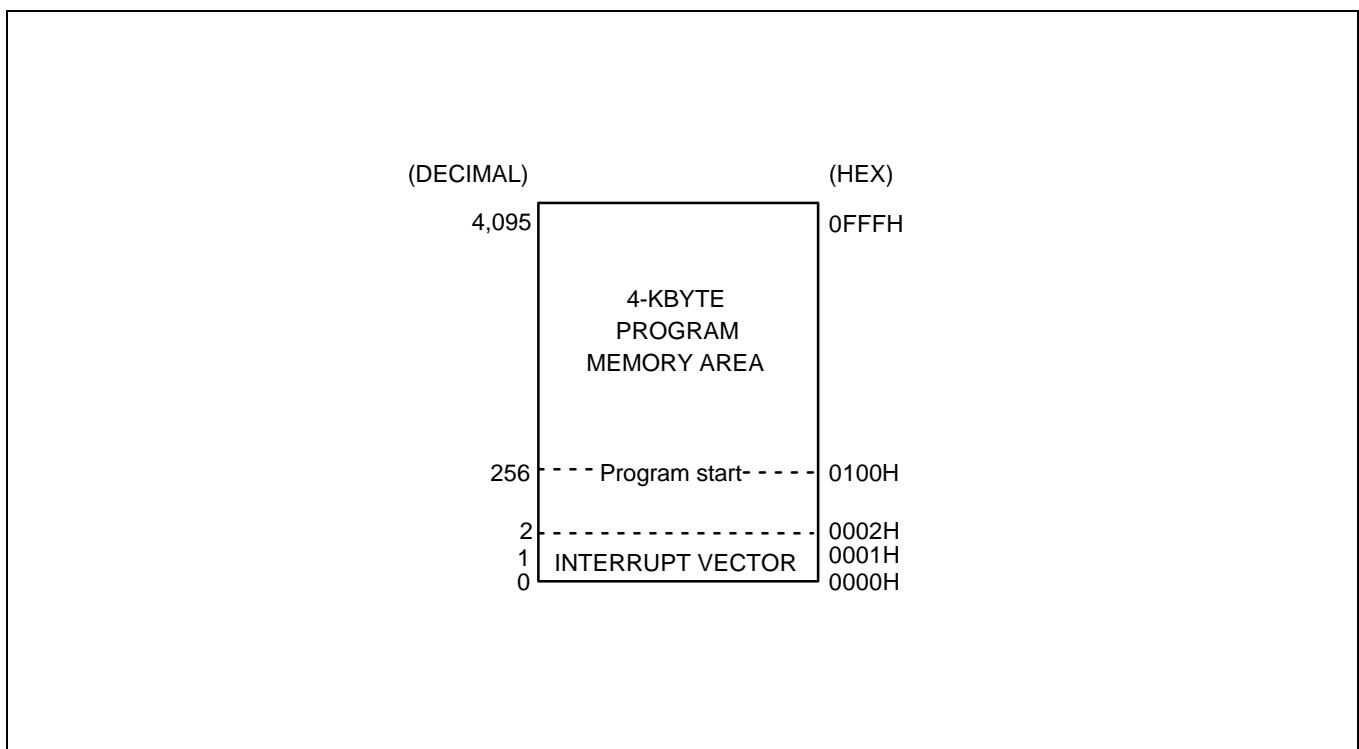


Figure 2-1. Program Memory Address Space

REGISTER ARCHITECTURE

The upper 64 bytes of the KS86C4004/C4104's internal register file are addressed as working registers, system control registers and peripheral control registers. The lower 192 bytes of internal register file(00H–BFH) is called the *general purpose register space*. The total addressable register space is thereby 256 bytes. 240 registers in this space can be accessed; 208 are available for general-purpose use.

For many SAM87Ri microcontrollers, the addressable area of the internal register file is further expanded by additional register pages at the general purpose register space(00H–BFH). This register file expansion is not implemented in the KS86C4004/C4104, however.

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

Table 2-1. Register Type Summary

Register Type	Number of Bytes
CPU and system control registers	10
Peripheral, I/O, and clock control and data registers	22
General-purpose registers (including the 16-bit common working register area)	208
Total Addressable Bytes	240

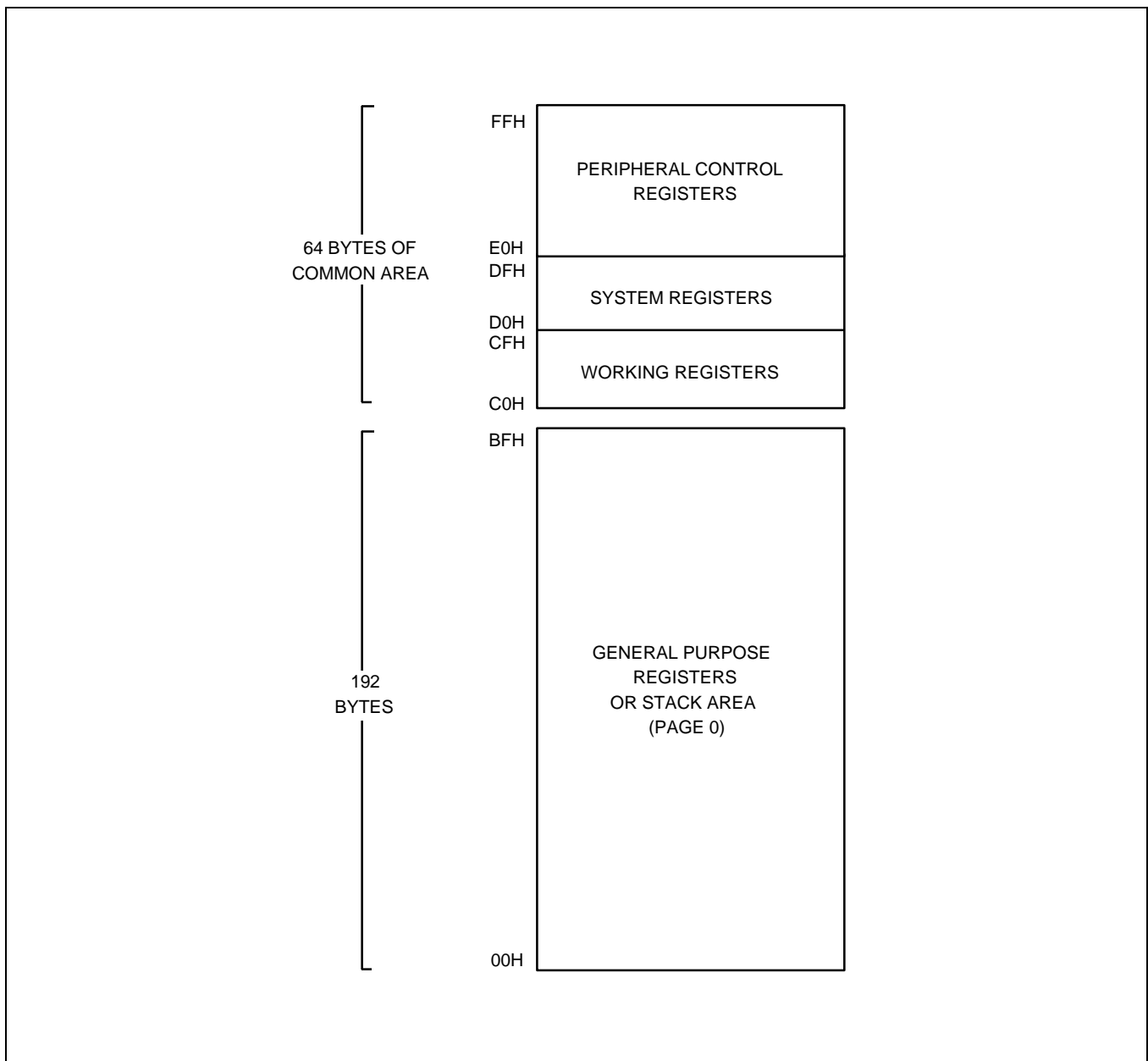


Figure 2-2. Internal Register File Organization

COMMON WORKING REGISTER AREA (C0H–CFH)

The SAM87Ri register architecture provides an efficient method of working register addressing that takes full advantage of shorter instruction formats to reduce execution time.

This 16-byte address range is called common area. That is, locations in this area can be used as working registers by operations that address any location on any page in the register file. Typically, these working registers serve as temporary buffers for data operations between different pages. However, because the KS86C4004/C4104 uses only page 0, you can use the common area for any internal data operation.

The Register (R) addressing mode can be used to access this area

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.

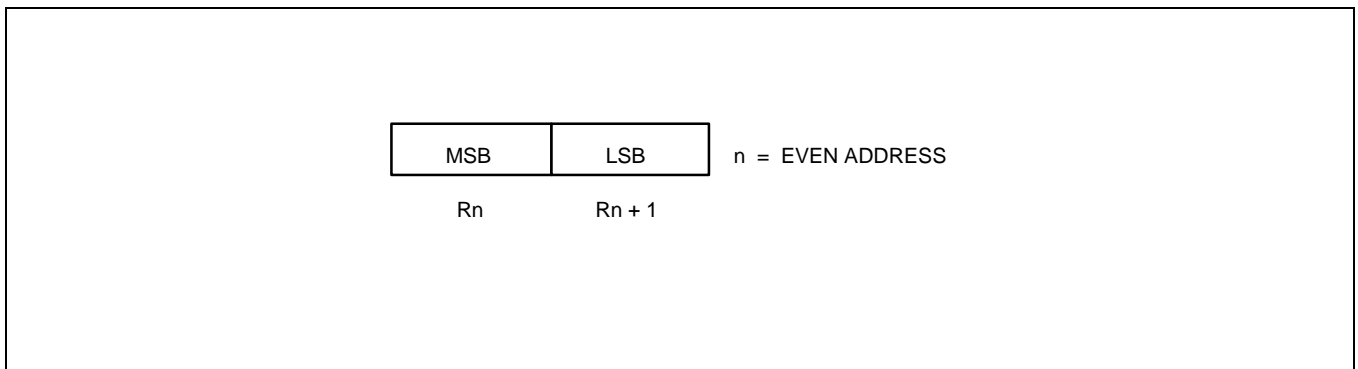


Figure 2-3. 16-Bit Register Pairs

PROGRAMMING TIP —Addressing the Common Working Register Area

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

Examples:

1.

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

Use working register addressing instead:

```
LD      R2,40H      ;R2 (C2H) ← the value in location 40H
```

2.

```
ADD     0C3H,#45H    ;Invalid addressing mode!
```

Use working register addressing instead:

```
ADD     R3,#45H     ;R3 (C3H) ← R3 + 45H
```

SYSTEM STACK

KS86-series microcontrollers use the system stack for subroutine calls and returns and to store data. The PUSH and POP instructions are used to control system stack operations. The KS86C4004/4104 architecture supports stack operations in the internal register file.

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.

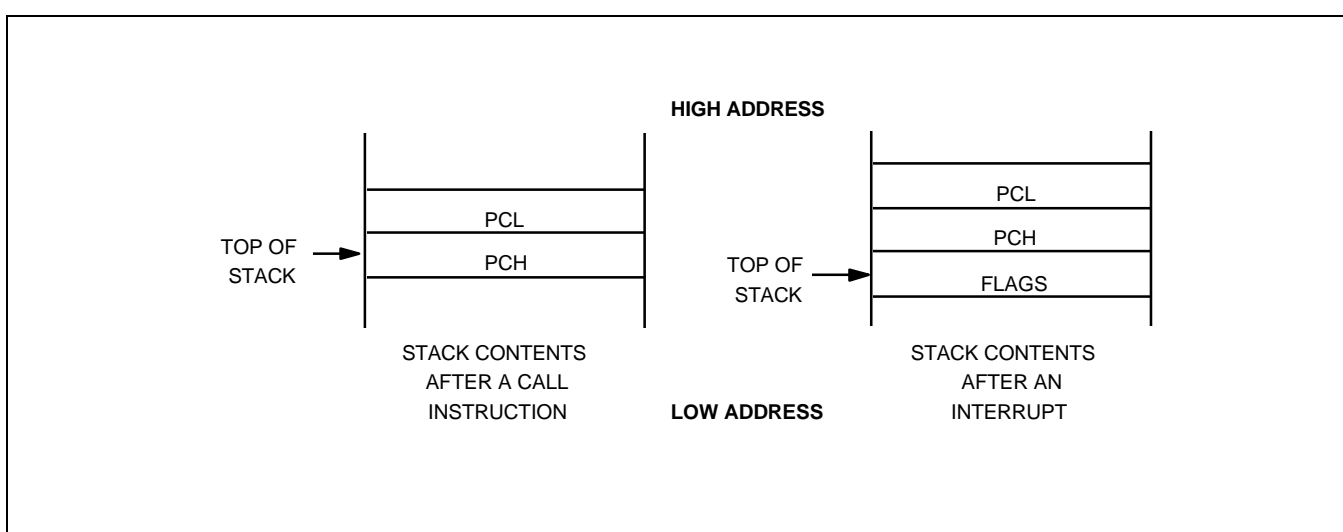


Figure 2-4. Stack Operations

Stack Pointer (SP)

Register location D9H contains the 8-bit stack pointer (SP) that is used for system stack operations. After a reset, the SP value is undetermined.

Because only internal memory space is implemented in the KS86C4004/C4104, the SP must be initialized to an 8-bit value in the range 00H–0C0H.

NOTE

In case a Stack Pointer is initialized to 00H, it is decreased to FFH when stack operation starts. This means that a Stack Pointer access invalid stack area.

 **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      SP,#0C0H      ; SP ← C0H (Normally, the SP is set to 0C0H by the
                      ; initialization routine)
.
.
.
PUSH   SYM            ; Stack address 0BFH ← SYM
PUSH   R15            ; Stack address 0BEH ← R15
PUSH   20H            ; Stack address 0BDH ← 20H
PUSH   R3             ; Stack address 0BCH ← R3
.
.
.
POP    R3             ; R3 ← Stack address 0BCH
POP    20H            ; 20H ← Stack address 0BDH
POP    R15            ; R15 ← Stack address 0BEH
POP    SYM            ; SYM ← Stack address 0BFH

```

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 SAM87Ri instructions may be condition codes, immediate data, or a location in the register file, program memory, or data memory.

The SAM87Ri instruction set supports six 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)
- Relative Address (RA)
- Immediate (IM)

REGISTER ADDRESSING MODE (R)

In Register addressing mode, the operand is the content of a specified register (see Figure 3-1). Working register addressing differs from Register addressing because it uses an 16-byte working register space in the register file and an 4-bit register within that space (see Figure 3-2).

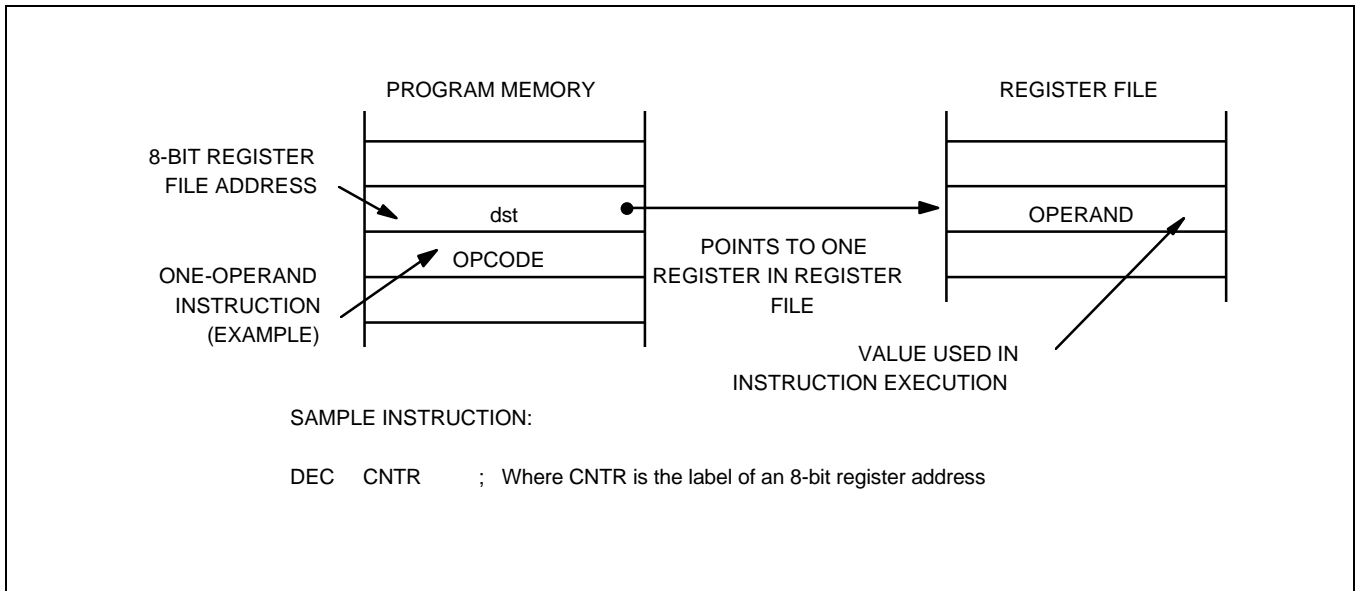


Figure 3-1. Register Addressing

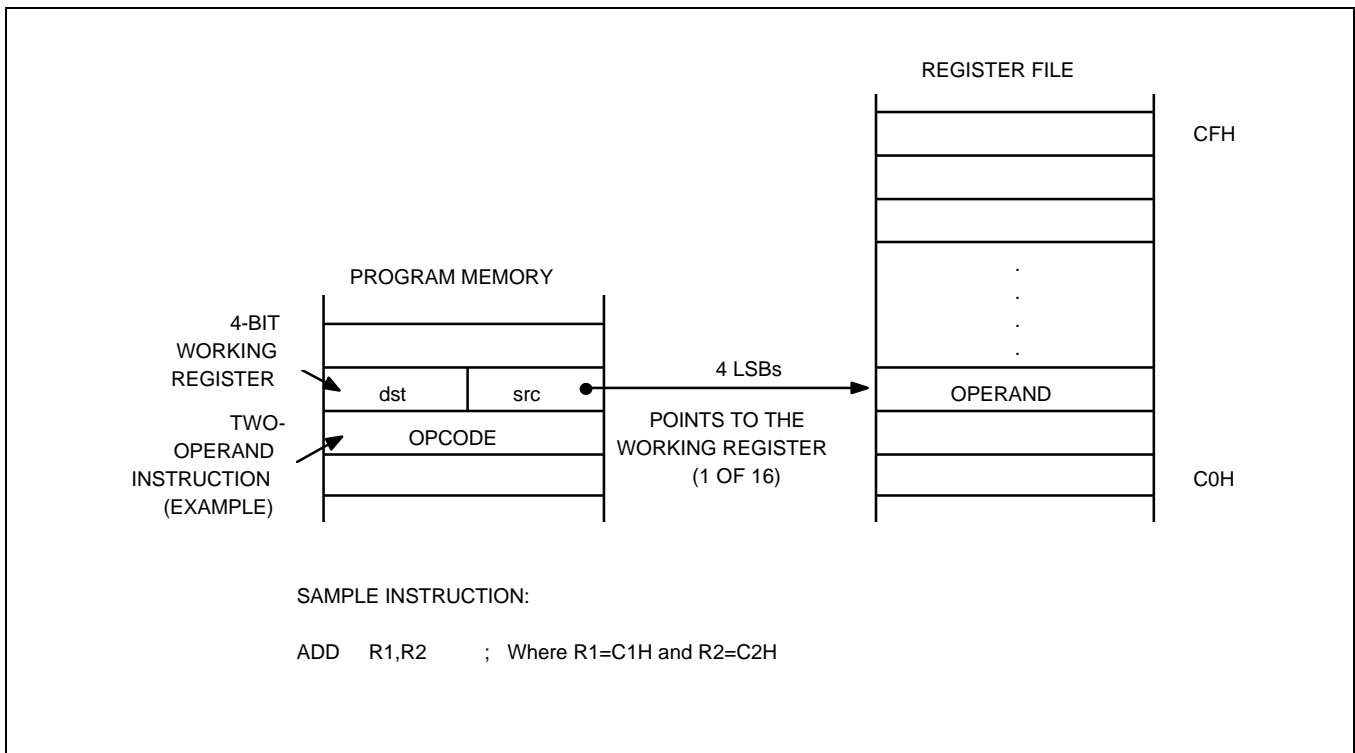


Figure 3-2. Working Register Addressing

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.

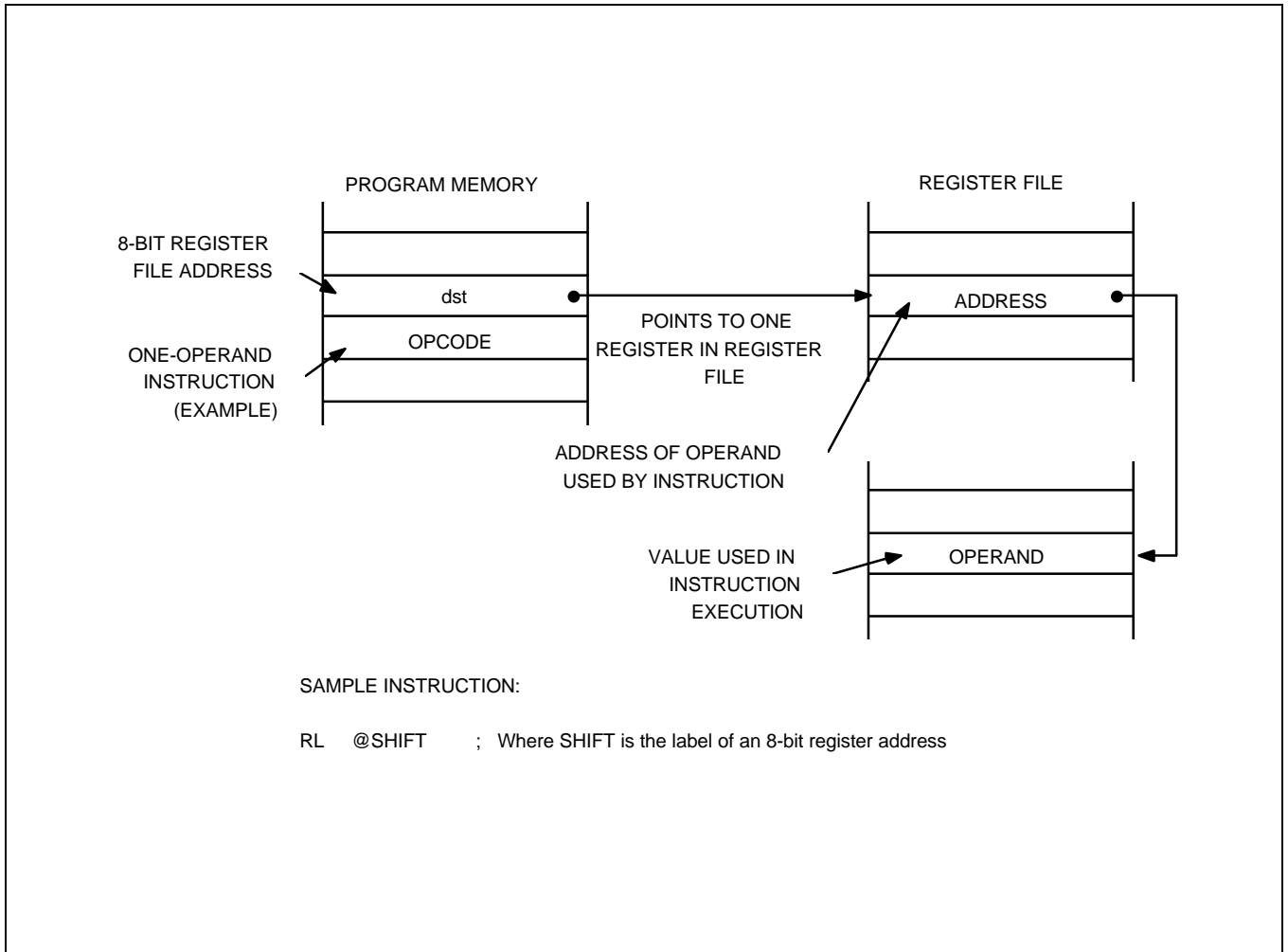


Figure 3-3. Indirect Register Addressing to Register File

INDIRECT REGISTER ADDRESSING MODE (Continued)

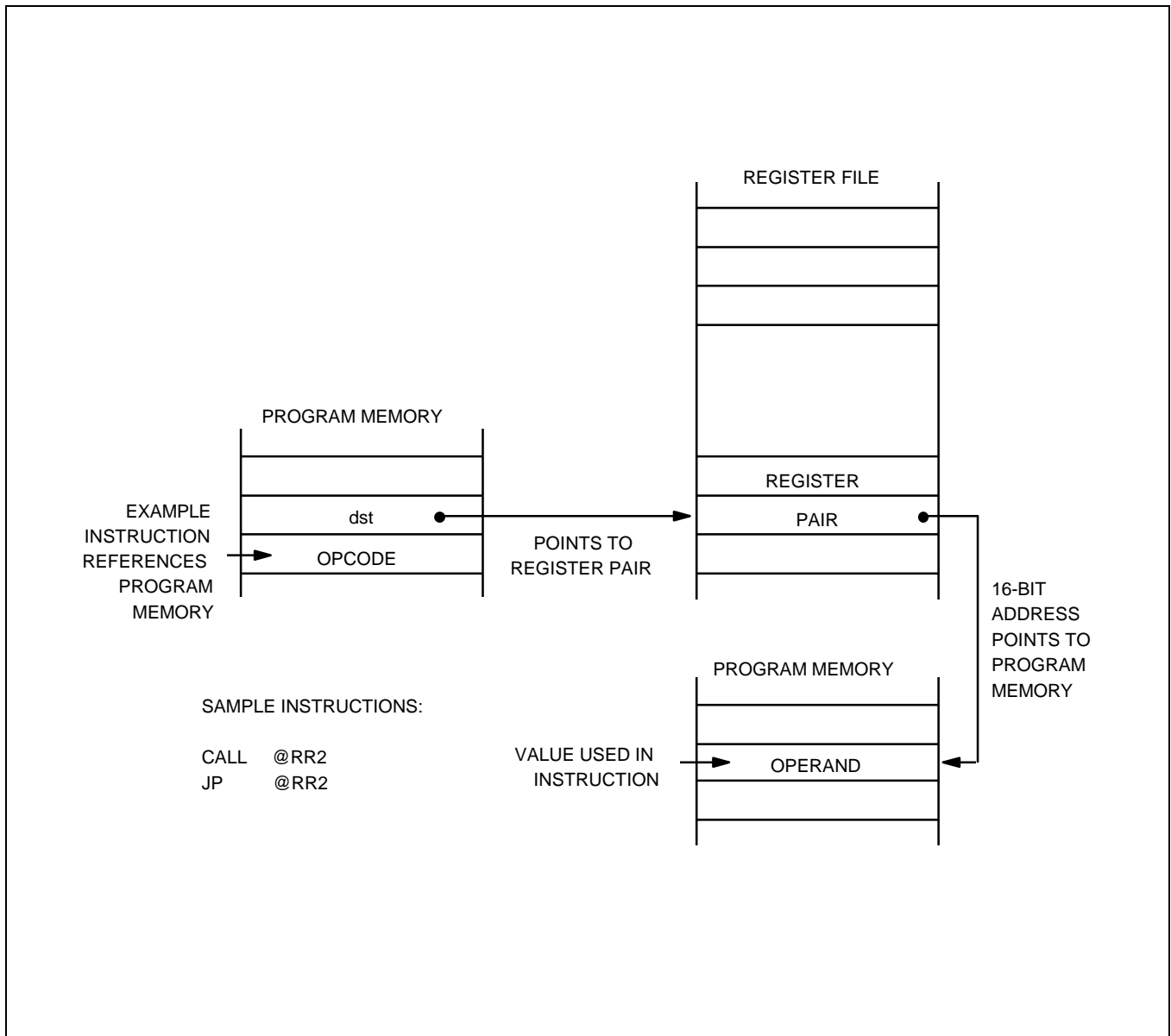


Figure 3-4. Indirect Register Addressing to Program Memory

INDIRECT REGISTER ADDRESSING MODE (Continued)

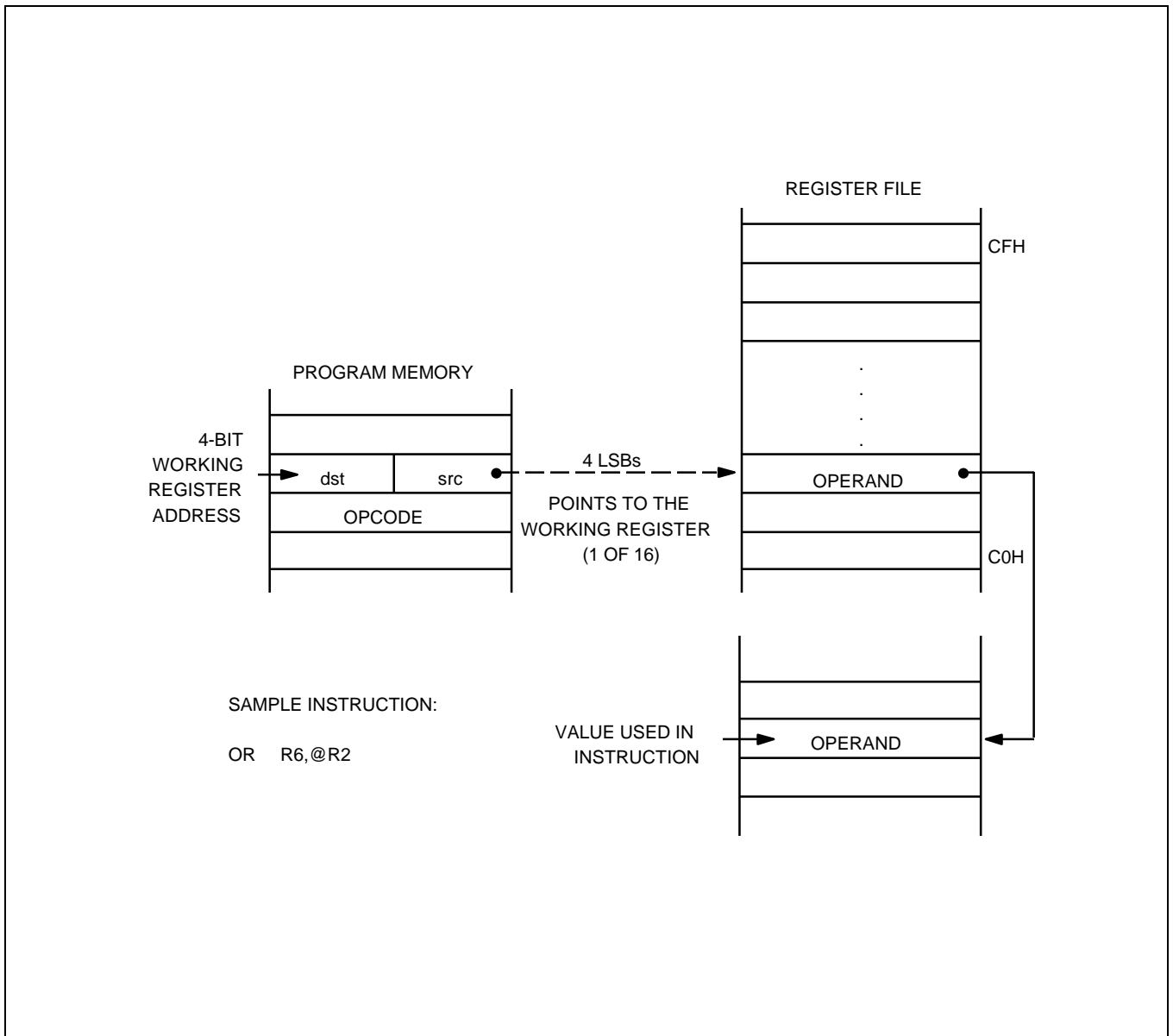


Figure 3-5. Indirect Working Register Addressing to Register File

INDIRECT REGISTER ADDRESSING MODE (Concluded)

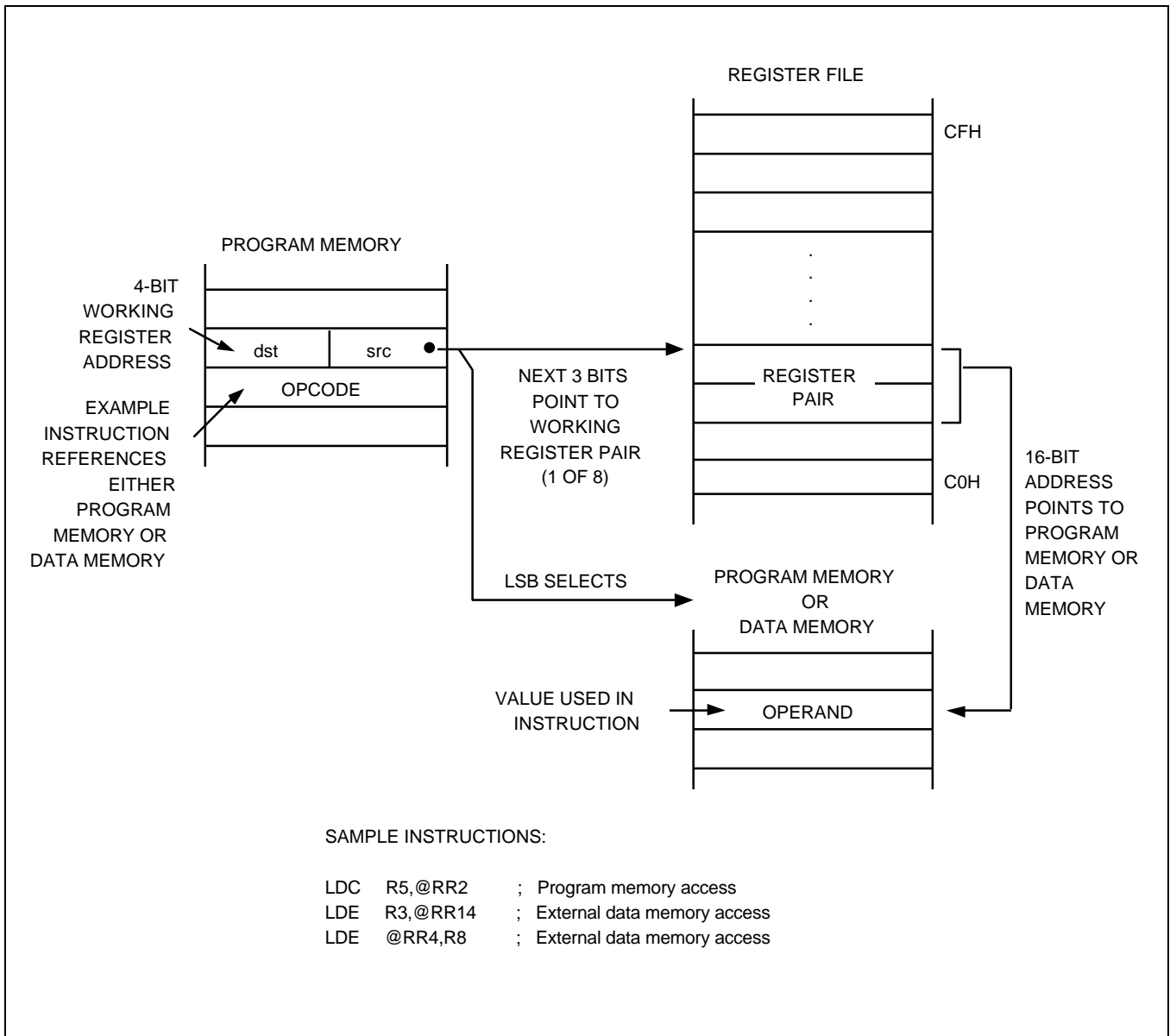


Figure 3-6. Indirect Working Register Addressing to Program or Data Memory

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.

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, external program memory, and for external data memory, when implemented.

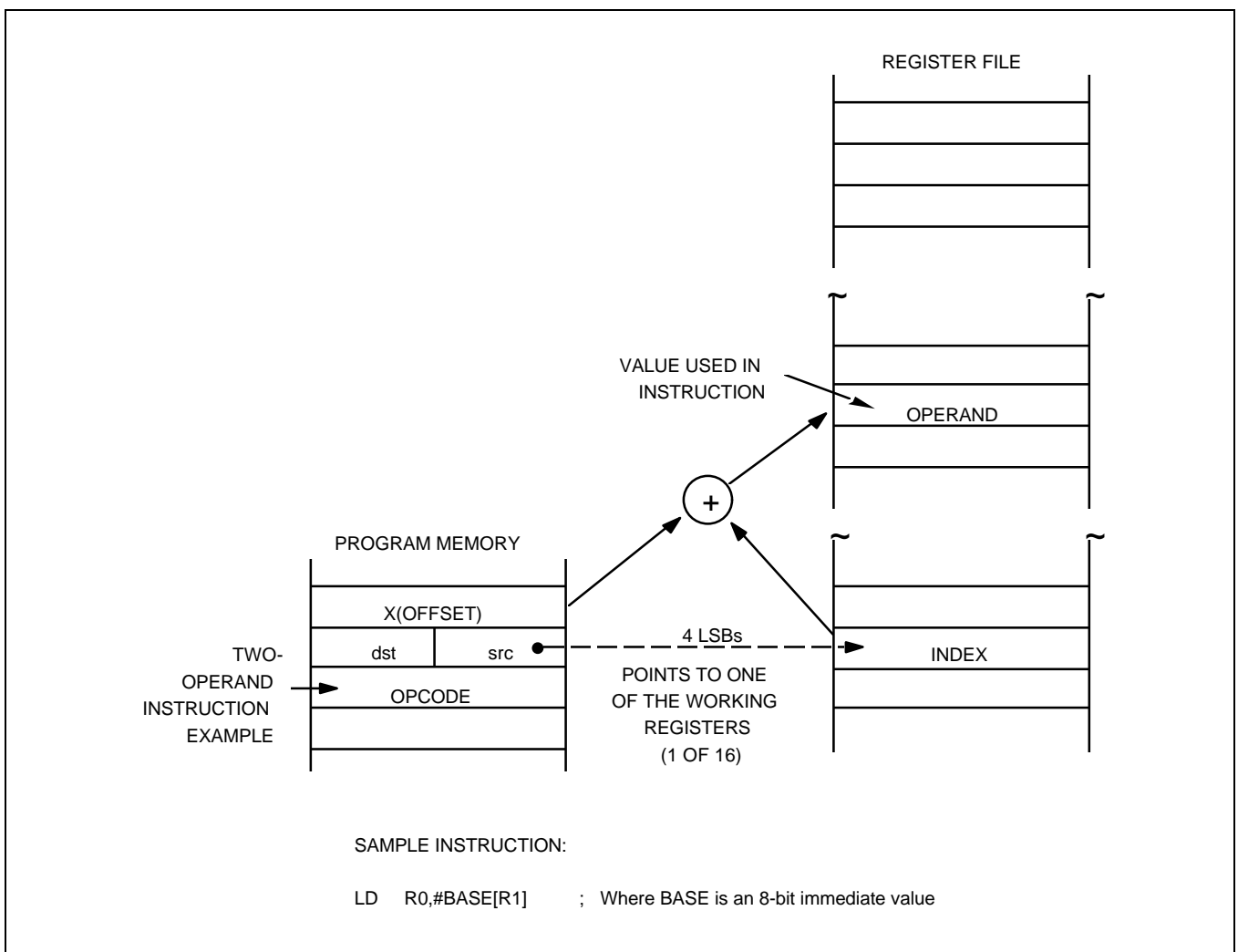


Figure 3–7. Indexed Addressing to Register File

INDEXED ADDRESSING MODE (Continued)

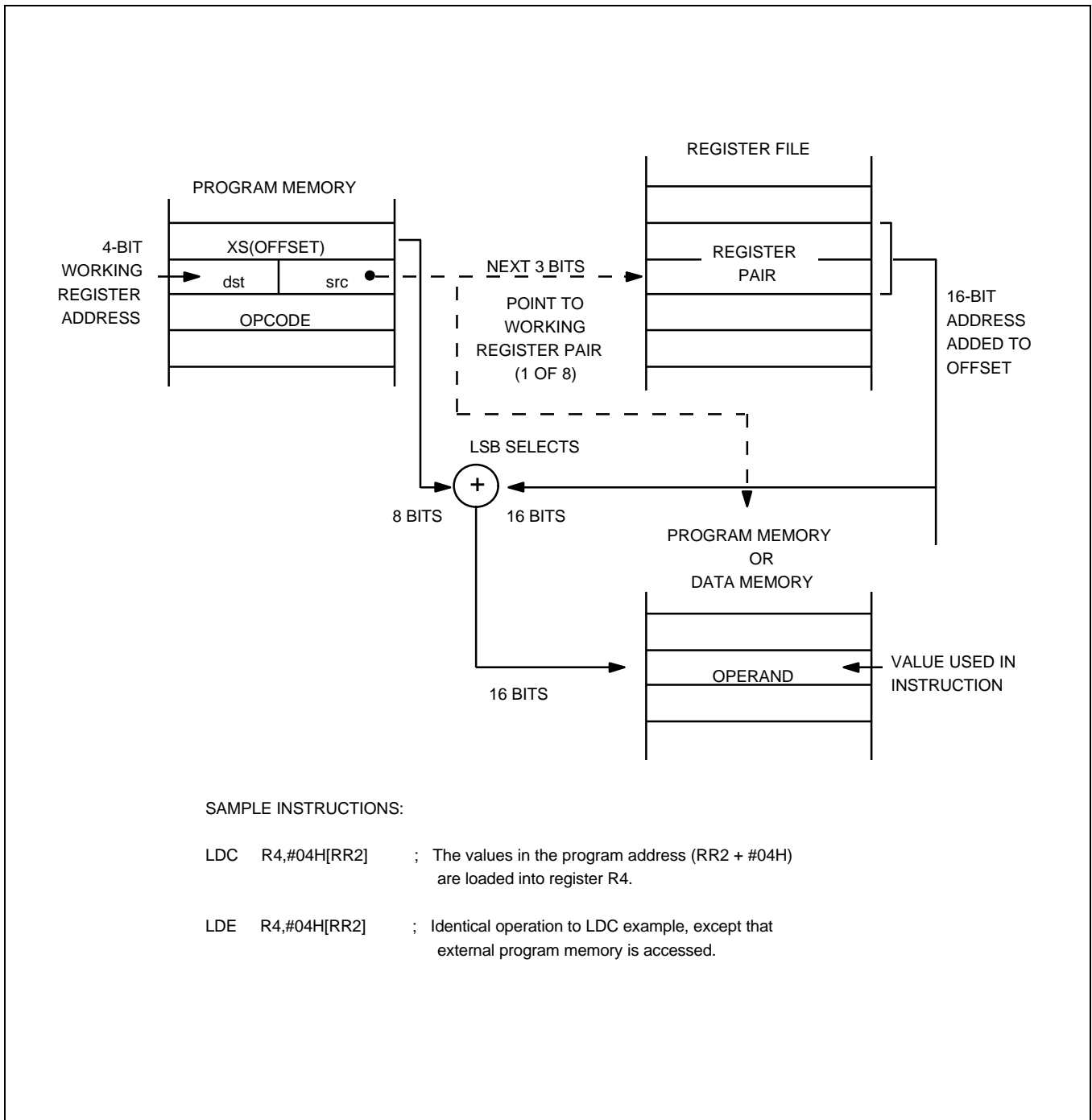


Figure 3-8. Indexed Addressing to Program or Data Memory with Short Offset

INDEXED ADDRESSING MODE (Concluded)

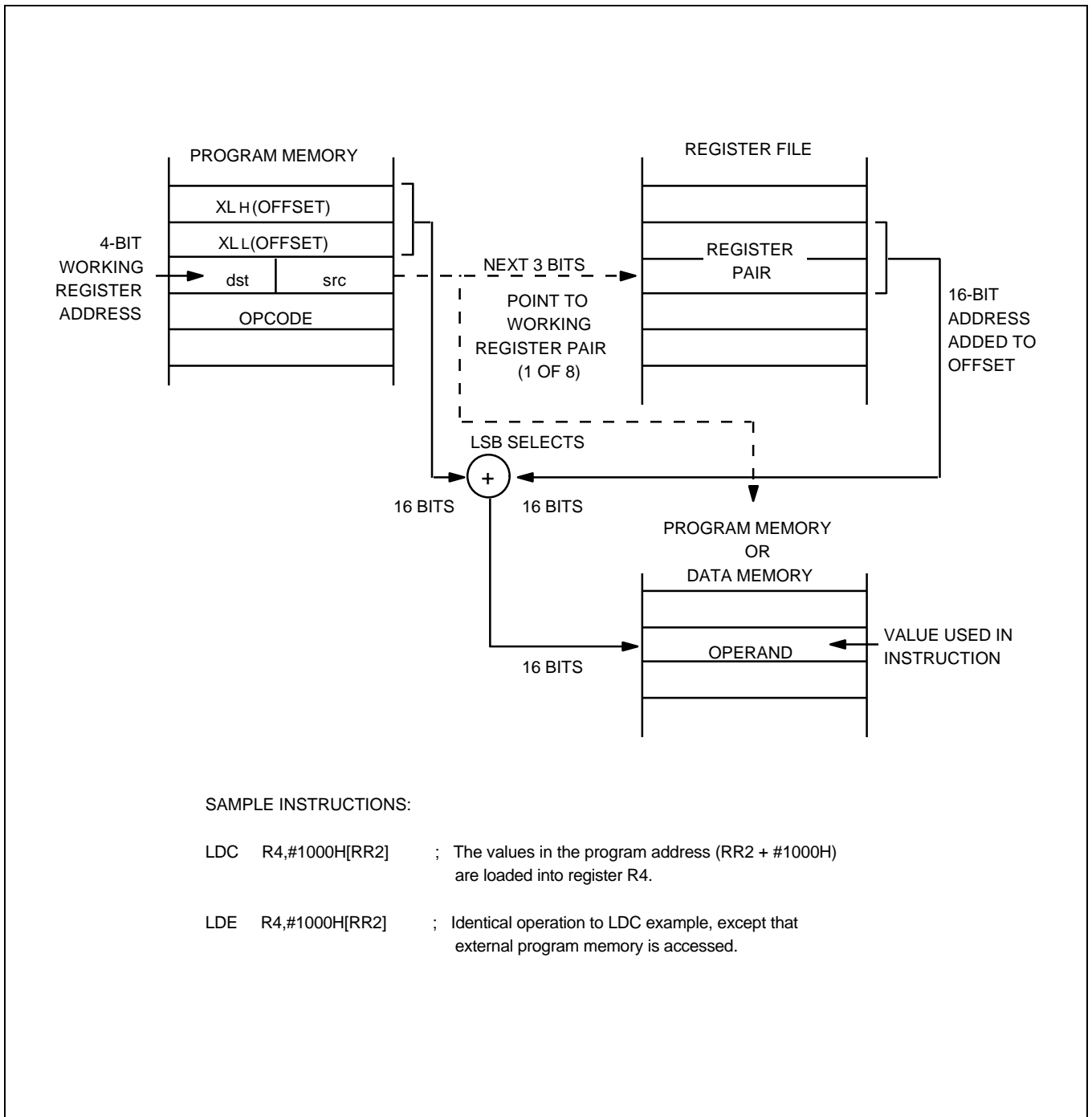


Figure 3-9. Indexed Addressing to Program or Data Memory with Long Offset

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.

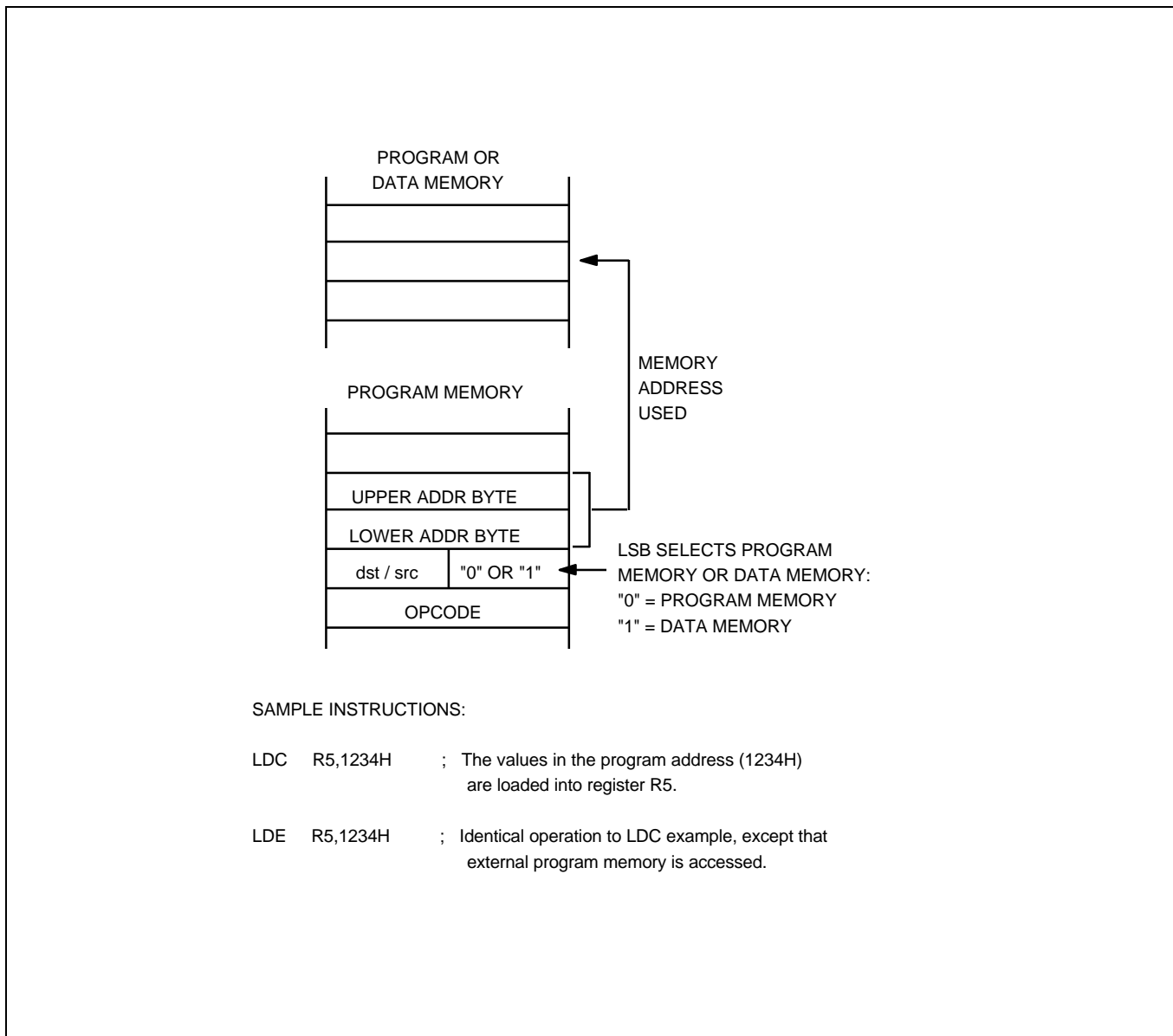


Figure 3–10. Direct Addressing for Load Instructions

DIRECT ADDRESS MODE (Continued)

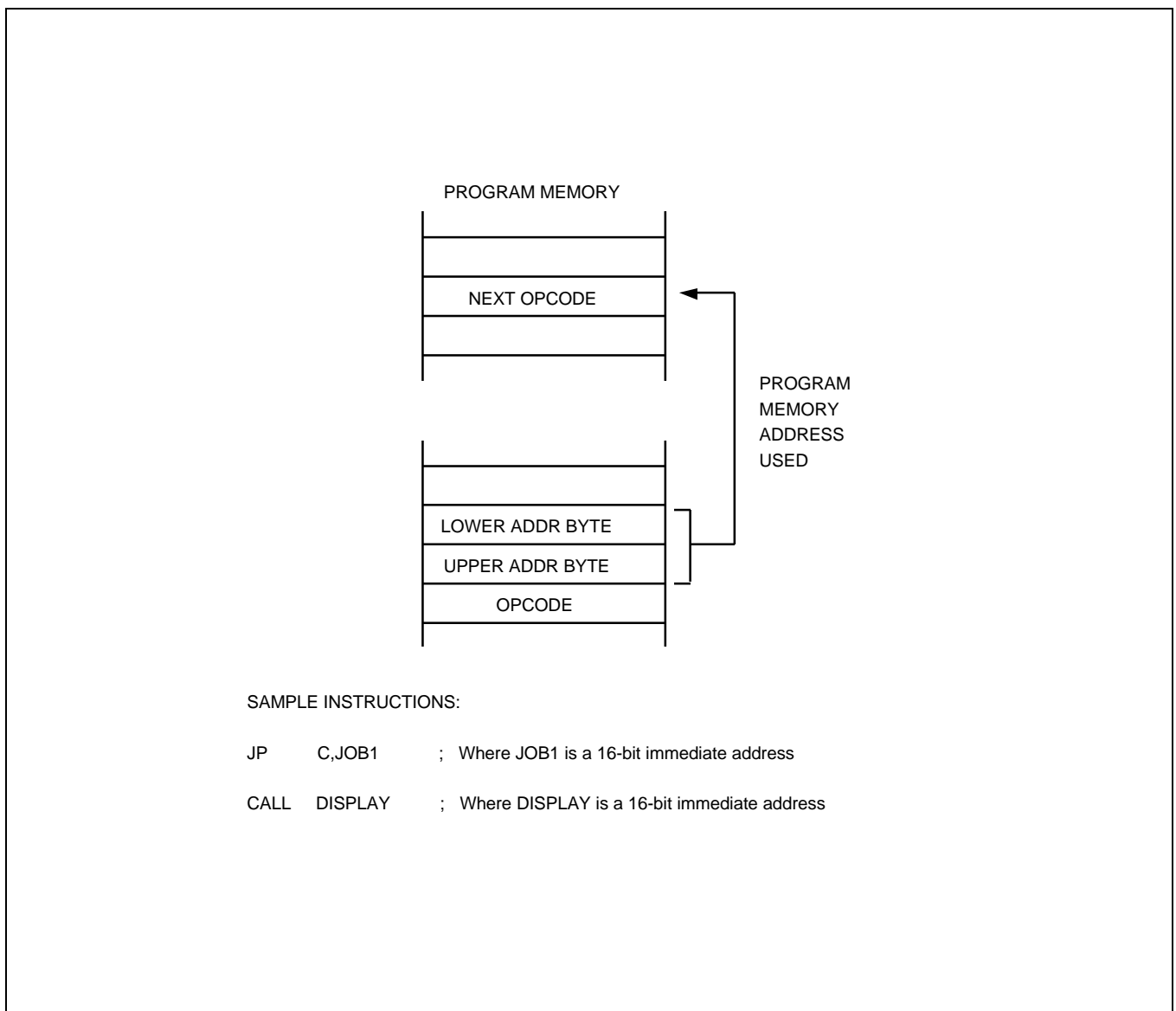


Figure 3-11. Direct Addressing for Call and Jump Instructions

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.

The instructions that support RA addressing is JR.

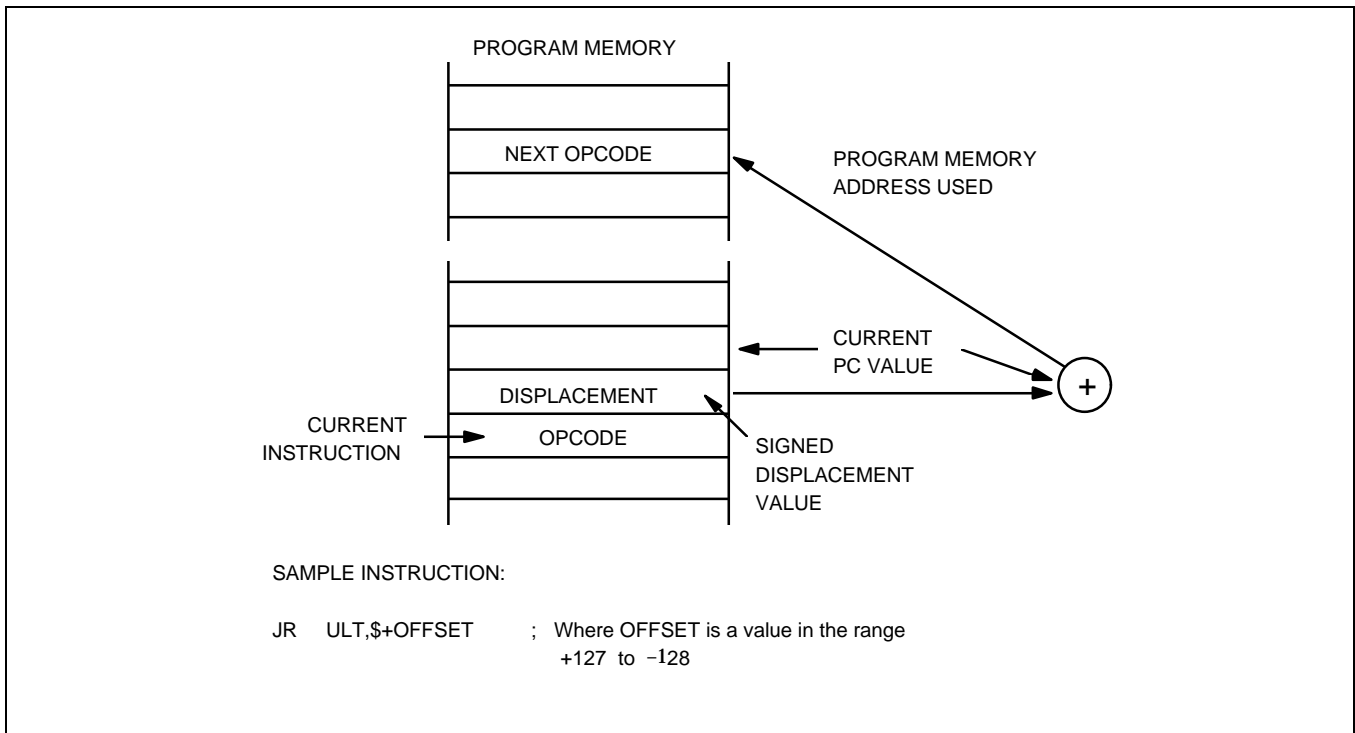


Figure 3-12. Relative Addressing

IMMEDIATE MODE (IM)

In Immediate (IM) addressing mode, the operand value used in the instruction is the value supplied in the operand field itself. Immediate addressing mode is useful for loading constant values into registers.

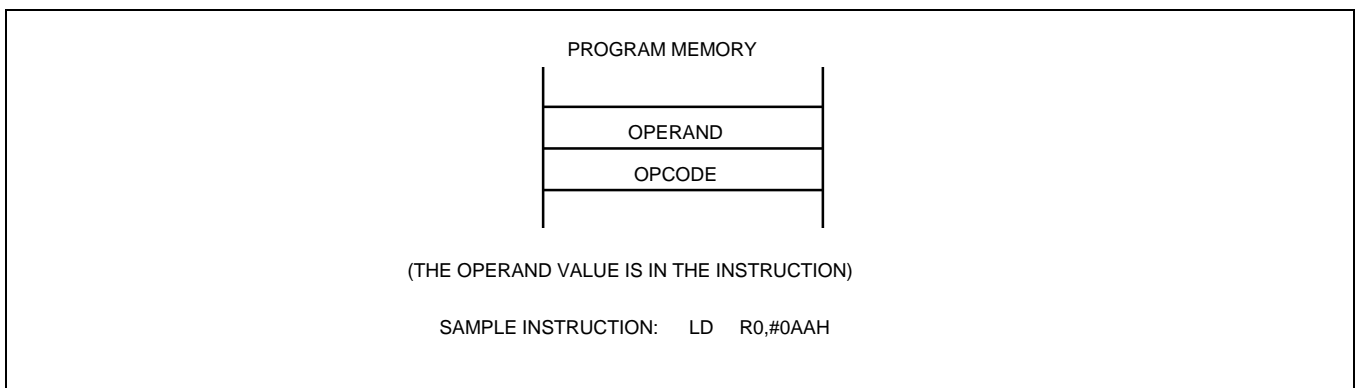


Figure 3-13. Immediate Addressing

4 CONTROL REGISTERS

OVERVIEW

In this section, detailed descriptions of the KS86C4004/C4104 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 Table 4-1. 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. System and Peripheral control Registers

Register Name	Mnemonic	Hex	R/W
Timer 0 counter register	T0CNT	D0H	R
Timer 0 data register	T0DATA	D1H	R/W
Timer 0 control register (high)	T0CONH	D2H	R/W
Timer 0 control register (low)	T0CONL	D3H	R/W
Clock control register	CLKCON	D4H	R/W
System flags register	FLAGS	D5H	R/W
Locations D6H–D8H are not mapped.			
Stack pointer register	SP	D9H	R/W
Location DAH is not mapped.			
Location DBH is reserved.			
Basic timer control register	BTCON	DCH	R/W
Basic timer counter	BTCNT	DDH	R
Location DEH is reserved.			
System mode register	SYM	DFH	R/W

Table 4–1. System and Peripheral Control Registers (Cont.)

Register Name	Mnemonic	Hex	R/W
Port 0 data register	P0	E0H	R/W
Port 1 data register	P1	E1H	R/W
Port 2 data register	P2	E2H	R/W
Port 3 data register	P3	E3H	R/W
Locations E4H–E5H are not mapped.			
Port 0 control register	P0CON	E6H	R/W
Port 0 pull-up resistor enable register	P0PUR	E7H	R/W
Port 0 N-channel open-drain mode register	P0PNE	E8H	R/W
Port 1 control register	P1CON	E9H	R/W
Port 2 control register	P2CON	EAH	R/W
Port 2 open-drain, pull-up resistor enable	P2DPUR	EBH	R/W
Port 2 interrupt pending register	P2PND	ECH	R/W
Location EDH is not mapped.			
Port 3 control register (high byte)	P3CONH	EEH	R/W
Port 3 control register (low byte)	P3CONL	EFH	R/W
Locations F0H–F1H are not mapped.			
Timer 1 counter register	T1CNT	F2H	R
Timer 1 control register	T1CON	F3H	R/W
Timer 1 data register	T1DATA	F4H	R/W
Zero-crossing detector control register	ZCMOD	F5H	R/W
8-bit prescaler for buzzer output	BUZPS	F6H	R/W
A/D control register	ADCON	F7H	R/W
A/D converter data register	ADDATAH	F8H	R
Location F9H is not mapped.			
PWM extension data register	PWMEX	FAH	R/W
PWM extension counter register	T0EXCNT	FBH	R
Locations FCH–FFH are not mapped.			

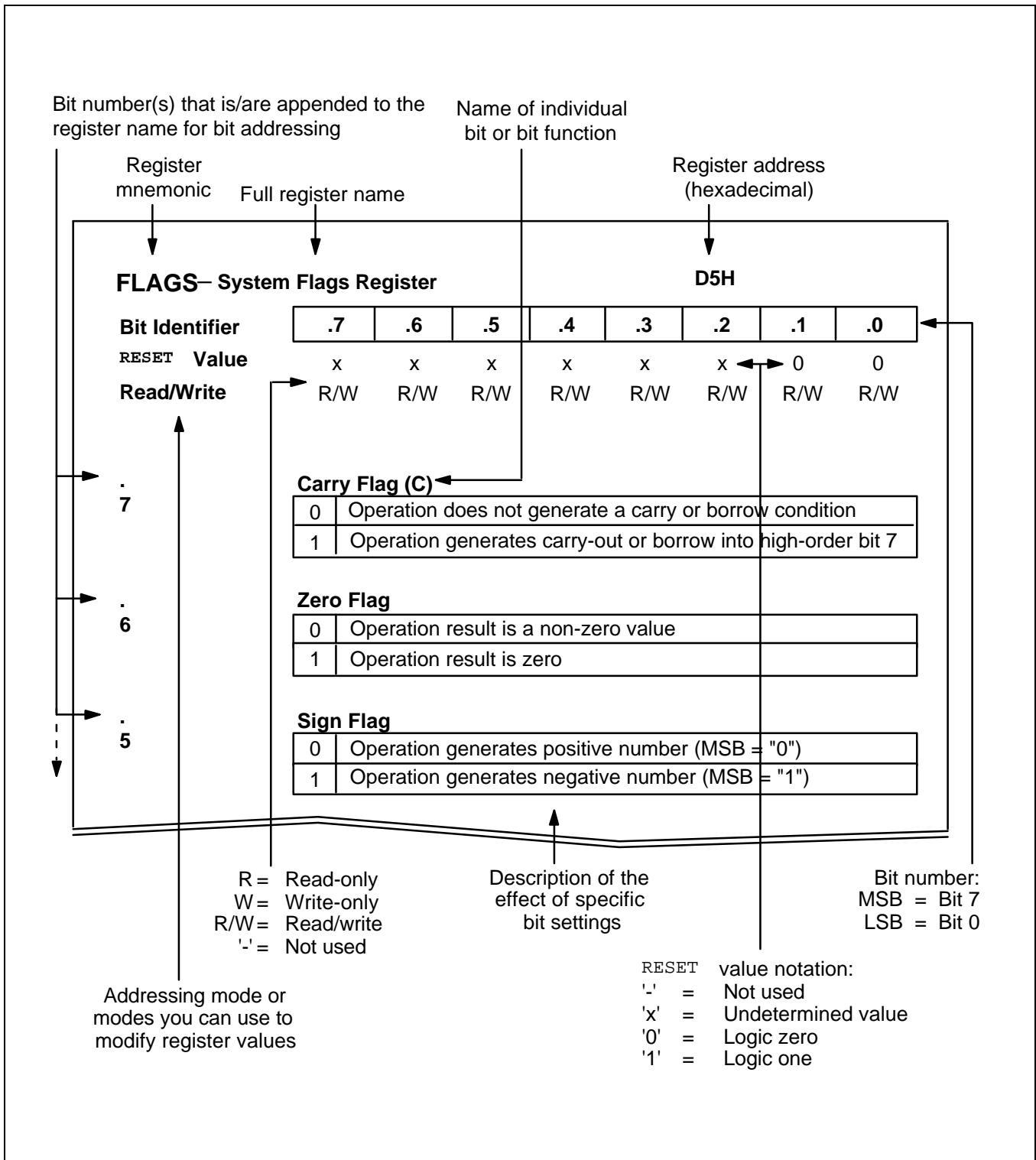


Figure 4-1. Register Description Format

ADCON—A/D Converter Control Register**F7H**

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

.7 Not used for KS86C4004/P4004/C4104/P4104

.6 and .4 A/D Converter Input Pin Selection Bits

0	0	0	ADC0 (P3.0)
0	0	1	ADC1 (P3.1)
0	1	0	ADC2 (P3.2)
0	1	1	ADC3 (P3.3)
1	0	0	ADC4 (P3.4)
1	0	1	ADC5 (P3.5) (Not used for KS86C4104/P4104)
1	1	0	ADC6 (P2.2) (Not used for KS86C4104/P4104)
1	1	1	ADC7 (P2.3) (Not used for KS86C4104/P4104)

.3 End-of-Conversion Status Bit

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

.2 – .1 Not used for KS86C4004/P4004/C4104/P4104

.0 Conversion Start Bit

0	No meaning
1	A/D conversion start

BTCON—Basic Timer Control Register

DCH

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

.7 – .4**Watchdog Timer Function Enable Bit**

1	0	1	0	Disable watchdog timer function
Any other value				Enable watchdog timer function

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

0	0	$f_{OSC}/4096$
0	1	$f_{OSC}/1024$
1	0	$f_{OSC}/128$
1	1	Invalid setting

.1**Basic Timer 8-bit Counter Clear Bit** (see Note)

0	No effect
1	Clear basic timer counter value

.0**Basic Timer Divider Clear Bit** (see Note)

0	No effect
1	Clear both dividers

NOTE: When you write a "1" to BTCON.0 (or BTCON.1), the basic timer counter (or basic timer divider) is cleared. The bit is then cleared automatically to "0".

BUZPS —6-Bit Prescaler for Buzzer Output

F6H

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

.7 Buzzer Output Enable Bit

0	Disable buzzer output (buzzer off)
1	Enable buzzer output (buzzer on)

.6 Buzzer Clock Selection Bit

0	Divided by 128 (fx/128)
1	Divided by 32 (fx/32)

.5 – .0 6-Bit Prescaler

0	0	0	0	0	0	divided by 2 [fx/(128 or 32)]
0	0	0	0	0	1	divided by 4 [fx/(128 or 32)]
0	0	0	0	1	0	divided by 6 [fx/(128 or 32)]
0	0	0	0	1	1	divided by 8 [fx/(128 or 32)]
•	•	•	•	•	•	divided by 2x(n+1) [fx/(128 or 32)]
•	•	•	•	•	•	
•	•	•	•	•	•	
1	1	1	1	1	1	divided by 128 [fx/(128 or 32)]

CLKCON – System Clock Control Register

D4H

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

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

0	Enable IRQ for main system oscillator wake-up function
1	Disable IRQ for main system oscillator wake-up function

.6 and .5**Not used for KS86C4004/P4004/C4104/P4104****.4 and .3****CPU Clock (System Clock) Selection Bits ⁽¹⁾**

0	0	Divide by 16 ($f_{OSC}/16$)
0	1	Divide by 8 ($f_{OSC}/8$)
1	0	Divide by 2 ($f_{OSC}/2$)
1	1	Non-divided clock (f_{OSC}) ⁽²⁾

.2 – .0**Not used for KS86C4004/P4004/C4104/P4104****NOTES:**

1. After a reset, the slowest clock (divided by 16) is selected as the system clock. To select faster clock speeds, load the appropriate values to CLKCON.3 and CLKCON.4.
2. f_{OSC} means oscillator frequency

FLAGS—System Flags Register

D5H

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

.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 $\leq +127$ or ≥ -128
1	Operation result is $> +127$ or < -128

.3-0.

Not used for KS86C4004/P4004/C4104/P4104

P0CON—Port 0 Control Register**E6H**

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

.7 **Port 0, P0.7 Configuration Bits (Not used for KS86C4104/P4104)**

0	Normal input
1	Push-pull output

.6 **Port 0, P0.6 Configuration Bits**

0	Normal input
1	Push-pull output

.5 **Port 0, P0.5 Configuration Bits**

0	Normal input
1	Push-pull output

.4 **Port 0, P0.4 Configuration Bits**

0	Normal input
1	Push-pull output

.3 **Port 0, P0.3 Configuration Bits**

0	Normal input
1	Push-pull output

.2 **Port 0, P0.2 Configuration Bits**

0	Normal input
1	Push-pull output

.1 **Port 0, P0.1 Configuration Bits**

0	Normal input
1	Push-pull output

.0 **Port 0, P0.0 Configuration Bits**

0	Normal input
1	Push-pull output

POPUR —Port 0 Pull-up Resistor Enable Register**E7H**

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

.7 **Port 0.7 Pull-up Resistor Enable Bit (Not used for KS86C4104/P4104)**

0	Disable pull-up
1	Enable pull-up

.6 **Port 0.6 Pull-up Resistor Enable Bit**

0	Disable pull-up
1	Enable pull-up

.5 **Port 0.5 Pull-up Resistor Enable Bit**

0	Disable pull-up
1	Enable pull-up

.4 **Port 0.4 Pull-up Resistor Enable Bit**

0	Disable pull-up
1	Enable pull-up

.3 **Port 0.3 Pull-up Resistor Enable Bit**

0	Disable pull-up
1	Enable pull-up

.2 **Port 0.2 Pull-up Resistor Enable Bit**

0	Disable pull-up
1	Enable pull-up

.1 **Port 0.1 Pull-up Resistor Enable Bit**

0	Disable pull-up
1	Enable pull-up

.0 **Port 0.0 Pull-up Resistor Enable Bit**

0	Disable pull-up
1	Enable pull-up

POPNE —Port 0 N-Channel Open-drain Mode Register**E8H**

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

.7 Port 0.7 N-Channel Open-drain Enable Bit (Not used for KS86C4104/P4104)

0	Configure P0.7 as a push-pull
1	Configure P0.7 as a n-channel open-drain

.6 Port 0.6 N-Channel Open-drain Enable Bit

0	Configure P0.6 as a push-pull
1	Configure P0.6 as a n-channel open-drain

.5 Port 0.5 N-Channel Open-drain Enable Bit

0	Configure P0.5 as a push-pull
1	Configure P0.5 as a n-channel open-drain

.4 Port 0.4 N-Channel Open-drain Enable Bit

0	Configure P0.4 as a push-pull
1	Configure P0.4 as a n-channel open-drain

.3 Port 0.3 N-Channel Open-drain Enable Bit

0	Configure P0.3 as a push-pull
1	Configure P0.3 as a n-channel open-drain

.2 Port 0.2 N-Channel Open-drain Enable Bit

0	Configure P0.2 as a push-pull
1	Configure P0.2 as a n-channel open-drain

.1 Port 0.1 N-Channel Open-drain Enable Bit

0	Configure P0.1 as a push-pull
1	Configure P0.1 as a n-channel open-drain

.0 Port 0.0 N-Channel Open-drain Enable Bit

0	Configure P0.0 as a push-pull
1	Configure P0.0 as a n-channel open-drain

P1CON—Port 1 Control Register**E9H**

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

.7 and .6**Port 1, P1.3/CLO Configuration Bits (Not used for KS86C4104/P4104)**

0	0	Schmitt trigger input
0	1	Schmitt trigger input; pull-up resistor enable
1	0	Push-pull output
1	1	Alternative function (CLO output)

.5 and .4**Port 1, P1.2/T0 Configuration Bits**

0	0	Schmitt trigger input (or T0 Capture input)
0	1	Schmitt trigger input; pull-up resistor enable
1	0	Push-pull output
1	1	Alternative function (T0 output: match or PWM)

.3 and .2**Port 1, P1.1/BUZ Configuration Bits**

0	0	Schmitt trigger
0	1	Schmitt trigger input; pull-up resistor enable
1	0	Push-pull output
1	1	Alternative function (BUZ output)

.1 and .0**Port 1, P1.0 /ZCD Configuration Bits**

0	0	Schmitt trigger
0	1	Schmitt trigger input; pull-up resistor enable
1	0	Push-pull output
1	1	Alternative function (ZCD input); ZCD enable

P2CON—Port 2 Control Register

EAH

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

.7 and .6**Port 2, P2.3 /AD7 Configuration Bits (Not used for KS86C4104/P4104)**

0	0	Schmitt trigger input; A/D converter off
0	1	Schmitt trigger input; A/D converter off
1	0	Push-pull output
1	1	A/D converter input (AD7); Schmitt trigger input off

.5 and .4**Port 2, P2.2 /AD6 Configuration Bits (Not used for KS86C4104/P4104)**

0	0	Schmitt trigger input; A/D converter off
0	1	Schmitt trigger input; A/D converter off
1	0	Push-pull output
1	1	A/D converter input (AD6); Schmitt trigger input off

.3 and .2**Port 2, P2.1 /INT1 Configuration Bits (Not used for KS86C4104/P4104)**

0	0	Schmitt trigger input; INT1 interrupt disable
0	1	Schmitt trigger input; interrupt on falling edge
1	0	Push-pull output
1	1	Schmitt trigger input; interrupt on rising edge

.1 and .0**Port 2, P2.0 /INT0 Configuration Bits**

0	0	Schmitt trigger input; INT0 interrupt disable
0	1	Schmitt trigger input; interrupt on falling edges
1	0	Push-pull output
1	1	Schmitt trigger input; interrupt on rising edges

P2DPUR —Port 2 Open-drain Enable & Pull-up Resistor Enable Register EBH

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

.7 Port 2.3/AD7, Open-drain Enable Bit (Not used for KS86C4104/P4104)

0	Configure P2.3 as a push-pull
1	Configure P2.3 as a n-channel open drain

.6 Port 2.2/AD6, Open-drain Enable Bit (Not used for KS86C4104/P4104)

0	Configure P2.2 as a push-pull
1	Configure P2.2 as a n-channel open drain

.5 Port 2.1/INT1, Open-drain Enable Bit (Not used for KS86C4104/P4104)

0	Configure P2.1 as a push-pull
1	Configure P2.1 as a n-channel open drain

.4 Port 2.0/INT0, Open-drain Enable Bit

0	Configure P2.0 as a push-pull
1	Configure P2.0 as a n-channel open drain

.3 Port 2.3/AD7, Pull-up Resistor Enable Bit (Not used for KS86C4104/P4104)

0	Disable pull-up
1	Enable pull-up

.2 Port 2.2/AD6, Pull-up Resistor Enable Bit (Not used for KS86C4104/P4104)

0	Disable pull-up
1	Enable pull-up

.1 Port 2.1/INT1, Pull-up Resistor Enable Bit (Not used for KS86C4104/P4104)

0	Disable pull-up
1	Enable pull-up

.0 Port 2.0/INT0, Pull-up Resistor Enable Bit

0	Disable pull-up
1	Enable pull-up

NOTE: In order to use the open-drain output mode the push-pull output bit in the P2CON should be set first.

P2PND —Port 2 Interrupt Pending Register

ECH

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

.7 – .2**Not used for KS86C4004/P4004/C4104/P4104****.1****P2.1/ INT1, Interrupt Pending Bit (Not used for KS86C4104/P4104)**

0	No interrupt pending (<i>when read</i>)
0	Clear this pending bit (<i>when write</i>)
1	Interrupt is pending (<i>when read</i>) / No effect (<i>when write</i>)

.0**P2.0/ INT0, Interrupt Pending Bit**

0	No interrupt pending (<i>when read</i>)
0	Clear this pending bit (<i>when write</i>)
1	Interrupt is pending (<i>when read</i>) / No effect (<i>when write</i>)

NOTES:

1. To clear an interrupt pending condition at a port2 pin, you must write a "0" to the corresponding P2PND bit location.
2. To avoid programming errors, we recommend using load instructions when manipulating P2PND values.

P3CONH—Port 3 Control Register (High Byte)

EEH

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

.7 and .4

Not used for KS86C4004/P4004/C4104/P4104

.3 and .2

Port 3, P3.5/ADC5 Configuration Bits (Not used for KS86C4104/P4104)

0	0	Schmitt trigger input; A/D converter off
0	1	Schmitt trigger input, pull-up resistor enabled; A/D converter off
1	0	Push-pull output
1	1	A/D converter input (ADC5); Schmitt trigger input off

.1 and .0

Port 3, P3.4/ADC4 Configuration Bits

0	0	Schmitt trigger input; A/D converter off
0	1	Schmitt trigger input, pull-up resistor enabled; A/D converter off
1	0	Push-pull output
1	1	A/D converter input (ADC4); Schmitt trigger input off

P3CONL—Port 3 Control Register (Low Byte)

EFH

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

.7 and .6**Port 3, P3.3/ADC3 Configuration Bits**

0	0	Schmitt trigger input; A/D converter off
0	1	Schmitt trigger input, pull-up resistor enabled; A/D converter off
1	0	Push-pull output
1	1	A/D converter input (ADC3); Schmitt trigger input off

.5 and .4**Port 3, P3.2/ADC2 Configuration Bits**

0	0	Schmitt trigger input; A/D converter off
0	1	Schmitt trigger input, pull-up resistor enabled; A/D converter off
1	0	Push-pull output
1	1	A/D converter input (ADC2); Schmitt trigger input off

.3 and .2**Port 3, P3.1/ADC1 Configuration Bits**

0	0	Schmitt trigger input; A/D converter off
0	1	Schmitt trigger input, pull-up resistor enabled; A/D converter off
1	0	Push-pull output
1	1	A/D converter input (ADC1); Schmitt trigger input off

.1 and .0**Port 3, P3.0/ADC0 Configuration Bits**

0	0	Schmitt trigger input; A/D converter off
0	1	Schmitt trigger input, pull-up resistor enabled; A/D converter off
1	0	Push-pull output
1	1	A/D converter input (ADC0); Schmitt trigger input off

SYM—System Mode Register

DFH

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

.7 – .3 **Not used for KS86C4004/P4004/C4104/P4104**

.2 **Global Interrupt Enable Bit** ^(note)

0	Disable all interrupt (DI instruction)
1	Enable all interrupt (EI Instruction)

.1 and .0

Page Selection Bits

0	0	page 0
0	1	page 1 (not used for KS86C4004/P4004/C4104/P4104)
1	0	page 2 (not used for KS86C4004/P4004/C4104/P4104)
1	1	page 3 (not used for KS86C4004/P4004/C4104/P4104)

NOTE: Following a reset, you enable global interrupt processing by executing an EI instruction (not by writing a “1” to SYM.2).

T0CONH—TIMER 0 Control Register (High Byte)

D2H

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0 (8)
RESET Value	–	–	–	–	–	–	–	0
Read/Write	–	–	–	–	–	–	–	R/W

.F – .9

Not used for KSC4004/P4004/C4104/P4104

.8

Timer 0 Overflow Interrupt Pending Bit (overflow interrupt)

0	No interrupt pending (<i>when read</i>)
0	Clear Pending bit (<i>when write</i>)
1	Interrupt is pending (<i>when read</i>)

T0CONL—TIMER 0 Control Register (Low Byte)**D3H**

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

.7 and .6**Timer 0 Input Clock Selection Bits**

0	0	$f_{osc}/4096$
0	1	$f_{osc}/256$
1	0	$f_{osc}/8$
1	1	$f_{osc}/1$

.5 and .4**Timer 0 Operating Mode Selection Bits**

0	0	Interval mode
0	1	Capture mode (capture on rising edge, counter running, OVF)
1	0	Capture mode (capture on falling edge, counter running, OVF)
0	0	PWM mode (OVF interrupt can occur)

.3**Timer 0 Counter Clear Bit**

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

.2**Timer 0 Overflow Interrupt Enable Bit**

0	Disable overflow interrupt
1	Enable overflow interrupt

.1**Timer 0 Interrupt Enable Bit**

0	Disable interrupt
1	Enable interrupt

.0**Timer 0 Interrupt Pending Bit (Capture or Match Interrupt)**

0	No interrupt pending (<i>when read</i>)
0	Clear pending bit (<i>when write</i>)
1	Interrupt is pending (<i>when read</i>)

NOTE: When you write a “1” to T0CONL.3 the timer 0 counter is cleared. The bit is then cleared automatically to “0”.

T1CON—Timer 1 Control Register**F3H**

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

.7 and .6 **Not used for KS86C4004/P4004/C4104/P4104**

.5 and .4 **Timer 1 Input Clock Selection Bits**

0	0	$f_{osc}/512$
0	1	$f_{osc}/256$
1	0	$f_{osc}/128$
1	1	$f_{osc}/64$

.3 **Timer 1 Counter Automatic Clear Enable Bit**

0	Disable
1	Enable ZCD clear signal to clear the timer 1 counter

.2 **Timer 1 Counter Clear Enable Bit**

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

.1 **Timer 1 Interrupt Enable Bit**

0	Disable interrupt
1	Enable interrupt

.0 **Timer 1 Interrupt Pending Bit**

0	No interrupt pending (<i>when read</i>)
0	Clear pending bit (<i>when write</i>)
1	Interrupt is pending (<i>when read</i>)

NOTE: When you write a “1” to T1CON.2, the timer 0 counter is cleared to “0”. The bit is then cleared automatically to “0”.

ZCMOD—Zero Crossing Detection Control Register**F5H**

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

.7 and .4 **Not used for KS86C4004/P4004/C4104/P4104**

.3 and .2 **Interrupt Mode Selection Bits**

0	0	Interrupt on falling edge
0	1	Interrupt on rising edge
1	0	Interrupt on both edge
1	1	Not used

.1 **ZCD Interrupt Enable Bit**

0	Disable interrupt
1	Enable interrupt

.0 **ZCD Interrupt Pending Bit**

0	No interrupt pending (<i>when read</i>)
0	Clear pending bit (<i>when write</i>)
1	Interrupt is pending (<i>when read</i>)

NOTES

5

INTERRUPT STRUCTURE

OVERVIEW

The SAM87Ri interrupt structure has two basic components: a vector, and sources. The number of interrupt sources can be serviced through an interrupt vector which is assigned in ROM address 0000H.

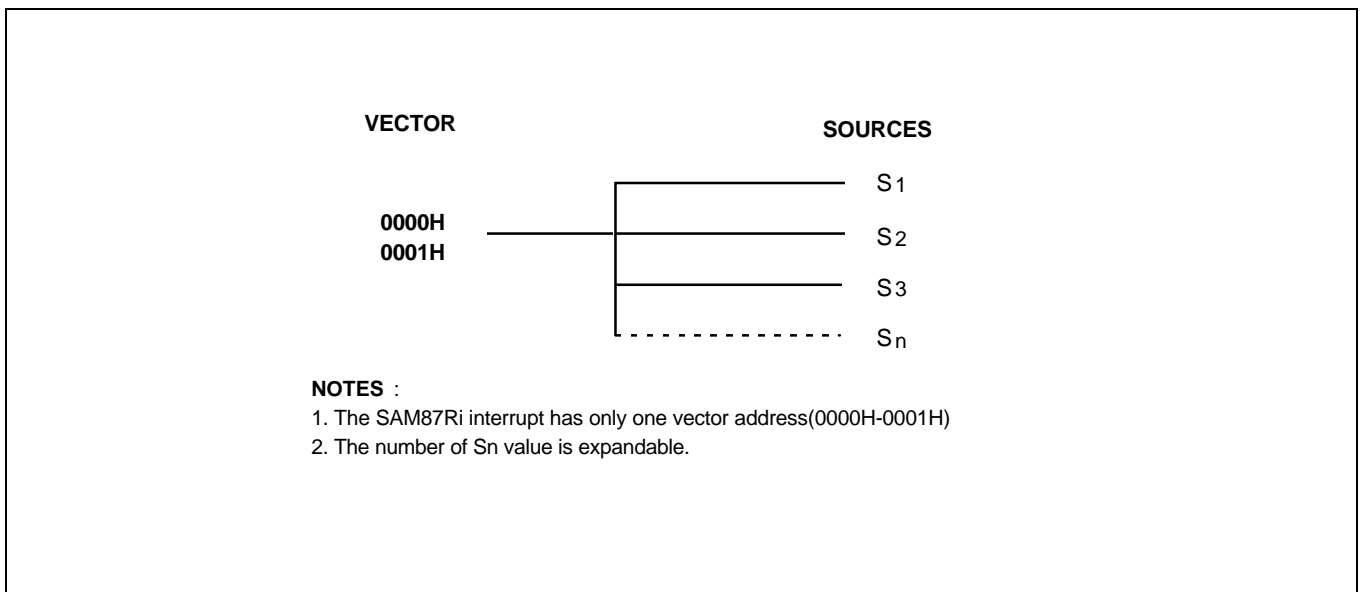


Figure 5-1. KS86-Series Interrupt Type

INTERRUPT PROCESSING CONTROL POINTS

Interrupt processing can 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)
- Interrupt source enable and disable settings in the corresponding peripheral control register(s)

ENABLE/DISABLE INTERRUPT INSTRUCTIONS (EI, DI)

The system mode register, SYM (DFH), is used to enable and disable interrupt processing.

SYM.2 is the enable and disable bit for global interrupt processing respectively, by modifying SYM.2. 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.2 directly to enable and disable interrupts during normal operation, we recommend that you use the EI and DI instructions for this purpose.

INTERRUPT PENDING FUNCTION TYPES

When the interrupt service routine has executed, the application program's service routine must clear the appropriate pending bit before the return from interrupt subroutine (IRET) occurs.

INTERRUPT PRIORITY

Because there is not a interrupt priority register in SAM87Ri, the order of service is determined by a sequence of source which is executed in interrupt service routine.

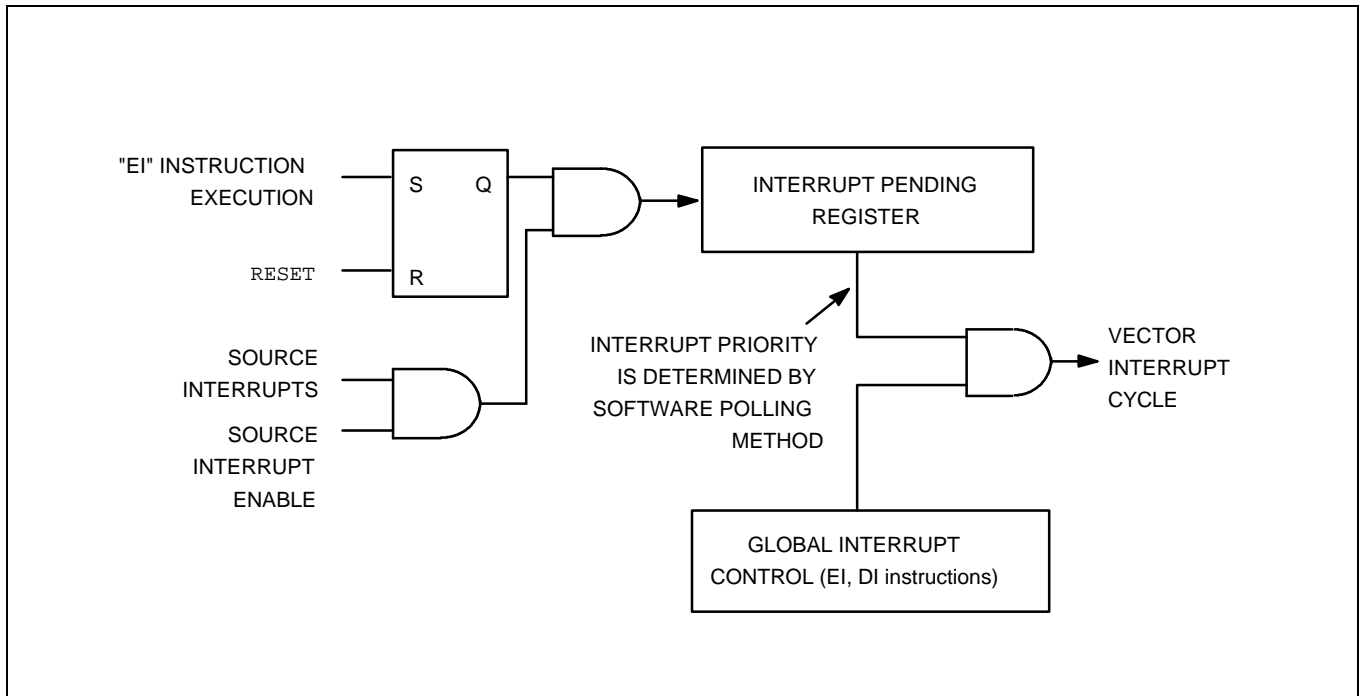


Figure 5-2. Interrupt Function Diagram

INTERRUPT SOURCE SERVICE SEQUENCE

The interrupt request polling and servicing sequence is as follows:

1. A source generates an interrupt request by setting the interrupt request pending bit to "1".
2. The CPU generates an interrupt acknowledge signal.
3. The service routine starts and the source's pending flag is cleared to "0" by software.
4. Interrupt priority must be determined by software polling method.

INTERRUPT SERVICE ROUTINES

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

- Interrupt processing must be enabled (EI, SYM.2 = "1")
- 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 global interrupt enable bit in the SYM register (DI, SYM.2 = "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.2 to "1"(EI), allowing the CPU to process the next interrupt request.

GENERATING INTERRUPT VECTOR ADDRESSES

The interrupt vector area in the ROM contains the address of the interrupt service routine. 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 0000H.
5. Fetch the service routine's low-byte address from the vector address 0001H.
6. Branch to the service routine specified by the 16-bit vector address.

KS86C4004/C4104 INTERRUPT STRUCTURE

The KS86C4004/C4104 microcontroller has six peripheral interrupt sources:

- Timer 0 match/capture interrupt
- Timer 0 overflow interrupt
- Timer 1 match interrupt
- Zero-cross detection
- Two external interrupts for port 2, P2.0–P2.1 (P2.1 not used for KS86C4104/P4104)

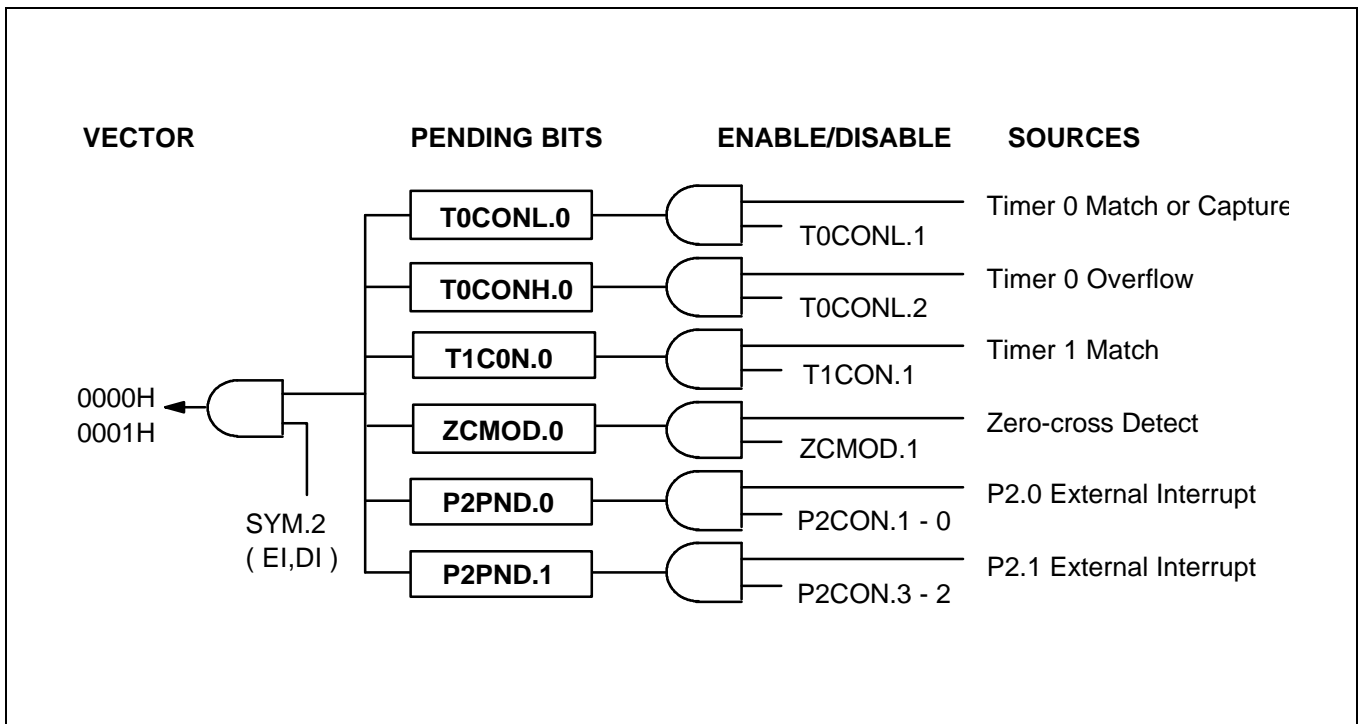


Figure 5-3. KS86C4004/C4104 Interrupt Structure

6

SAM87RI INSTRUCTION SET

OVERVIEW

The SAM87Ri instruction set is designed to support the large register file. It includes a full complement of 8-bit arithmetic and logic operations. There are 41 instructions. No special I/O instructions are necessary because I/O control and data registers are mapped directly into the register file. Flexible instructions for bit addressing, rotate, and shift operations complete the powerful data manipulation capabilities of the SAM87Ri instruction set.

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 13-bit program memory or data memory addresses. For detailed information about register addressing, please refer to Section 2, "Address Spaces".

ADDRESSING MODES

There are six addressing modes: Register (R), Indirect Register (IR), Indexed (X), Direct (DA), Relative (RA), and Immediate (IM). 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
LDC	dst,src	Load program memory
LDE	dst,src	Load external data memory
LDCD	dst,src	Load program memory and decrement
LDED	dst,src	Load external data memory and decrement
LDCI	dst,src	Load program memory and increment
LDEI	dst,src	Load external data memory and increment
POP	dst	Pop from stack
PUSH	src	Push to stack
Arithmetic Instructions		
ADC	dst,src	Add with carry
ADD	dst,src	Add
CP	dst,src	Compare
DEC	dst	Decrement
INC	dst	Increment
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		
CALL	dst	Call procedure
IRET		Interrupt return
JP	cc, dst	Jump on condition code
JP	dst	Jump unconditional
JR	cc, dst	Jump relative on condition code
RET		Return
Bit Manipulation Instructions		
TCM	dst, src	Test complement under mask
TM	dst, src	Test under mask
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
CPU Control Instructions		
CCF		Complement carry flag
DI		Disable interrupts
EI		Enable interrupts
IDLE		Enter Idle mode
NOP		No operation
RCF		Reset carry flag
SCF		Set carry flag
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;

FLAGS register can be set or reset by instructions as long as its outcome does not affect the flags, such as, Load instruction. Logical and Arithmetic instructions such as, AND, OR, XOR, ADD, and SUB can affect the Flags register. For example, the AND instruction updates the Zero, Sign and Overflow flags based on the outcome of the AND instruction. If the AND instruction uses the Flags register as the destination, then simultaneously, two write will occur to the Flags register producing an unpredictable result.

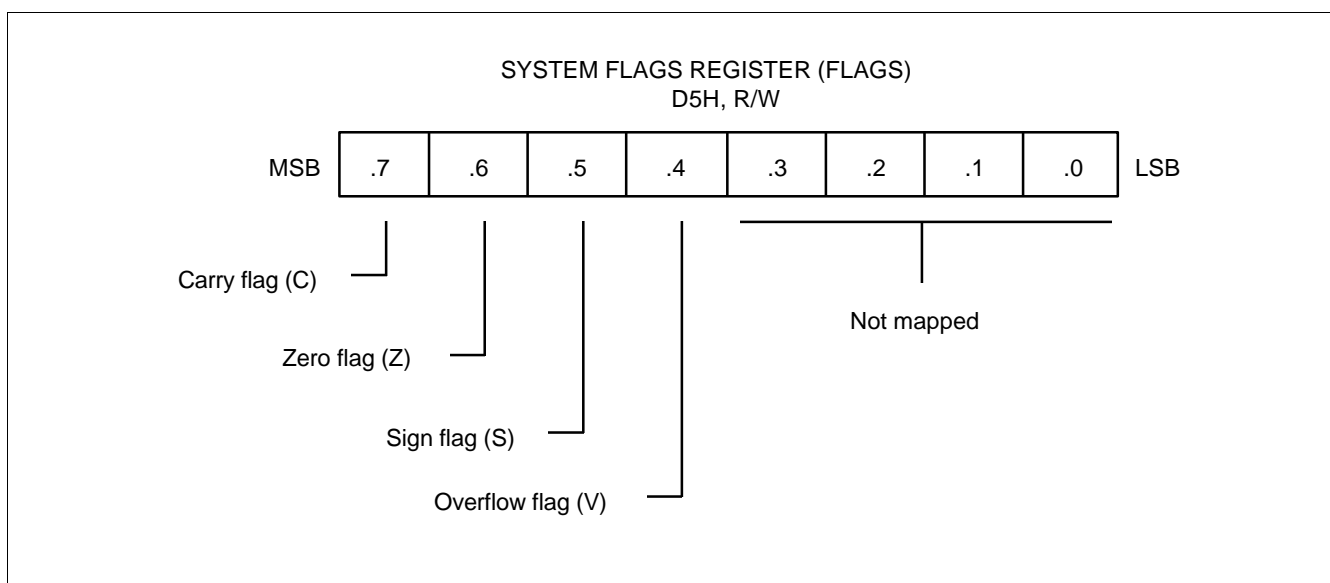


Figure 6-1. System Flags Register (FLAGS)

FLAG DESCRIPTIONS

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
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
FLAGS	Flags register (D5H)
#	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)
rr	Working register pair	RRp (p = 0, 2, 4, ..., 14)
R	Register or working register	reg or Rn (reg = 0–255, n = 0–15)
RR	Register pair or working register pair	reg or RRp (reg = 0–254, even number only, where p = 0, 2, ..., 14)
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 0–8191, where p = 0, 2, ..., 14)
DA	Direct addressing mode	addr (addr = range 0–8191)
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)

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,lr2	ADD R2,R1	ADD IR2,R1	ADD R1,IM	
	1	RLC R1	RLC IR1	ADC r1,r2	ADC r1,lr2	ADC R2,R1	ADC IR2,R1	ADC R1,IM	
P	2	INC R1	INC IR1	SUB r1,r2	SUB r1,lr2	SUB R2,R1	SUB IR2,R1	SUB R1,IM	
E	3	JP IRR1		SBC r1,r2	SBC r1,lr2	SBC R2,R1	SBC IR2,R1	SBC R1,IM	
	4			OR r1,r2	OR r1,lr2	OR R2,R1	OR IR2,R1	OR R1,IM	
R	5	POP R1	POP IR1	AND r1,r2	AND r1,lr2	AND R2,R1	AND IR2,R1	AND R1,IM	
	6	COM R1	COM IR1	TCM r1,r2	TCM r1,lr2	TCM R2,R1	TCM IR2,R1	TCM R1,IM	
N	7	PUSH R2	PUSH IR2	TM r1,r2	TM r1,lr2	TM R2,R1	TM IR2,R1	TM R1,IM	
	8								LD r1, x, r2
B	9	RL R1	RL IR1						LD r2, x, r1
L	A			CP r1,r2	CP r1,lr2	CP R2,R1	CP IR2,R1	CP R1,IM	LDC r1, lrr2, xL
	B	CLR R1	CLR IR1	XOR r1,r2	XOR r1,lr2	XOR R2,R1	XOR IR2,R1	XOR R1,IM	LDC r2, lrr2, xL
E	C	RRC R1	RRC IR1		LDC r1,lrr2				LD r1, lr2
	D	SRA R1	SRA IR1		LDC r2,lrr1			LD IR1,IM	LD lr1, r2
H	E	RR R1	RR IR1	LDCD r1,lrr2	LDCI r1,lrr2	LD R2,R1	LD R2,IR1	LD R1,IM	LDC r1, lrr2, xs
	F					CALL IRR1	LD IR2,R1	CALL DA1	LDC r2, lrr1, xs
X									

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		JR cc,RA	LD r1,IM	JP cc,DA	INC r1	
	P	1	↓	↓	↓	↓	↓	↓	
P	2								
E	3								
R	4								
	5								
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		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 SAM87Ri 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: $dst \leftarrow dst + src + c$

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 <u>dst</u> <u>src</u>
opc	dst src	2	6	12	r r
				13	r lr
opc	src	3	10	14	R R
				15	R IR
opc	dst	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: $dst \leftarrow dst + src$

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 <u>dst</u>	Mode <u>src</u>			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst src</td> </tr> </table>	opc	dst src		2	6	02	r	r	
	opc	dst src							
				03	r	lr			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">src</td> <td style="padding: 2px 10px;">dst</td> </tr> </table>	opc	src	dst		3	10	04	R	R
	opc	src	dst						
				05	R	IR			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst</td> <td style="padding: 2px 10px;">src</td> </tr> </table>	opc	dst	src		3	10	06	R	IM
opc	dst	src							

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 <u>dst</u>	Mode <u>src</u>
opc	dst src	2	6	52	r	r
				53	r	lr
opc	src	3	10	54	R	R
				55	R	IR
opc	dst	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.

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 <u>dst</u>
opc	dst	3	18	F6	DA
opc	dst	2	18	F4	IRR

Examples: Given: R0 = 15H, R1 = 21H, PC = 1A47H, and SP = 0B2H:

CALL 1521H → SP = 0B0H
 (Memory locations 00H = 1AH, 01H = 4AH, where 4AH
 is the address that follows the instruction.)

CALL @RR0 → SP = 0B0H (00H = 1AH, 01H = 49H)

In the first example, if the program counter value is 1A47H and the stack pointer contains the value 0B2H, the statement "CALL 1521H" pushes the current PC value onto the top of the stack. The stack pointer now points to memory location 00H. The PC is then loaded with the value 1521H, 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 01H (because the two-byte instruction format was used). The PC is then loaded with the value 1521H, the address of the first instruction in the program sequence to be executed.

CCF —Complement Carry Flag

CCF

Operation: $C \leftarrow \text{NOT } C$

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 <u>dst</u>
opc	dst	2	6	B0 B1	R 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 <u>dst</u>
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, 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:** Unaffected.
- H:** Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr dst	Mode src			
<table border="1" style="display: inline-table;"> <tr> <td style="width: 40px; text-align: center;">opc</td> <td style="width: 40px; text-align: center;">dst src</td> </tr> </table>			opc	dst src	2	6	A2	r	r	
opc	dst src									
					A3	r	lr			
<table border="1" style="display: inline-table;"> <tr> <td style="width: 40px; text-align: center;">opc</td> <td style="width: 40px; text-align: center;">src</td> <td style="width: 40px; text-align: center;">dst</td> </tr> </table>			opc	src	dst	3	10	A4	R	R
opc	src	dst								
					A5	R	IR			
<table border="1" style="display: inline-table;"> <tr> <td style="width: 40px; text-align: center;">opc</td> <td style="width: 40px; text-align: center;">dst</td> <td style="width: 40px; text-align: center;">src</td> </tr> </table>			opc	dst	src	3	10	A6	R	IM
opc	dst	src								

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.

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, that is, dst value is –128(80H) and result value is +127(7FH); cleared otherwise.
- D:** Unaffected.
- H:** Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
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.

DI –Disable Interrupts

DI

Operation: SYM (2) ← 0

Bit zero of the system mode register, SYM.2, 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)	
<table border="1"><tr><td>opc</td></tr></table>	opc	1	6	8F
opc				

Example: Given: SYM = 04H:

DI

If the value of the SYM register is 04H, the statement "DI" leaves the new value 00H in the register and clears SYM.2 to "0", disabling interrupt processing.

EI —Enable Interrupts

EI

Operation: SYM (2) ← 1

An EI instruction sets bit 2 of the system mode register, SYM.2 to "1". This allows interrupts to be serviced as they occur. 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 04H, enabling all interrupts. (SYM.2 is the enable bit for global interrupt processing.)

IDLE —Idle Operation

IDLE

Operation:

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

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
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, that is dst value is +127(7FH) and result is -128(80H); cleared otherwise.
- D:** Unaffected.
- H:** Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
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.

IRET —Interrupt Return

IRET IREI

Operation: $FLAGS \leftarrow @SP$
 $SP \leftarrow SP + 1$
 $PC \leftarrow @SP$
 $SP \leftarrow SP + 2$
 $SYM(2) \leftarrow 1$

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.

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

JP —Jump

JP cc,dst (Conditional)

JP dst (Unconditional)

Operation: If cc is true, PC ← dst

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

Flags: No flags are affected.

Format: (1)

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst
(2)					
cc opc		3	10/12 (3)	ccD	DA
				cc = 0 to F	
opc		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 op code 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, $PC \leftarrow PC + dst$

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
cc opc	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 op code 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 <u>dst</u>	Mode <u>src</u>
dst opc	src		2	6	rC	r	IM
				6	r8	r	R
src opc	dst		2	6	r9	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

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 'lrr' 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 <u>dst</u>	Mode <u>src</u>
1.	opc dst src	2	12	C3	r	lrr
2.	opc src dst	2	12	D3	lrr	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 —Load Memory

LDC/LDE (Continued)

Examples: Given: R0 = 11H, R1 = 34H, R2 = 01H, R3 = 04H, R4 = 00H, R5 = 60H; Program memory locations 0061 = AAH, 0103H = 4FH, 0104H = 1A, 0105H = 6DH, and 1104H = 88H. External data memory locations 0061H = BBH, 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[RR4]	; R0 ← contents of program memory location 0061H ; (01H + RR4), ; R0 = AAH, R2 = 00H, R3 = 60H
LDE	R0,#01H[RR4]	; R0 ← contents of external data memory location 0061H ; (01H + RR4), R0 = BBH, R4 = 00H, R5 = 60H
LDC *	#01H[RR4],R0	; 11H (contents of R0) is loaded into program memory location ; 0061H (01H + 0060H)
LDE	#01H[RR4],R0	; 11H (contents of R0) is loaded into external data memory ; location 0061H (01H + 0060H)
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 'lrr' 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 lrr

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 'lrr' even for program memory and odd for data memory.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u> <u>src</u>
opc	dst src	2	16	E3	r lrr

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
```

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)	
<table border="1"><tr><td>opc</td></tr></table>	opc	1	6	FF
opc				

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 <u>dst</u>	Mode <u>src</u>			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst src</td> </tr> </table>	opc	dst src		2	6	42	r	r	
	opc	dst src							
			6	43	r	lr			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">src</td> <td style="padding: 2px 10px;">dst</td> </tr> </table>	opc	src	dst		3	10	44	R	R
	opc	src	dst						
			10	45	R	IR			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst</td> <td style="padding: 2px 10px;">src</td> </tr> </table>	opc	dst	src		3	10	46	R	IM
opc	dst	src							

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, SP (0D9H) = 0BBH, and stack register 0BBH = 55H:

POP 00H → Register 00H = 55H, SP = 0BCH

POP @00H → Register 00H = 01H, register 01H = 55H, SP = 0BCH

In the first example, general register 00H contains the value 01H. The statement "POP 00H" loads the contents of location 0BBH (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 0BCH.

RCF —Reset Carry Flag

RCF RCF

Operation: $C \leftarrow 0$

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)	
<table border="1"><tr><td>opc</td></tr></table>	opc	1	6	CF
opc				

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 = 0BCH, (SP) = 101AH, and PC = 1234:

RET → PC = 101AH, SP = 0BEH

The statement "RET" pops the contents of stack pointer location 0BCH (10H) into the high byte of the program counter. The stack pointer then pops the value in location 0BDH (1AH) into the PC's low byte and the instruction at location 101AH is executed. The stack pointer now points to memory location 0BEH.

RL —Rotate Left

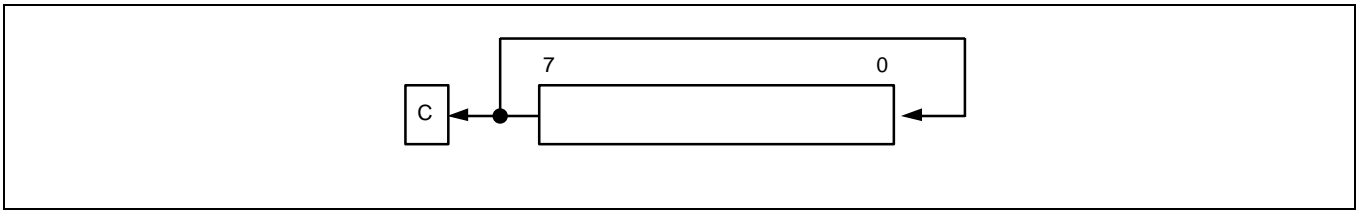
RL dst

Operation: $C \leftarrow \text{dst}(7)$

$\text{dst}(0) \leftarrow \text{dst}(7)$

$\text{dst}(n + 1) \leftarrow \text{dst}(n), n = 0-6$

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, 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	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

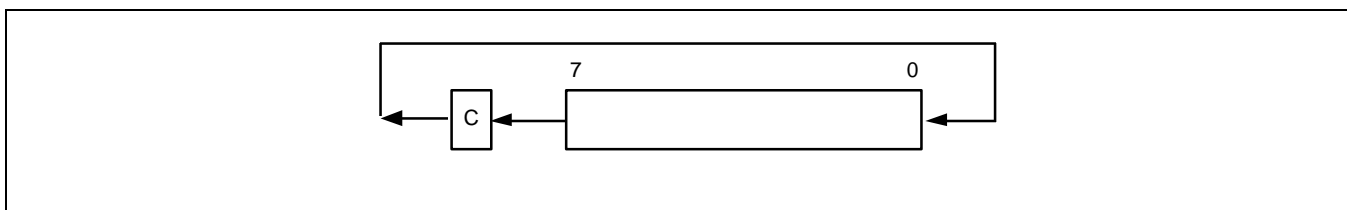
RLC dst

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

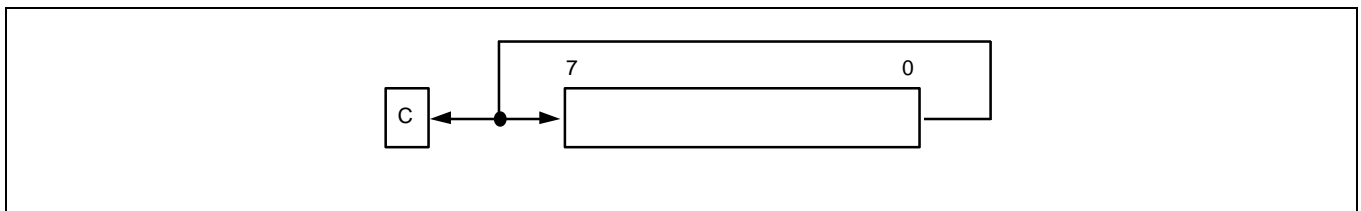
RR dst

Operation: $C \leftarrow \text{dst}(0)$

$\text{dst}(7) \leftarrow \text{dst}(0)$

$\text{dst}(n) \leftarrow \text{dst}(n + 1), n = 0-6$

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
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

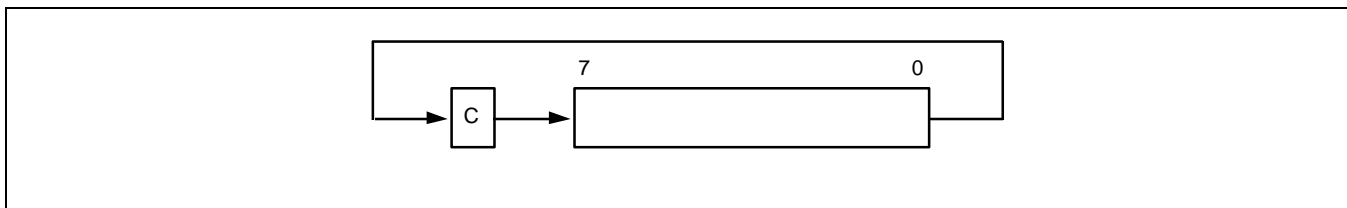
RRC dst

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".

SBC –Subtract With Carry

SBC dst,src

Operation: $dst \leftarrow dst - src - c$

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 <u>dst</u>	Mode <u>src</u>			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst src</td> </tr> </table>	opc	dst src		2	6	32	r	r	
	opc	dst src							
			6	33	r	lr			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">src</td> <td style="padding: 2px 10px;">dst</td> </tr> </table>	opc	src	dst		3	10	34	R	R
	opc	src	dst						
			10	35	R	IR			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst</td> <td style="padding: 2px 10px;">src</td> </tr> </table>	opc	dst	src		3	10	36	R	IM
opc	dst	src							

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

SBC	R1,R2	→	R1 = 0CH, R2 = 03H
SBC	R1,@R2	→	R1 = 05H, R2 = 03H, register 03H = 0AH
SBC	01H,02H	→	Register 01H = 1CH, register 02H = 03H
SBC	01H,@02H	→	Register 01H = 15H, register 02H = 03H, register 03H = 0AH
SBC	01H,#8AH	→	Register 01H = 95H; C, S, and V = "1"

In the first example, if working register R1 contains the value 10H and register R2 the value 03H, the statement "SBC R1,R2" subtracts the source value (03H) and the C flag value ("1") from the destination (10H) and then stores the result (0CH) in register R1.

SCF —Set Carry Flag

SCF

Operation: $C \leftarrow 1$

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)	
<table border="1"><tr><td>opc</td></tr></table>	opc	1	6	DF
opc				

Example: The statement

SCF

sets the carry flag to logic one.

SRA –Shift Right Arithmetic

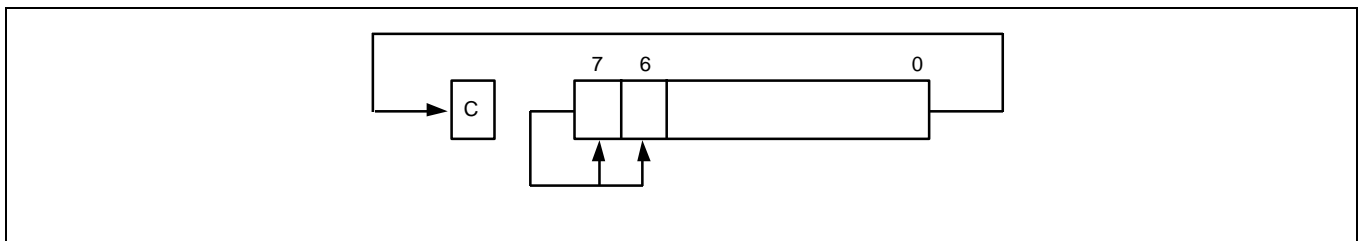
SRA dst

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
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.

STOP —Stop Operation

STOP

Operation:

The STOP instruction stops both the CPU clock and system clock and causes the microcontroller to enter Stop mode. During Stop mode, the contents of on-chip CPU registers, peripheral registers, and I/O port control and data registers are retained. Stop mode can be released by an external reset operation or External interrupt input. 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 <u>dst</u>	Mode <u>src</u>
opc	1	3	7F	–	–

Example: The statement

```
STOP
```

halts all microcontroller operations.

SUB –Subtract

SUB dst,src

Operation: $dst \leftarrow dst - src$

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 <u>dst</u>	Mode <u>src</u>			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst src</td> </tr> </table>	opc	dst src		2	6	22	r	r	
	opc	dst src							
			6	23	r	lr			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">src</td> <td style="padding: 2px 10px;">dst</td> </tr> </table>	opc	src	dst		3	10	24	R	R
	opc	src	dst						
			10	25	R	IR			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst</td> <td style="padding: 2px 10px;">src</td> </tr> </table>	opc	dst	src		3	10	26	R	IM
opc	dst	src							

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.

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 dst	Mode src
opc	dst src	2	6	62	r	r
			6	63	r	lr
opc	src	3	10	64	R	R
			10	65	R	IR
opc	dst	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 dst	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.

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 <u>dst</u>	Mode <u>src</u>			
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst src</td> </tr> </table>	opc	dst src		2	6	B2	r	r	
	opc	dst src							
6	B3	r	lr						
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">src</td> <td style="padding: 2px 10px;">dst</td> </tr> </table>	opc	src	dst		3	10	B4	R	R
	opc	src	dst						
10	B5	R	IR						
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding: 2px 10px;">opc</td> <td style="padding: 2px 10px;">dst</td> <td style="padding: 2px 10px;">src</td> </tr> </table>	opc	dst	src		3	10	B6	R	IM
opc	dst	src							

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.

Clock Circuit

RESET and Power-Down

I/O Ports

Basic Timer and Timers

A/D Converter

Zero-Crossing Detection Circuit

Electrical Data

Mechanical Data

KS86P4004/P4104 OTP

Development Tools

7

CLOCK CIRCUIT

OVERVIEW

An RC oscillation source provides a typical 4-MHz clock for KS86C4004/C4104. An internal capacitor supports the RC oscillator circuit. An external crystal or ceramic oscillation source provides a maximum 10-MHz clock. The X_{IN} and X_{OUT} pins connect the oscillation source to the on-chip clock circuit. Simplified RC oscillator and crystal/ceramic oscillator circuits are shown in Figures 7-1 and 7-2.

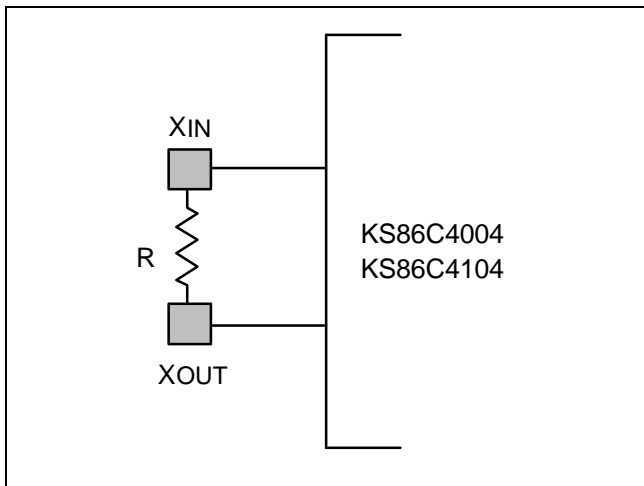


Figure 7-1. Main Oscillator Circuit
(RC Oscillator with Internal Capacitor)

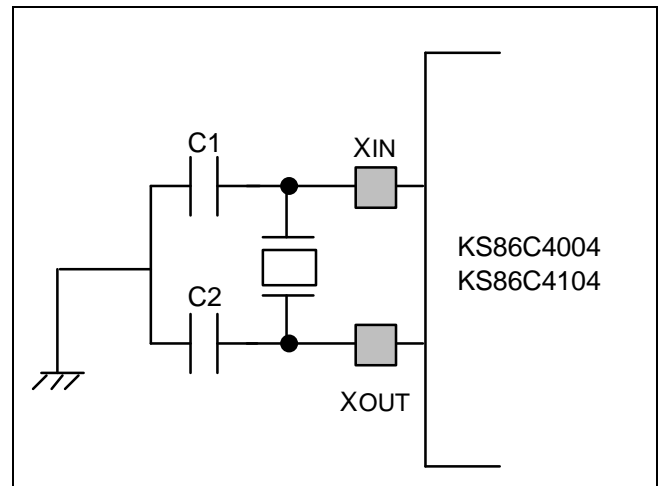


Figure 7-2. Main Oscillator Circuit
(Crystal/Ceramic Oscillator)

MAIN OSCILLATOR LOGIC

To increase processing speed and to reduce clock noise, non-divided logic is implemented for the main oscillator circuit. For this reason, very high resolution waveforms (square signal edges) must be generated in order for the CPU to efficiently process logic operations.

CLOCK STATUS DURING POWER-DOWN MODES

The two power-down modes, Stop mode and Idle mode, affect clock oscillation as follows:

- In Stop mode, the main oscillator "freezes," halting the CPU and peripherals. The contents of the register file and current system register values are retained. Stop mode is released, and the oscillator started, by a reset operation or by an external interrupt with RC-delay noise filter (for KS86C4004/C4104, INT0–INT1).
- In Idle mode, the internal clock signal is gated off to the CPU, but not to interrupt control and the timer. The current CPU status is preserved, including stack pointer, program counter, and flags. Data in the register file is retained. Idle mode is released by a reset or by an interrupt (external or internally-generated).

SYSTEM CLOCK CONTROL REGISTER (CLKCON)

The system clock control register, CLKCON, is located in location D4H. It is read/write addressable and has the following functions:

- Oscillator IRQ wake-up function enable/disable (CLKCON.7)
- Oscillator frequency divide-by value: non-divided, 2, 8, or 16 (CLKCON.4 and CLKCON.3)

The CLKCON register controls whether or not an external interrupt can be used to trigger a Stop mode release (This is called the "IRQ wake-up" function). The IRQ wake-up enable bit is CLKCON.7.

After a reset, the external interrupt oscillator wake-up function is enabled, the main oscillator is activated, and the $f_{OSC}/16$ (the slowest clock speed) is selected as the CPU clock. If necessary, you can then increase the CPU clock speed to f_{OSC} , $f_{OSC}/2$ or $f_{OSC}/8$.

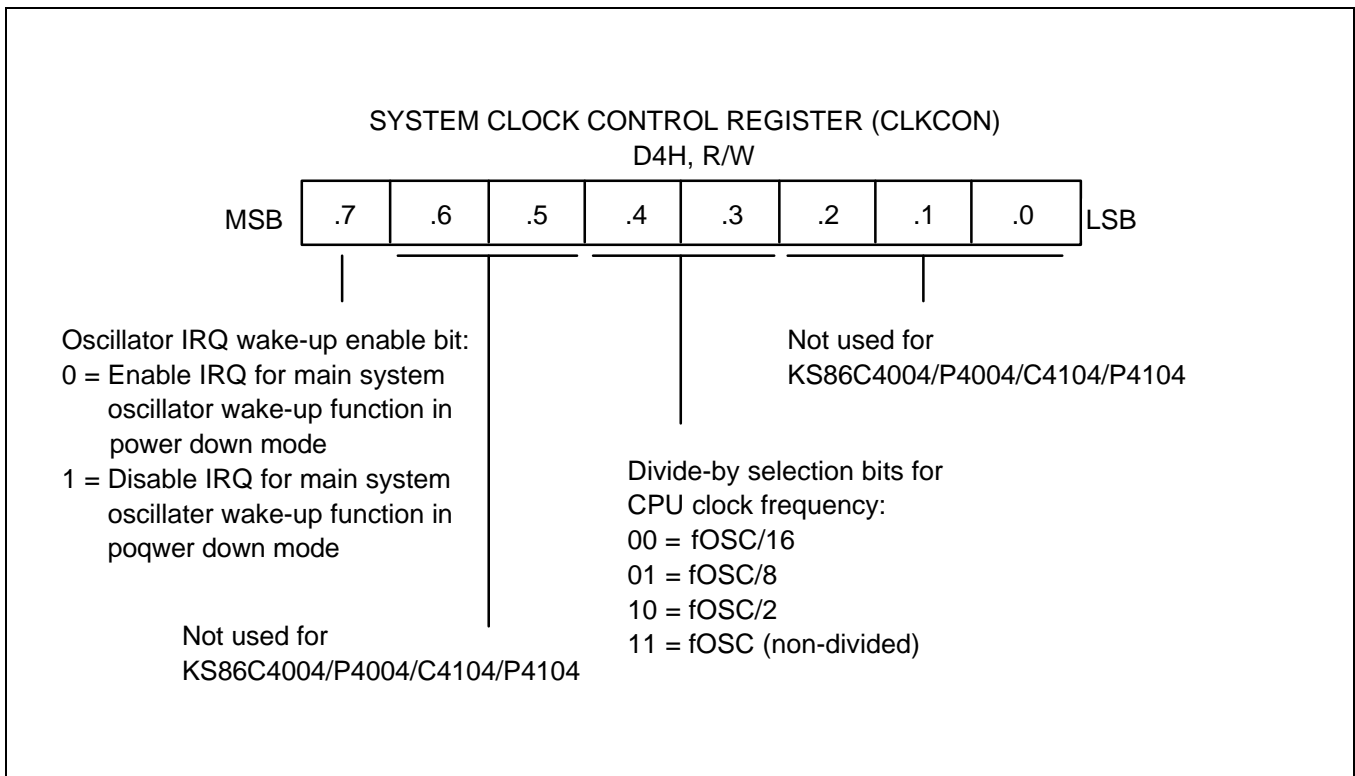


Figure 7-3. System Clock Control Register (CLKCON)

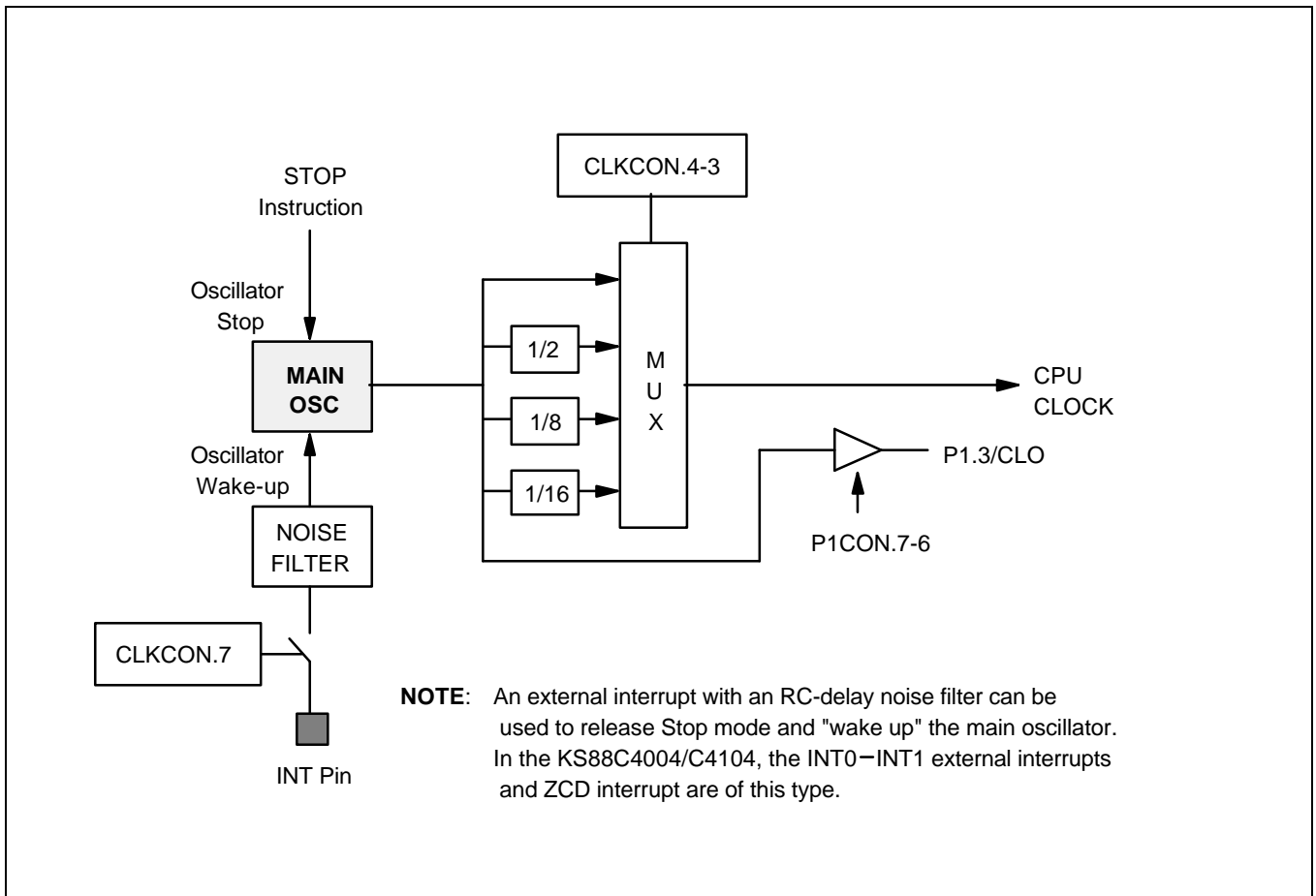


Figure 7-4. System Clock Circuit Diagram

NOTES

8

RESET and POWER-DOWN

SYSTEM RESET

OVERVIEW

During a power-on reset, the voltage at V_{DD} is High level and the RESET pin is forced to Low level. The RESET signal is input through a Schmitt trigger circuit where it is then synchronized with the CPU clock. This brings the KS86C4004/4104 into a known operating status.

The RESET pin must be held to Low level for a minimum time interval after the power supply comes within tolerance in order to allow time for internal CPU clock oscillation to stabilize. The minimum required oscillation stabilization time for a reset is approximately $6.55\text{ms} (\cong 2^{16}/f_{\text{osc}}, f_{\text{osc}} = 10\text{MHz})$.

When a reset occurs during normal operation (with both V_{DD} and RESET at High level), the signal at the RESET pin is forced Low and the reset operation starts. All system and peripheral control registers are then set to their default hardware reset values (see Table 8-1).

The following sequence of events occurs during a reset operation:

- All interrupts are disabled.
- The watchdog function (basic timer) is enabled.
- Ports 0-3 are set to input mode and all pull-up resistors are disabled.
- Peripheral control and data registers are disabled and reset to their initial values.
- The program counter is loaded with the ROM reset address, 0100H.
- When the programmed oscillation stabilization time interval has elapsed, the address stored in ROM location 0100H (and 0101H) is fetched and executed.

NOTE

To program the duration of the oscillation stabilization interval, you must make the appropriate settings to the basic timer control register, BTCON, before entering Stop mode. Also, if you do not want to use the basic timer watchdog function (which causes a system reset if a basic timer counter overflow occurs), you can disable it by writing '1010B' to the upper nibble of BTCON.

POWER-DOWN MODES

STOP MODE

Stop mode is invoked by the instruction STOP (opcode 7FH). In Stop mode, the operation of the CPU and all peripherals is halted. That is, the on-chip main oscillator stops and the supply current is reduced to less than 5 μ A. All system functions are halted when the clock "freezes," but data stored in the internal register file is retained. Stop mode can be released in one of two ways: by a RESET signal or by an external interrupt.

Using RESET to Release Stop Mode

Stop mode is released when the RESET signal is released and returns to High level. All system and peripheral control registers are then reset to their default values and the contents of all data registers are retained. A reset operation automatically selects a slow clock (1/16) because CLKCON.3 and CLKCON.4 are cleared to '00B'. After the oscillation stabilization interval has elapsed, the CPU executes the system initialization routine by fetching the 16-bit address stored in ROM locations 0100H and 0101H.

Using an External Interrupt to Release Stop Mode

Only external interrupts with an RC-delay noise filter circuit can be used to release Stop mode (Clock-related external interrupts cannot be used). External interrupts INT0–INT1 in the KS86C4004/4104 interrupt structure meet this criteria.

Note that when Stop mode is released by an external interrupt, the current values in system and peripheral control registers are not changed. When you use an interrupt to release Stop mode, the CLKCON.3 and CLKCON.4 register values remain unchanged, and the currently selected clock value is used. If you use an external interrupt for Stop mode release, you can also program the duration of the oscillation stabilization interval. To do this, you must make the appropriate control and clock settings *before* entering Stop mode.

The external interrupt is serviced when the Stop mode release occurs. Following the IRET from the service routine, the instruction immediately following the one that initiated Stop mode is executed.

IDLE MODE

Idle mode is invoked by the instruction IDLE (opcode 6FH). In Idle mode, CPU operations are halted while select peripherals remain active. During Idle mode, the internal clock signal is gated off to the CPU, but not to interrupt logic and timer/counters. Port pins retain the mode (input or output) they had at the time Idle mode was entered.

There are two ways to release Idle mode:

1. Execute a reset. All system and peripheral control registers are reset to their default values and the contents of all data registers are retained. The reset automatically selects a slow clock (1/16) because CLKCON.3 and CLKCON.4 are cleared to '00B'. If interrupts are masked, a reset is the only way to release Idle mode.
2. Activate any enabled interrupt, causing Idle mode to be released. When you use an interrupt to release Idle mode, the CLKCON.3 and CLKCON.4 register values remain unchanged, and the currently selected clock value is used. The interrupt is then serviced. Following the IRET from the service routine, the instruction immediately following the one that initiated Idle mode is executed.

NOTES

1. Only external interrupts that are not clock-related can be used to release stop mode. To release Idle mode, however, any type of interrupt (that is, internal or external) can be used.
2. Before enter the STOP or IDLE mode, the ZCD (P1CON) and ADC (P2CON, P3CONH, P3CONL) must be disabled. Otherwise, the STOP or IDLE current will be increased significantly.

HARDWARE RESET VALUES

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

- A "1" or a "0" shows the reset bit value as logic one or logic zero, respectively.
- An 'x' means that the bit value is undefined following a reset.
- A dash ('-') means that the bit is either not used or not mapped.

Table 8–1. Register Values After a Reset

Register Name	Mnemonic	Address	Bit Values After RESET								
			7	6	5	4	3	2	1	0	
General purpose register (page 0)	–	00H–BFH	x	x	x	x	x	x	x	x	x
Working registers	R0–R15	C0H–CFH	x	x	x	x	x	x	x	x	x
Timer 0 counter register	T0CNT	D0H	0	0	0	0	0	0	0	0	0
Timer 0 data register	T0DATA	D1H	1	1	1	1	1	1	1	1	1
Timer 0 control register (high)	T0CONH	D2H	–	–	–	–	–	–	–	–	0
Timer 0 control register (low)	T0CONL	D3H	0	0	0	0	0	0	0	0	0
Clock control register	CLKCON	D4H	0	–	–	0	0	–	–	–	–
System flags register	FLAGS	D5H	x	x	x	x	–	–	–	–	–
Locations D6H–D8H are not mapped.											
Stack pointer register	SP	D9H	x	x	x	x	x	x	x	x	x
Location DAH is not mapped.											
Location DBH is reserved.											
Basic timer control register	BTCON	DCH	0	0	0	0	0	0	0	0	0
Basic timer counter	BTCNT	DDH	0	0	0	0	0	0	0	0	0
Location DEH is reserved.											
System mode register	SYM	DFH	–	–	–	–	–	–	0	0	0

Table 8-1. Register Values After a Reset (continued)

Bank 0 Register Name	Mnemonic	Address	Bit Values After a Reset							
			7	6	5	4	3	2	1	0
Port 0 data register	P0	E0H	0	0	0	0	0	0	0	0
Port 1 data register	P1	E1H	–	–	–	–	0	0	0	0
Port 2 data register	P2	E2H	–	–	–	–	0	0	0	0
Port 3 data register	P3	E3H	–	–	0	0	0	0	0	0
Locations E4H–E5H are not mapped.										
Port 0 control register	P0CON	E6H	0	0	0	0	0	0	0	0
Port 0 pull-up resistor enable register	P0PUR	E7H	0	0	0	0	0	0	0	0
Port 0 N-channel open-drain mode	P0PNE	E8H	0	0	0	0	0	0	0	0
Port 1 control register	P1CON	E9H	0	0	0	0	0	0	0	0
Port 2 control register	P2CON	EAH	0	0	0	0	0	0	0	0
Port 2 open-drain, pull-up resistor enable	P2DPUR	EBH	0	0	0	0	0	0	0	0
Port 2 interrupt pending register	P2PND	ECH	–	–	–	–	–	–	0	0
Location EDH is not mapped.										
Port 3 control register (high byte)	P3CONH	EEH	–	–	–	–	0	0	0	0
Port 3 control register (low byte)	P3CONL	EFH	0	0	0	0	0	0	0	0
Locations F0H – F1H are not mapped.										
Timer 1 counter register	T1CNT	F2H	0	0	0	0	0	0	0	0
Timer 1 control register	T1CON	F3H	–	–	0	0	0	0	0	0
Timer 1 data register	T1DATA	F4H	1	1	1	1	1	1	1	1
Zero-crossing detector control register	ZCMOD	F5H	–	–	–	–	0	0	0	0
8-bit prescaler for buzzer output	BUZPS	F6H	0	0	0	0	0	0	0	0
A/D control register	ADCON	F7H	–	0	0	0	0	–	–	0
A/D converter data register	ADDATAH	F8H	x	x	x	x	x	x	x	x
Location F9H is not mapped.										
PWM extension data register	PWMEX	FAH	–	–	–	–	–	–	0	0
PWM extension counter register	T0EXCNT	FBH	–	–	–	–	–	–	0	0
Locations FCH – FFH are not mapped.										

NOTE: “–” means not mapped, “x” means undefined.

PROGRAMMING TIP -- Sample KS86C4004 Initialization Routine

The following sample program suggests how to program the initial program settings for KS86C4004 address space, interrupt, and peripheral function. Program comment guide you through the necessary steps.

;-----<< Interrupt vector address >>

```
.ORG 0000H
.VECTOR 00H, COMMON_INT ; IRQ0 / Interrupt vector address
```

;-----<< Initialize system and peripherals >>

```
RESET: .ORG 0100H ; Reset start address
DI ; disable interrupt
LD BTCON,#00000010B ; enable watch-dog function
; clock source:fosc/4096 (104ms overflow at 10MHz)
LD CLKCON,#00011000B ; CPU clock source select (non-divided)
LD SP,#0C0H ; KS86C4004 Stack pointer initial
LD P0CON,#0FFH ; push-pull output (LED direct drive --> Low active)
LD P0PUR,#00H ; disable pull-up resistor
LD P0PNE,#00H ; disable open-drain
LD P1CON,#1010101011B ; P1.0 ZCD input enable / P1.1-3 push-pull
LD ZCMOD,#00000010B ; enable falling edge interrupt
LD T1DATA,#81H
LD T1CON,#00001100B ; ZCD clear enable (fosc/512)
; timer 1 interrupt disable
LD P2CON,#11000110B ; P2.3 A/D input mode / P2.2 input mode
; P2.1 falling edge interrupt
LD P2DPUR,#00010110B ; P2.0 open-drain output mode
; P2.2 pull-up enable
; P2.1 pull-up enable
LD P2PND,#00H ; no interrupt pending
LD P3CONH,#0FH ; AD input mode(AD4,AD5)
LD P3CONL,#0AAH ; P3.0-3 push-pull output mode
.
.
.
```

;-----<< Initialize data register >>

```
RAMCLR: LD R0,#0BFH
CLR @R0 ; (00h~0BFh) <- #00h
DEC R0
JR NZ,RAMCLR ; Initialize data register
.
.
LD T0DATA,#26H ; 1ms interval at 10MHz system clock
LD T0CONH,#00H ; timer 0 overflow interrupt disable
LD T0CONL,#01001010B ; timer 0 match interrupt enable
; clock source : fosc/256
EI ; enable interrupt
```

;-----<< Main loop >>

```

MAIN:      .
           CALL   XXX           ; subroutine call
           CALL   YYY           ; subroutine call
           .
           .
           LD     BTCON,#02H     ; enable watch-dog function
                                   ; basic counter(BTCNT) clear
           JP     T,MAIN        ; for main loop

```

;-----<< Subroutines >>

```

XXX:      .
           .
           RET
YYY:      .
           .
           .
           RET
           .
           .
           .

```

;-----<< Interrupt service routine >>

```

COMMON_INT:  TM     ZCMOD,#00000001B
              JP     NZ,ZCD_INT
              TM     T1CON,#00000001B
              JP     NZ,TIMER1_INT
              TM     T0CONL,#00000001B
              JP     NZ,TIMER0_INT
              TM     P2PND,#00000001B
              JP     NZ,EXT20_INT
              TM     P2PND,#00000010B
              JP     NZ,EXT21_INT
              TM     T0CONH,#00000001B
              JP     T0OVERFLOW_INT

TIMER0_INT:  .
              AND    T0CONL,#11111110B ; timer0 pending bit clear
              .
              .
              IRET ; timer0 interrupt return

TIMER1_INT:  .
              AND    T1CON,#11111100B ; timer1 pending bit clear / timer 1 interrupt disable
              XOR    P1,#00000100B   ; P1.2 toggle
              RET

ZCD_INT:     AND    ZCMOD,#11111110B ; pending bit clear
              XOR    P1,#00001000B   ; P1.3 toggle
              LD     T1CON,#00001110B ; timer1 interrupt enable (fosc/512)
                                   ; enable ZCD clear signal to clear the timer 1 counter
              IRET

```

```
EXT20_INT:      PUSH  R5
                LD    P2PND,#00000010B ; P2.0 pending bit clear
                .
                LD    R5,#X1
                POP   R5
                IRET
EXT21_INT:      PUSH  R8
                LD    P2PND,#00000001B ; P2.1 pending bit clear
                .
                AND   R8,#X2
                POP   R8
                IRET
T0OVERFLOW_INT: NOP
                LD    T0CONH,#0 ; overflow pending bit clear
                INC   CAPTURE_BUFH
                IRET
                .
                .
                END
```

NOTES

9 I/O PORTS

OVERVIEW

The KS86C4004/C4104 has four I/O ports (0–3): KS86C4004/P4004, with 22 pins total and KS86C4104/P4104, with 16 pins total. You access these ports directly by writing or reading port data register addresses.

Port 0 can be configured as LED drive. (High current output: typical 15mA)

Table 9-1. KS86C4004/C4104 Port Configuration Overview

Port	Function Description	Programmability
0	Bit-programmable I/O port for normal input or push-pull, open-drain output. Pull-up resistors are assignable by software	Bit
1	Bit-programmable I/O port for Schmitt trigger input or push-pull output. Pull-up resistors are assignable by software. Port1 pins can also be used as alternative function. (ZCD, PWM, buzzer)	Bit
2	Bit-programmable I/O port for Schmitt trigger input or push-pull, open drain output. Pull-up resistors are assignable by software. Port2 can also be used as external interrupt, A/D converter input.	Bit
3	Bit-programmable I/O port for Schmitt trigger input or push-pull output. Pull-up resistors are assignable by software. Port3 pins can also be used as A/D converter input.	Bit

PORT DATA REGISTERS

Table 9-2 gives you an overview of the port data register names, locations, and addressing characteristics. Data registers for ports 0-3 have the structure shown in Figure 9-1.

Table 9-2. Port Data Register Summary

Register Name	Mnemonic	Hex	R/W
Port 0 data register	P0	E0H	R/W
Port 1 data register	P1	E1H	R/W
Port 2 data register	P2	E2H	R/W
Port 3 data register	P3	E3H	R/W

NOTE: A reset operation clears the P0-P3 data register to '00H'.

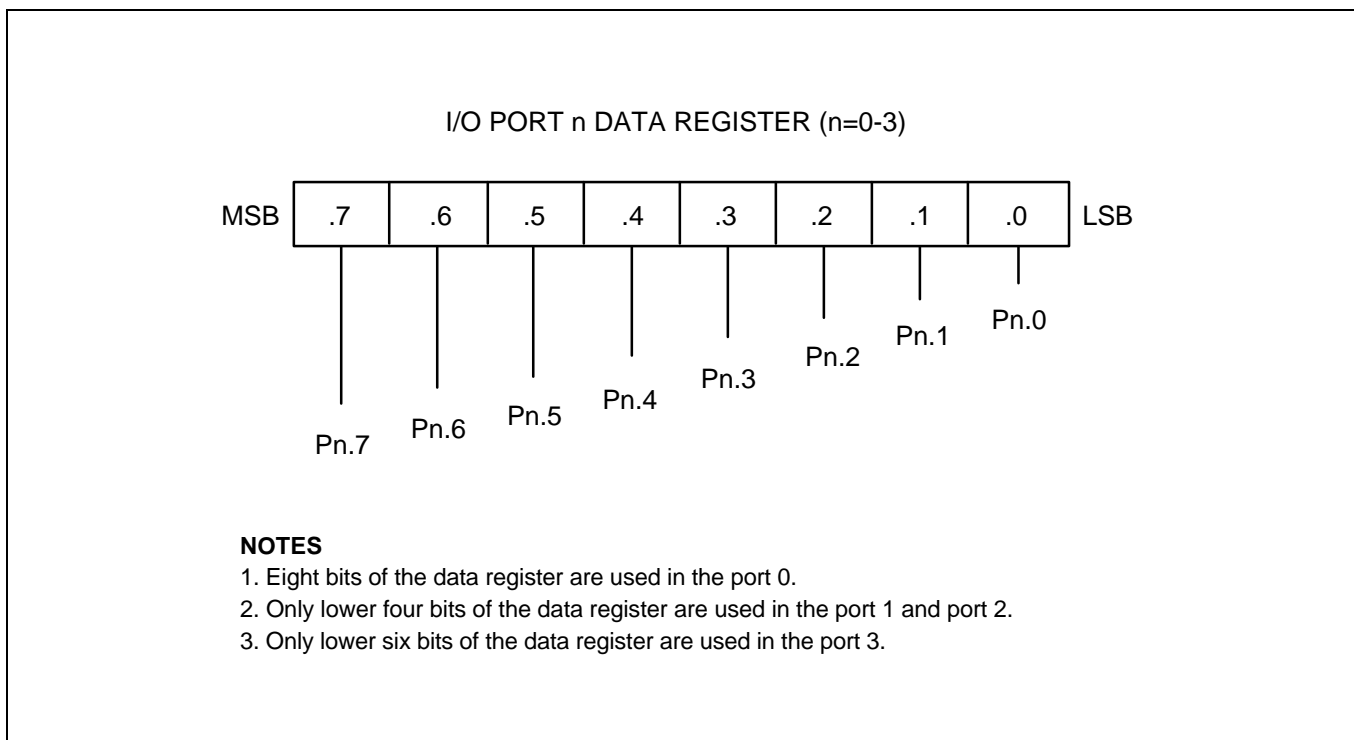


Figure 9-1. Port Data Register Format

PORT 0

Port 0 is a bit-programmable, general-purpose, I/O ports. You can select normal input or push-pull, open drain output mode. In addition, you can configure a pull-up resistor to individual pins using control register settings. It is designed for high-current functions such as LED direct drive.

You access port 0 directly by writing or reading the corresponding port data register, P0 (E0H). A reset clears the port control register, P0CON, to '00H' configuring port 0 pins as normal inputs.

Two addition registers are used to control Port 0: P0PUR(E7H) and P0PNE(E8H). By setting bits in the Port 0 open-drain enable register P0PNE, you can configure specific pin as a open-drain output.

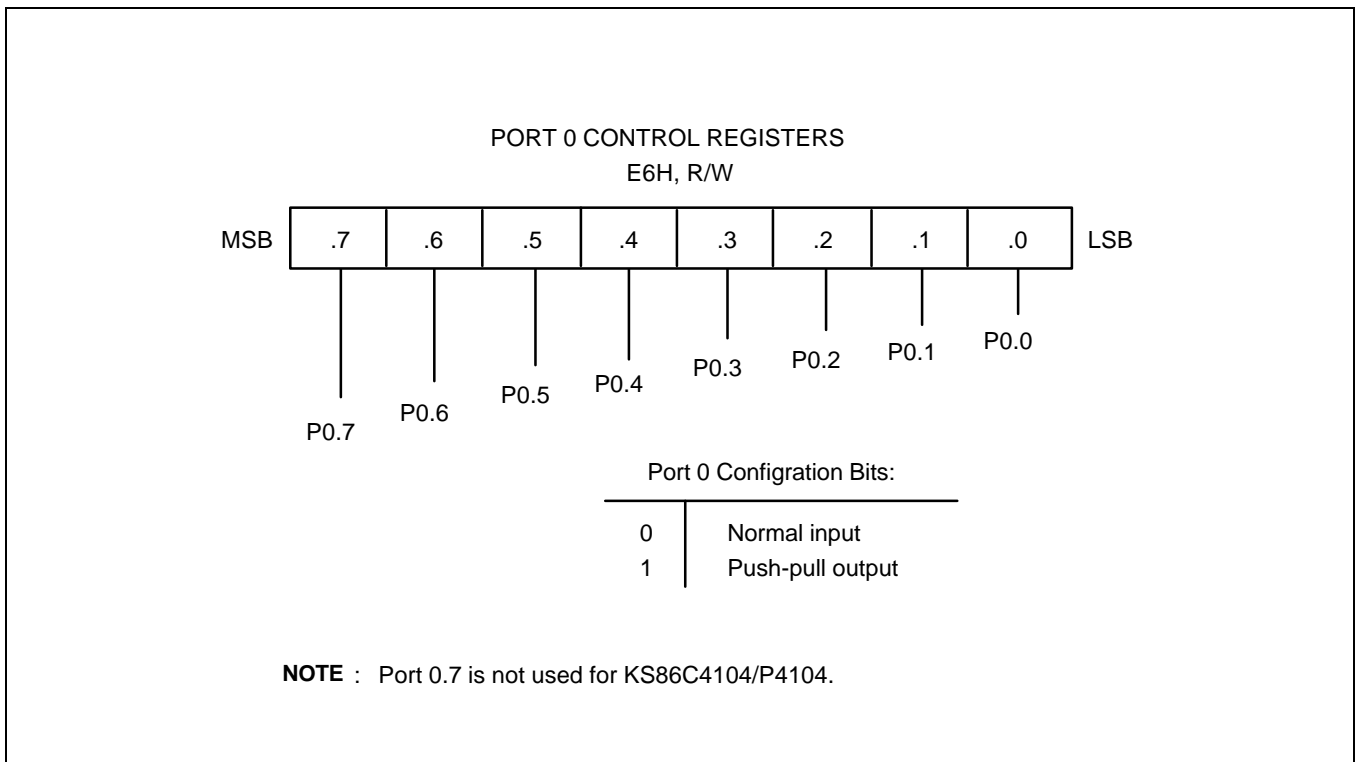


Figure 9-2. Port 0 Control Registers (P0CON)

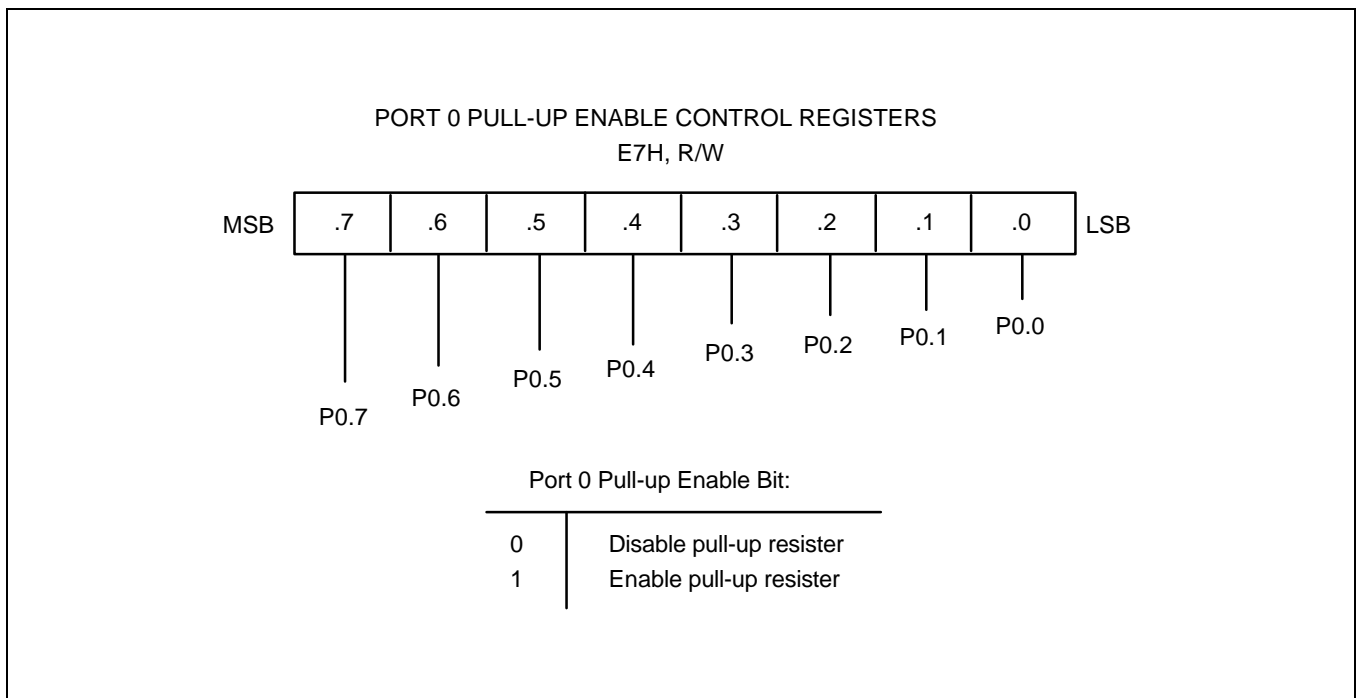


Figure 9-3. Port 0 Pull-up Enable Control Registers (P0PUR)

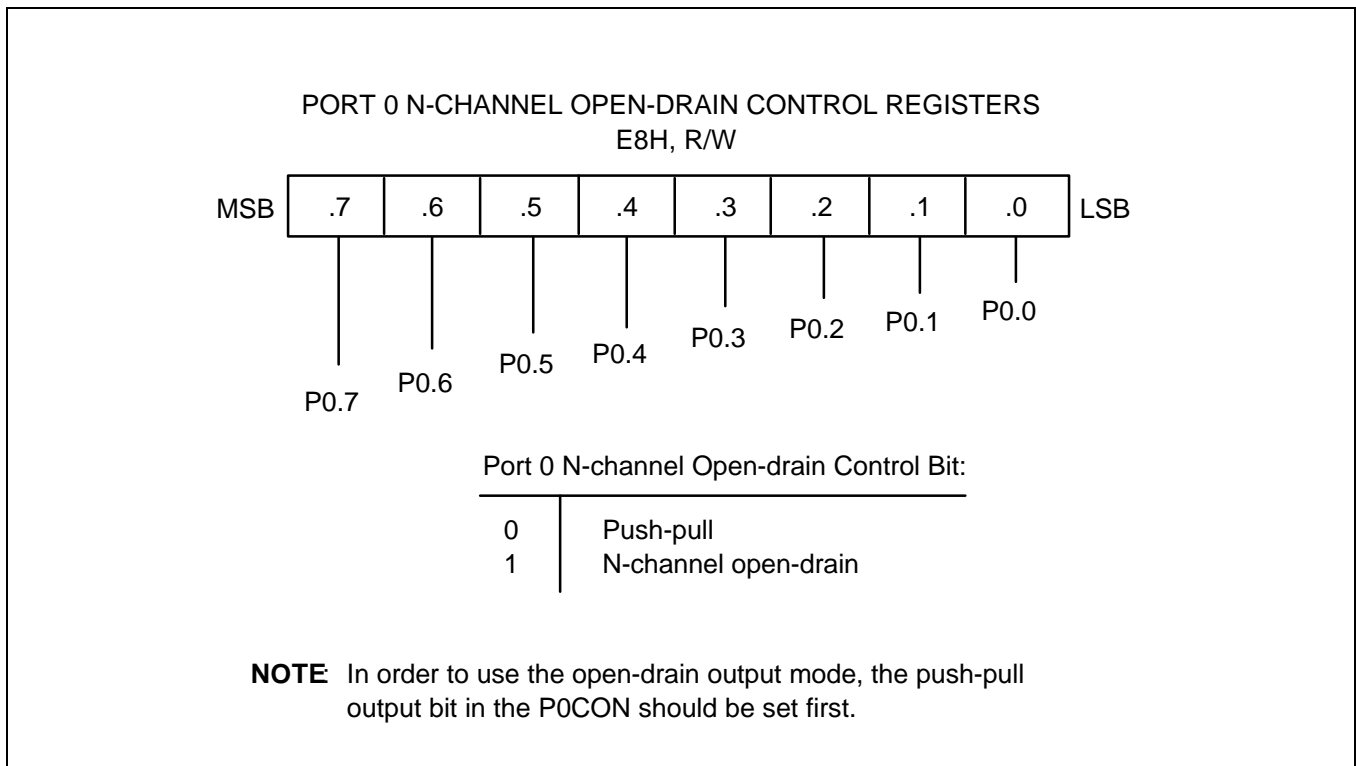


Figure 9-4. Port 0 Control Registers (P0PNE)

PORT 1

Port 1, is a 4-bit I/O port with individually configurable pins. It can be used for general I/O port (Schmitt trigger input mode or push-pull output mode). You can also use port1 as special input (ZCD) or output (BUZ, T0, CLO). In addition, you can configure a pull-up resistor to individual pin using control register settings.

In normal operating mode, a reset clears P1CON to '00H', configuring P1.0–P1.3 as normal Schmitt trigger inputs, but you can also configure P1CON to '0FFH' for alternative functions.

You address port 1 bits directly by writing or reading the port 1 data register, P1 (E1H). The port 1 control register, P1CON is located at addresses E9H.

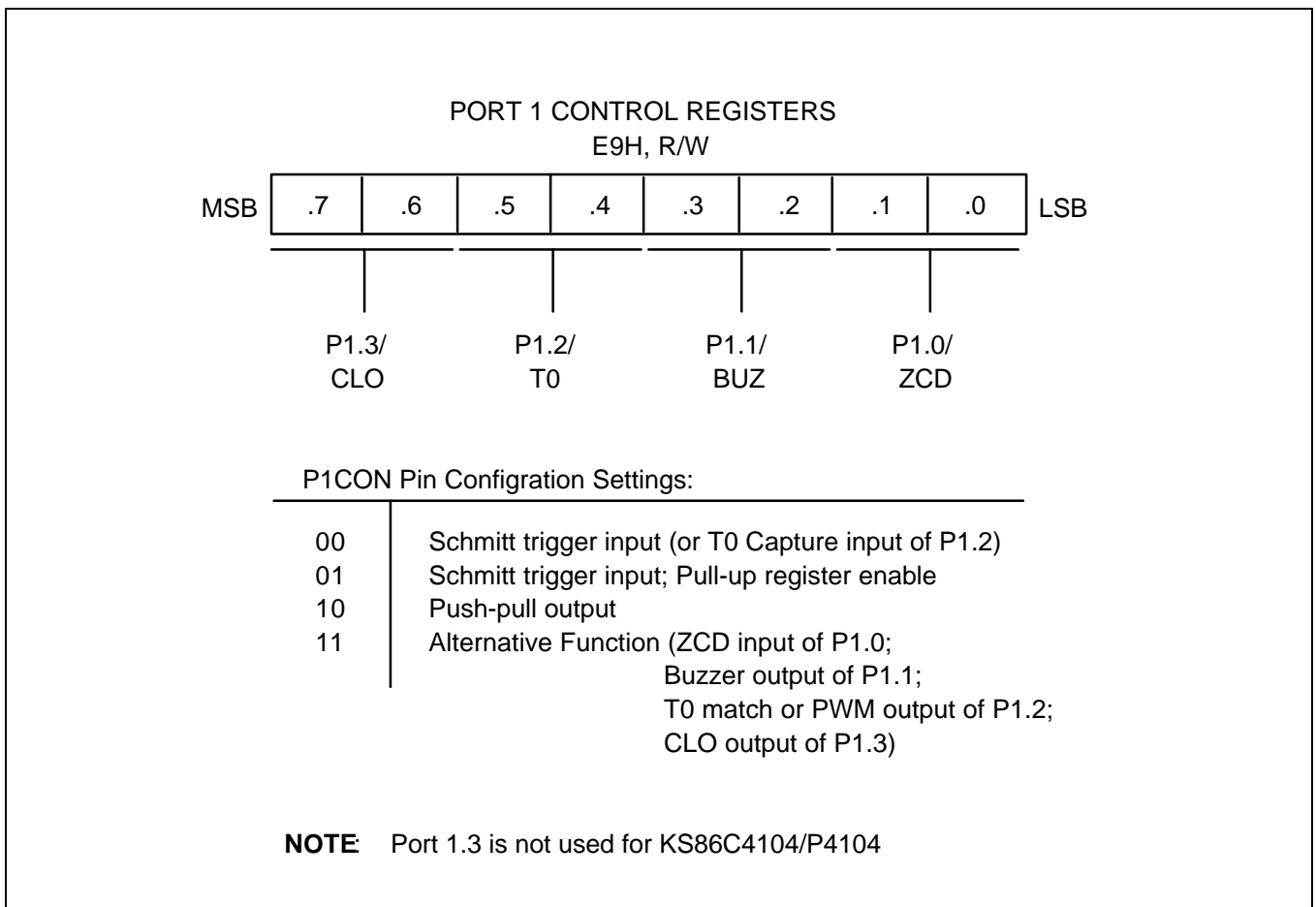


Figure 9-5. Port 1 Control Registers (P1CON)

PORT 2

Port 2 is a 4-bit I/O port with individually configurable pins. It can be used for general I/O port (Schmitt trigger input mode or push-pull output mode or N-channel open-drain output mode). You can also use port 2 pins as external interrupt (INT0–INT1) or A/D inputs. In addition, you can configure a pull-up resistor to individual pins using control register settings.

In normal operating mode, a reset clears P2CON to '00H', configuring P2.0–P2.3 as normal Schmitt trigger inputs.

You address port 2 bits directly by writing or reading the port 2 data register, P2 (E2H). The port 2 control register, P2CON is located at addresses EAH.

Two additional registers are used to control Port 2: P2DPUR (EBH) and P2PND (ECH). By setting port 2 open-drain and pull-up resistor enable register, P2DPUR, you can configure specific pins as open-drain or push-pull output. The application program polls the port 2 interrupt pending register, P2PND, to detect interrupt requests. When an interrupt request is acknowledged, the corresponding pending bit must be cleared by the interrupt service routine.

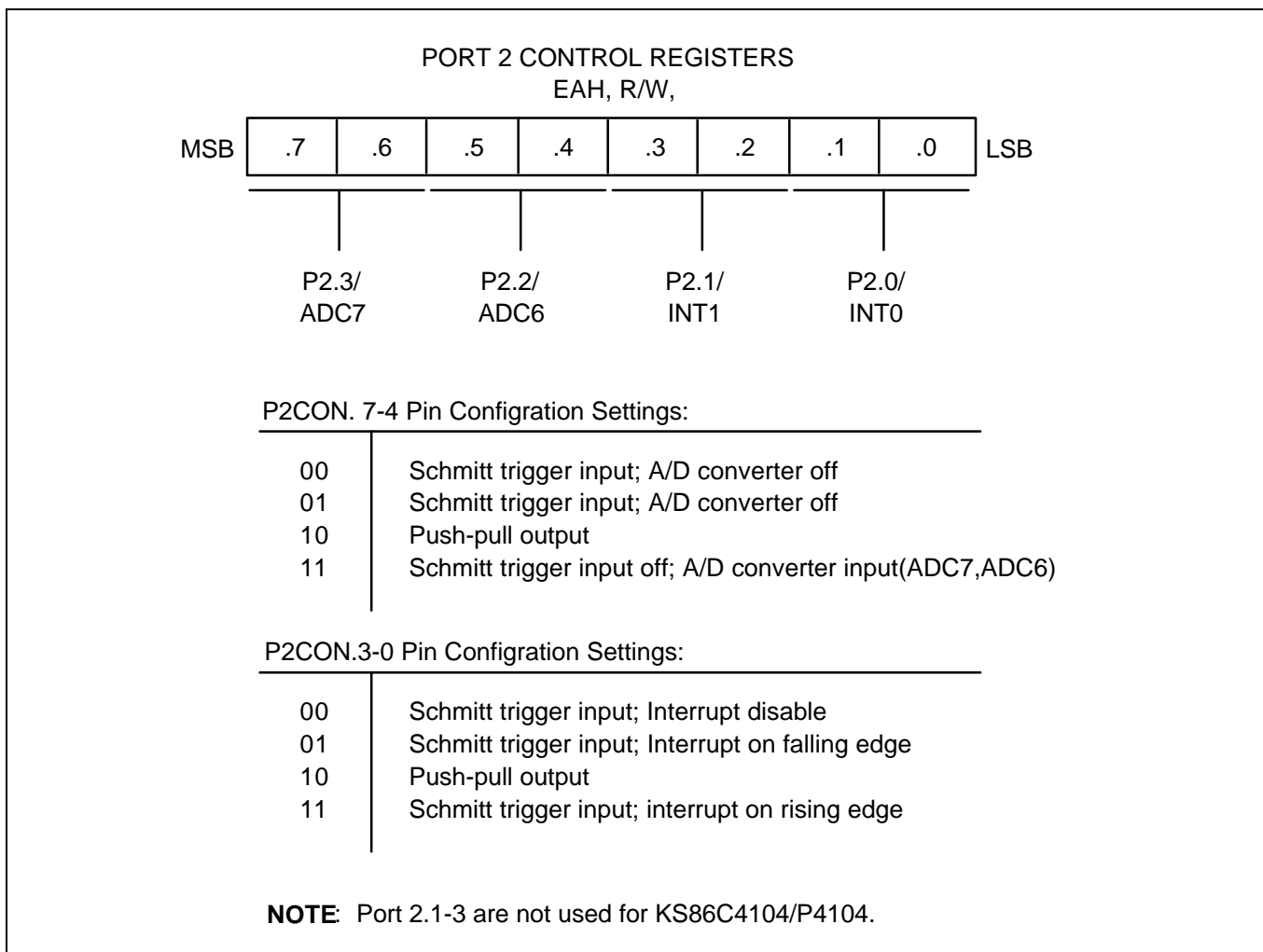


Figure 9-6. Port 2 Control Registers (P2CON)

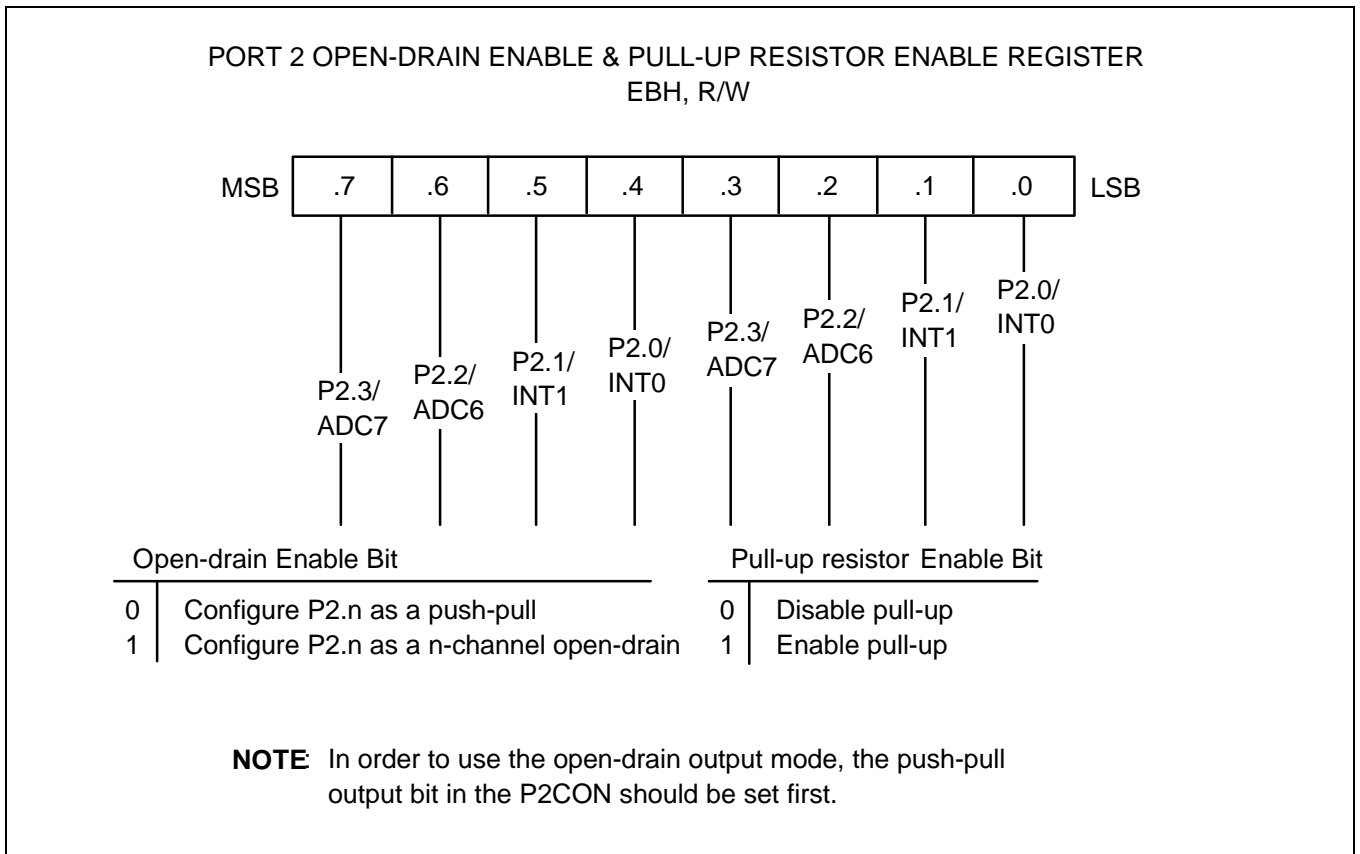


Figure 9-7. Port 2 Open-Drain and Pull-Up Resistor Enable Register (P2DPUR)

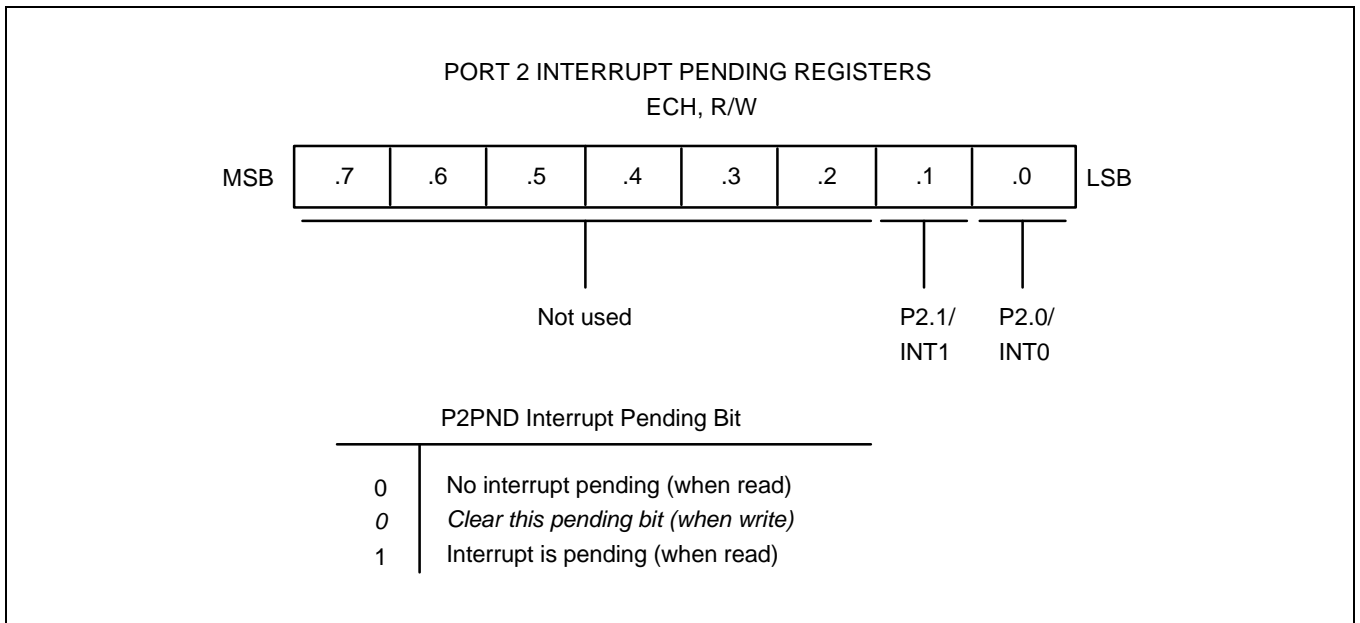


Figure 9-8. Port 2 Interrupt Pending Register (P2PND)

PORT 3

Port 3 is a 6-bit I/O port with individually configurable pins. It can be used for general I/O port. You can also use port 3 pins as A/D I/O inputs. In addition, you can configure a pull-up register to individual pins using control register settings.

In normal operating mode, reset clears P3CONH and P3CONL to '00H', configures P3.0–P3.5 schmitt trigger input mode. Using the P3CONH and P3CONL registers (EEH and EFH respectively), you can alternatively configure the port 3 pins as push-pull outputs, or as A/D converter input pins.

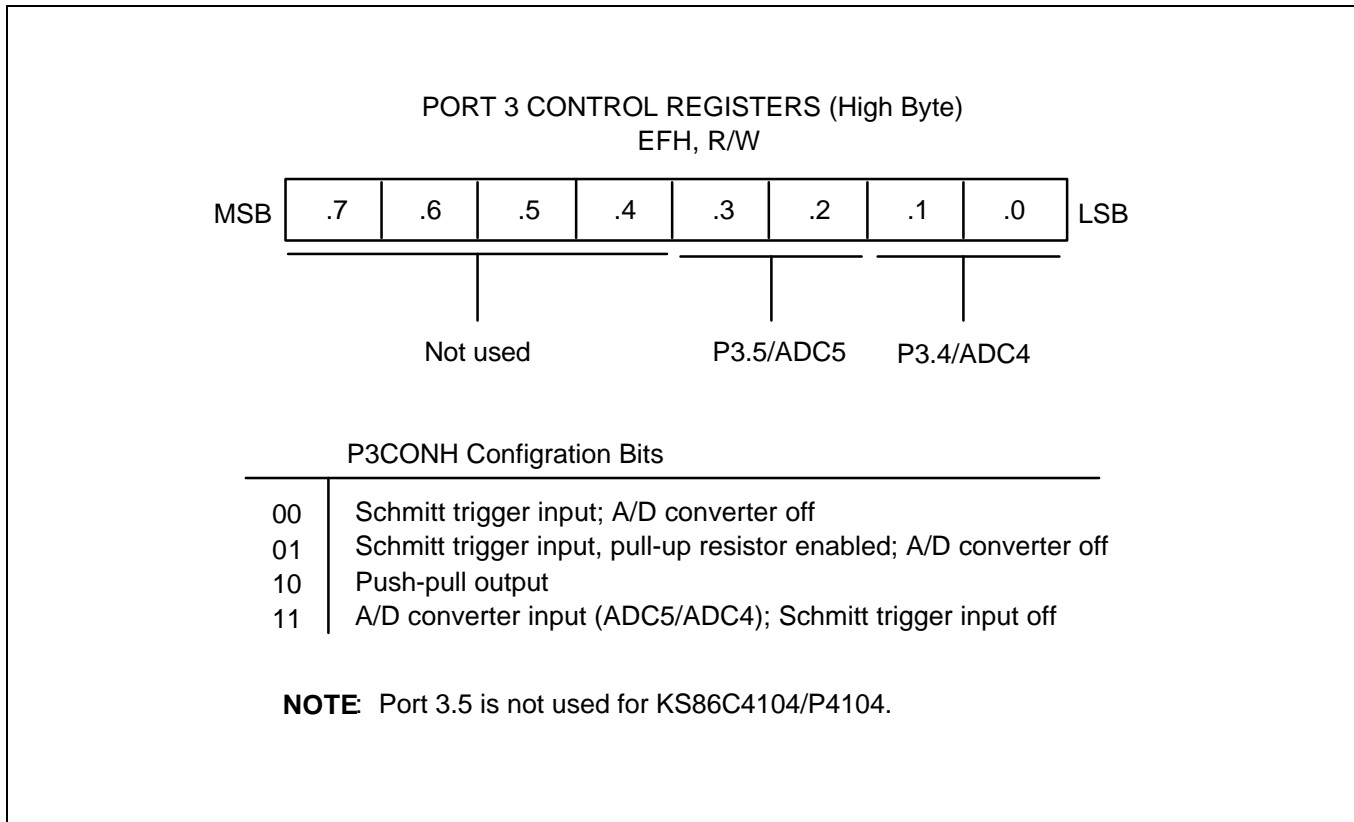


Figure 9-9. Port 3 High-Byte Control Registers (P3CONH)

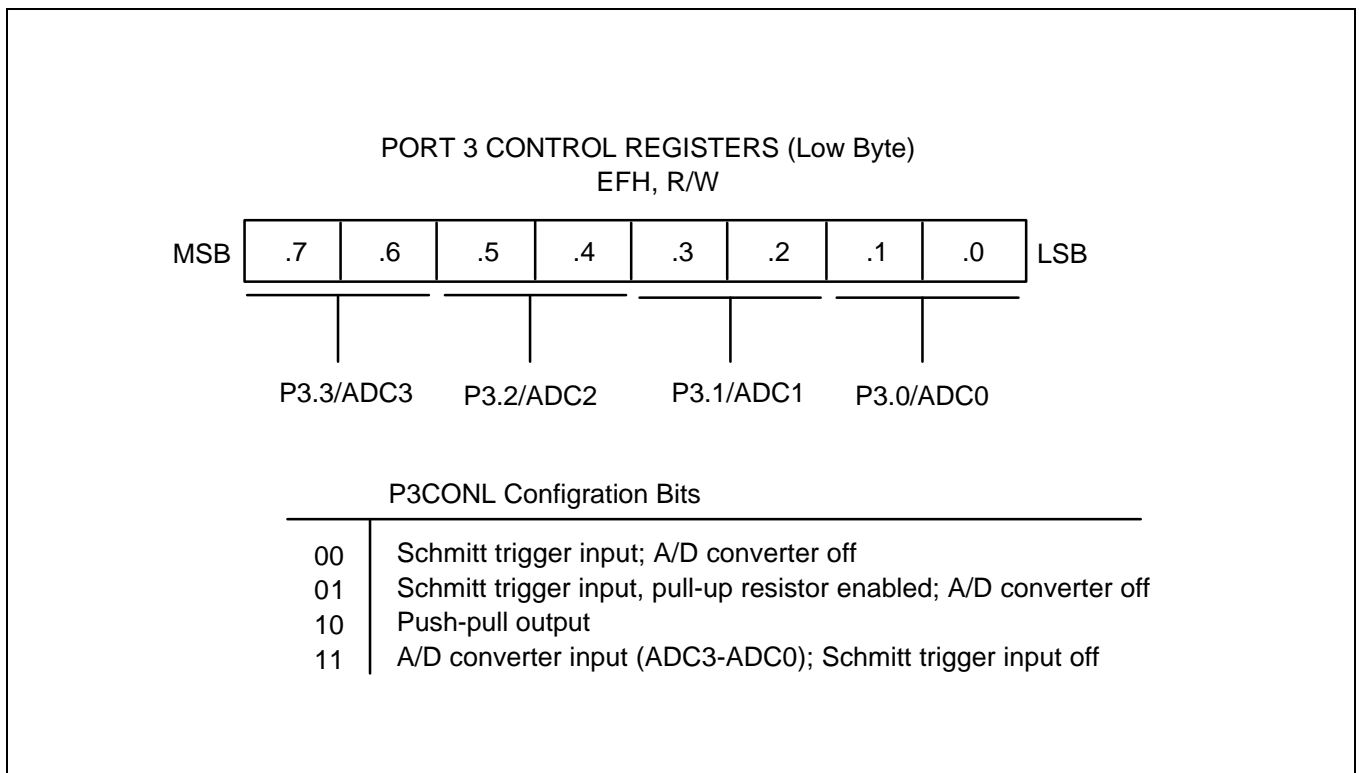


Figure 9-10. Port 3 Low-Byte Control Registers (P3CONL)

PROGRAMMING TIP – Configuring I/O Port Pins to Specification

The following sample program shows how to configure the KS86C4004 I/O ports to specification

Program comments show the effect of the settings:

```

.
.
.
LD      P0CON,#0FFH      ; push-pull output(LED direct drive-->Low active)
LD      P0PUR,#00H      ; disable pull-up resistor
LD      P0PNE,#00H      ; disable open-drain
LD      P1CON,#1010101011B ; P1.0 ZCD input enable / P1.1-3 push-pull
LD      ZCMOD,#00000010B ; enable falling edge interrupt
LD      T1DATA,#81H
LD      T1CON,#00001100B ; ZCD clear enable (fosc/512)
                          ; timer 1 interrupt disable
LD      P2CON,#11000110B ; P2.3 A/D input mode / P2.2 input mode
                          ; P2.1 falling edge interrupt
LD      P2DPUR,#00010110B ; P2.0 open-drain output mode
                          ; P2.2 pull-up enable
                          ; P2.1 pull-up enable
LD      P2PND,#00H      ; no interrupt pending
LD      P3CONH,#0FH      ; AD input mode(ADC4,ADC5)
LD      P3CONL,#0AAH     ; P3.0-3 push-pull output mode
.
.
.

```

10 BASIC TIMER and TIMERS

MODULE OVERVIEW

The KS86C4004/C4104 has three default timers: an 8-bit *basic timer*, one 8-bit general-purpose timer/counter, called *timer 0*, and one 8-bit timer/counter for the zero-crossing detection circuit called timer 1.

Basic Timer (BT)

You can use the basic timer (BT) in two different ways:

- As a watchdog timer to provide an automatic reset mechanism in the event of a system malfunction.
- To signal the end of the required oscillation stabilization interval after a reset or a Stop mode release.

The functional components of the basic timer block are:

- Clock frequency divider (f_{OSC} divided by 4096, 1024, or 128) with multiplexer
- 8-bit basic timer counter, BTCNT (DDH, read-only)
- Basic timer control register, BTCON (DCH, read/write)

Timer 0

Timer 0 has three operating modes, one of which you select by an appropriate T0CON setting:

- Interval timer mode
- Capture input mode
- 10-bit PWM output mode

Timer 0 has the following functional components:

- Clock frequency divider (f_{OSC} divided by 4096, 256, 8 or 1) with multiplexer
- 10(8)-bit counter (T0CNT+T0EXCNT), 8-bit comparator, and 10(8)-bit data register (T0DATA+PWMMEX)
- I/O pin (P1.2, T0 match or PWM) for timer 0 match/PWM output or capture input
- Timer 0 overflow interrupt (T0OVF) and match interrupt (T0INT) generation
- Timer 0 control registers, T0CONH and T0CONL (D2H and D3H respectively)

Timer 1

Timer 1 has one operating mode, interval timer mode. You can clear the timer 1 counter by appropriate setting of T1CON register. If T1CON.3 is set to "1", T1CNT is cleared by the ZCD edge detection.

Timer 1 has the following components:

- Clock frequency divider (f_{OSC} divided by 512, 256, 128 or 64)
- 8-bit counter (T1CNT), 8-bit compare register, and a 8-bit data register (T1DATA)
- Timer 1 control register, T1CON

BASIC TIMER (BT)

BASIC TIMER CONTROL REGISTER (BTCON)

The basic timer control register, BTCON, is used to select the input clock frequency, to clear the basic timer counter and frequency dividers, and to enable or disable the watchdog timer function.

A reset clears BTCON to '00H'. This enables the watchdog function and selects a basic timer clock frequency of $f_{OSC}/4096$. To disable the watchdog function, you must write the signature code '1010B' to the basic timer register control bits BTCON.7–BTCON.4.

The 8-bit basic timer counter, BTCNT, can be cleared during normal operation by writing a "1" to BTCON.1. To clear the frequency dividers for both the basic timer input clock and the timer 0 clock, you write a "1" to BTCON.0.

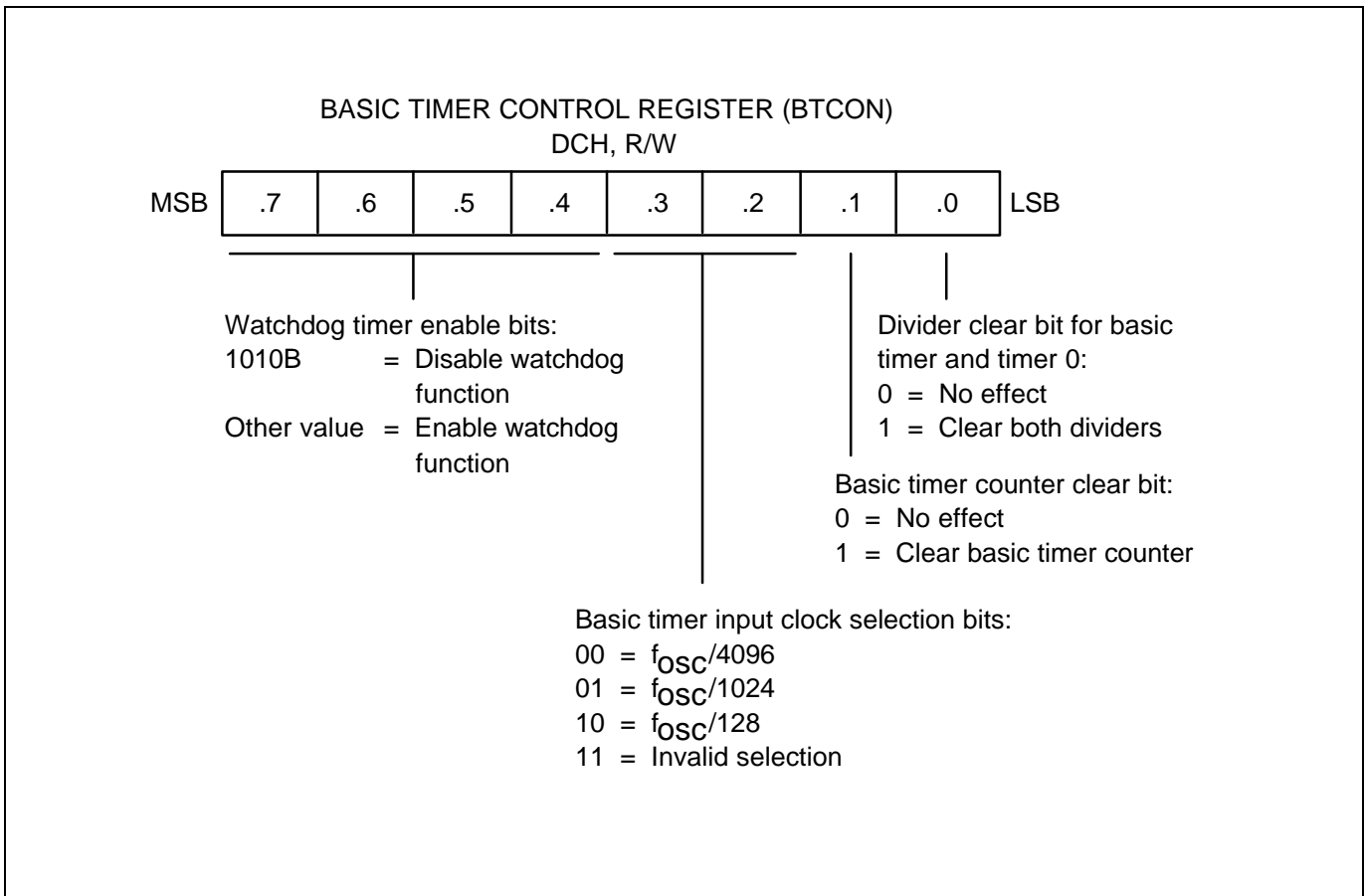


Figure 10-1. Basic Timer Control Register (BTCON)

BASIC TIMER FUNCTION DESCRIPTION

Watchdog Timer Function

You can program the basic timer overflow signal (BTOVF) to generate a reset by setting BTCON.7–BTCON.4 to any value other than '1010B' (The '1010B' value disables the watchdog function). A reset clears BTCON to '00H', automatically enabling the watchdog timer function. A reset also selects the CPU clock (as determined by the current CCON register setting) divided by 4096 as the BT clock.

A reset whenever a basic timer counter overflow occurs. During normal operation, the application program must prevent the overflow, and the accompanying reset operation, from occurring. To do this, the BTCNT value must be cleared (by writing a "1" to BTCON.1) at regular intervals.

If a system malfunction occurs due to circuit noise or some other error condition, the BT counter clear operation will not be executed and a basic timer overflow will occur, initiating a reset. In other words, during normal operation, the basic timer overflow loop (a bit 7 overflow of the 8-bit basic timer counter, BTCNT) is always broken by a BTCNT clear instruction. If a malfunction does occur, a reset is triggered automatically.

Oscillation Stabilization Interval Timer Function

You can also use the basic timer to program a specific oscillation stabilization interval following a reset or when Stop mode has been released by an external interrupt.

In Stop mode, whenever a reset or an external interrupt occurs, the oscillator starts. The BTCNT value then starts increasing at the rate of $f_{OSC}/4096$ (for reset), or at the rate of the preset clock source (for an external interrupt). When BTCNT.4 is set, a signal is generated to indicate that the stabilization interval has elapsed and to gate the clock signal off to the CPU so that it can resume normal operation.

In summary, the following events occur when Stop mode is released:

1. During Stop mode, a power-on reset or an external interrupt occurs to trigger the Stop mode release and oscillation starts.
2. If a power-on reset occurred, the basic timer counter will increase at the rate of $f_{OSC}/4096$. If an external interrupt is used to release Stop mode, the BTCNT value increases at the rate of the preset clock source.
3. Clock oscillation stabilization interval begins and continues until bit 4 of the basic timer counter is set.
4. When a BTCNT.4 is set, normal CPU operation resumes.

Figure 10–2 and 10–3 shows the oscillation stabilization time on RESET and STOP mode release

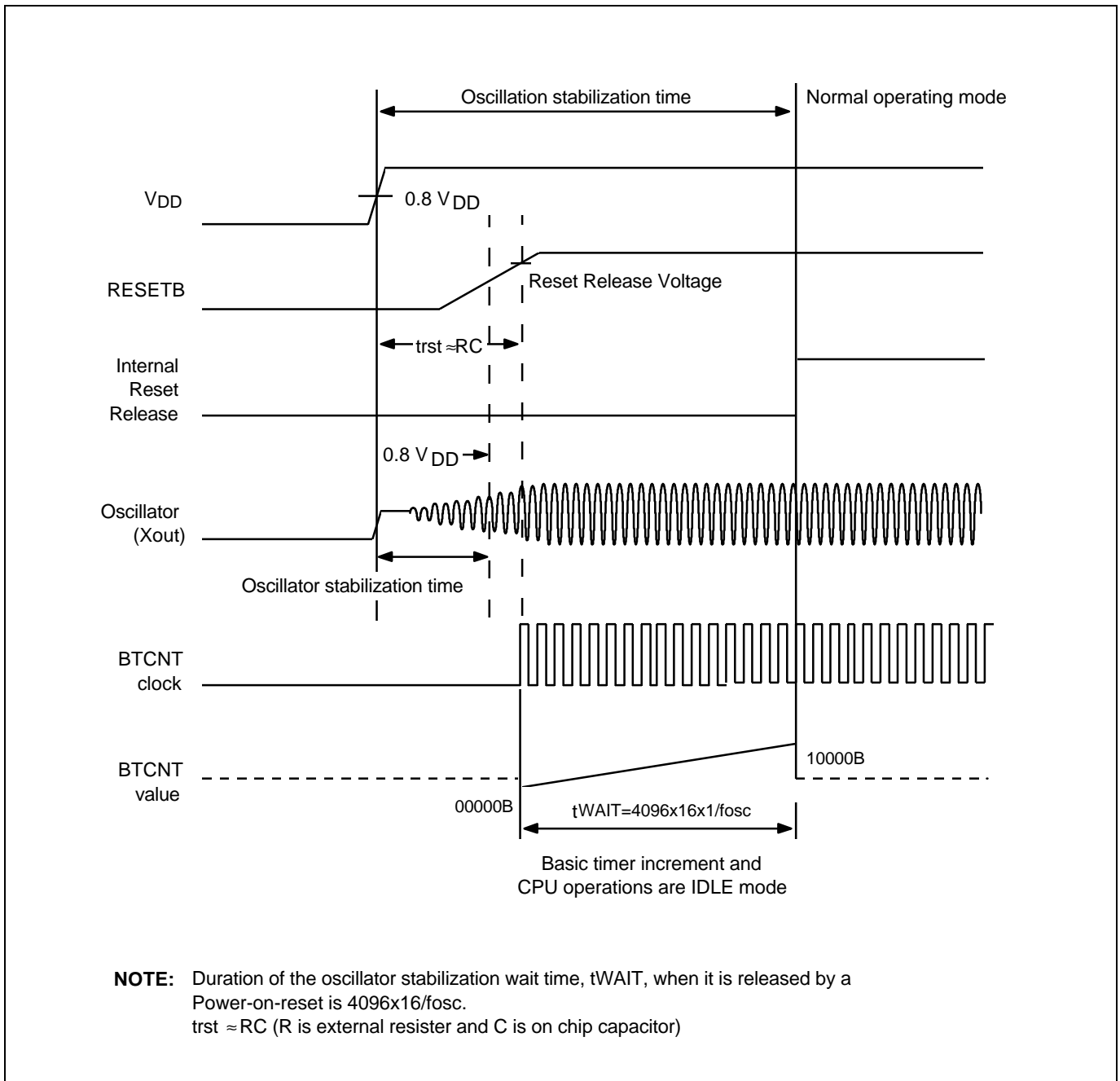


Figure 10-2. Oscillation Stabilization Time on RESET

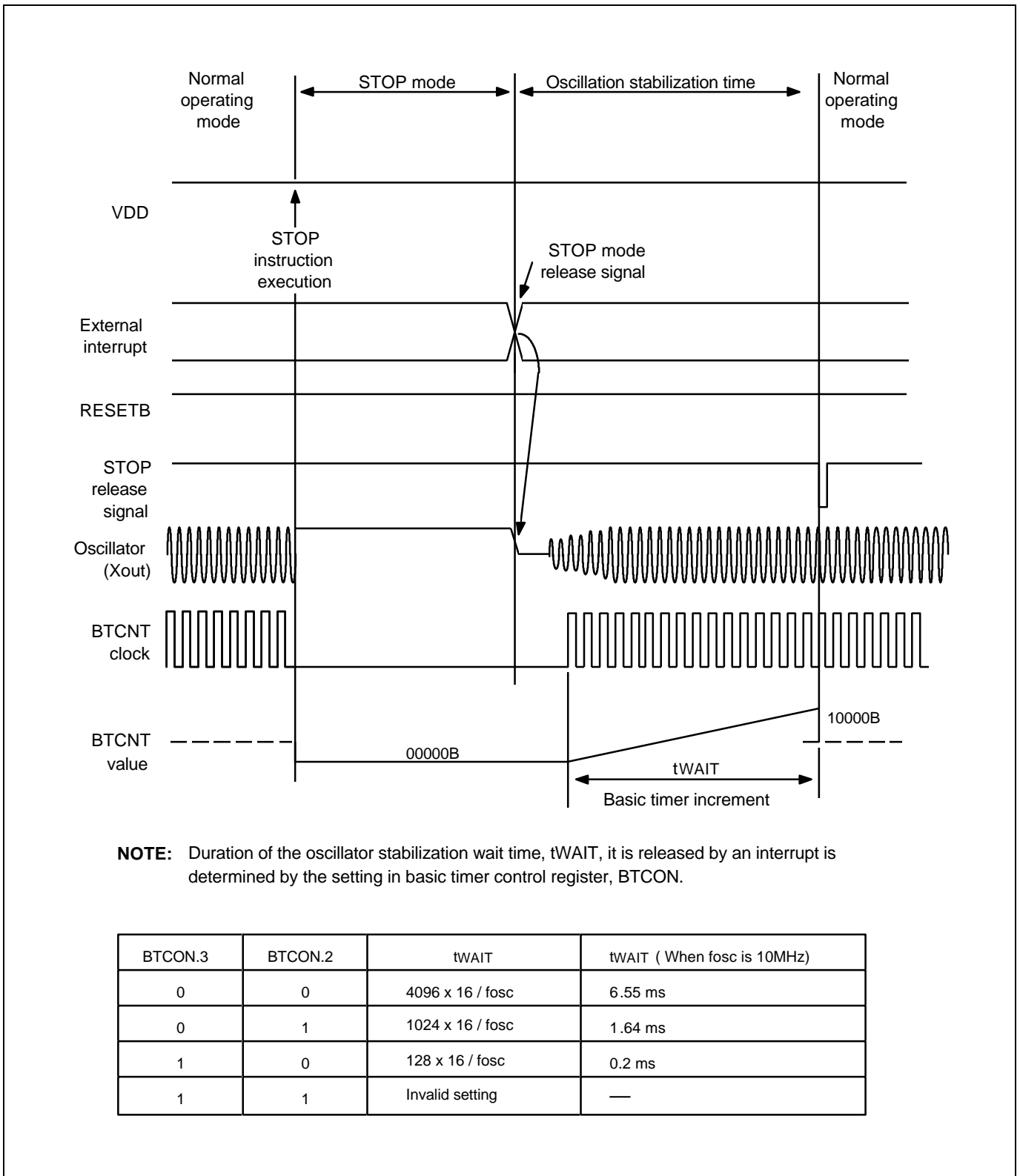


Figure 10-3. Oscillation Stabilization Time on STOP Mode Release

Programming Tip —Configuring the Basic Timer

This example shows how to configure the basic timer to sample specification

```
;-----<< Interrupt vector address>>
```

```
    ORG    0000H
    VECTOR 00H, COMMON_INT      ; IRQ0 / Interrupt vector address
```

```
;-----<< Initialize system and peripherals>>
```

```
RESET  ORG      0100H          ; Reset start address
        DI              ; disable interrupt
        LD      BTCON,#10100010B ; disable watch-dog function
                                           ; clock source:fosc/4096(104ms overflow at 10MHz)
        LD      CLKCON,#00011000B ; CPU clock source select(non-divided)
        LD      SP,#0C0H      ; KS86C4004 Stack pointer initial
        .
        .
        .
        EI              ; enable interrupt
```

```
;-----<< Main loop >>
```

```
MAIN
        .
        .
        .
        LD      BTCON,#02H      ; enable watch-dog function
                                           ; basic Timer Counter(BTCNT) clear
        .
        .
        .
        JP      T,MAIN         ; for main loop
        .
        .
        .
```

TIMER 0

Timer 0 Control Registers (T0CONH AND T0CONL)

The timer 0 control register low byte, T0CONL, is used to select the timer 0 operating mode (interval timer, capture mode, or PWM mode) and input clock frequency, to clear the timer 0 counter, and to enable the T0 overflow interrupt and T0 match/capture interrupt. It also contains a pending bit for T0 match/capture interrupts.

Timer 0 control register high byte, T0CONH, contains a pending bit for T0 overflow interrupt. Only one bit in T0CONH register is used, T0CONH.0.

A reset clears T0CONL to '00H'. This sets timer 0 to normal interval timer mode, selects an input clock frequency of $f_{OSC} / 4096$, and disables the T0 overflow interrupt and match/capture interrupts. The T0 counter can be cleared at any time during normal operation by writing a "1" to T0CONL.3.

The T0 overflow interrupt, T0OVF, is IRQ0 with vector 00H. When a T0 overflow interrupt occurs and is serviced by the CPU, the pending condition is cleared manually set by writing "0" to T0CONH.0. To enable the T0 match/capture interrupt (T0INT, IRQ0, vector 00H), you must set T0CONL.1 to "1". The interrupt service routine must clear the pending condition by writing a "0" to the T0 interrupt pending bit, T0CONL.0.

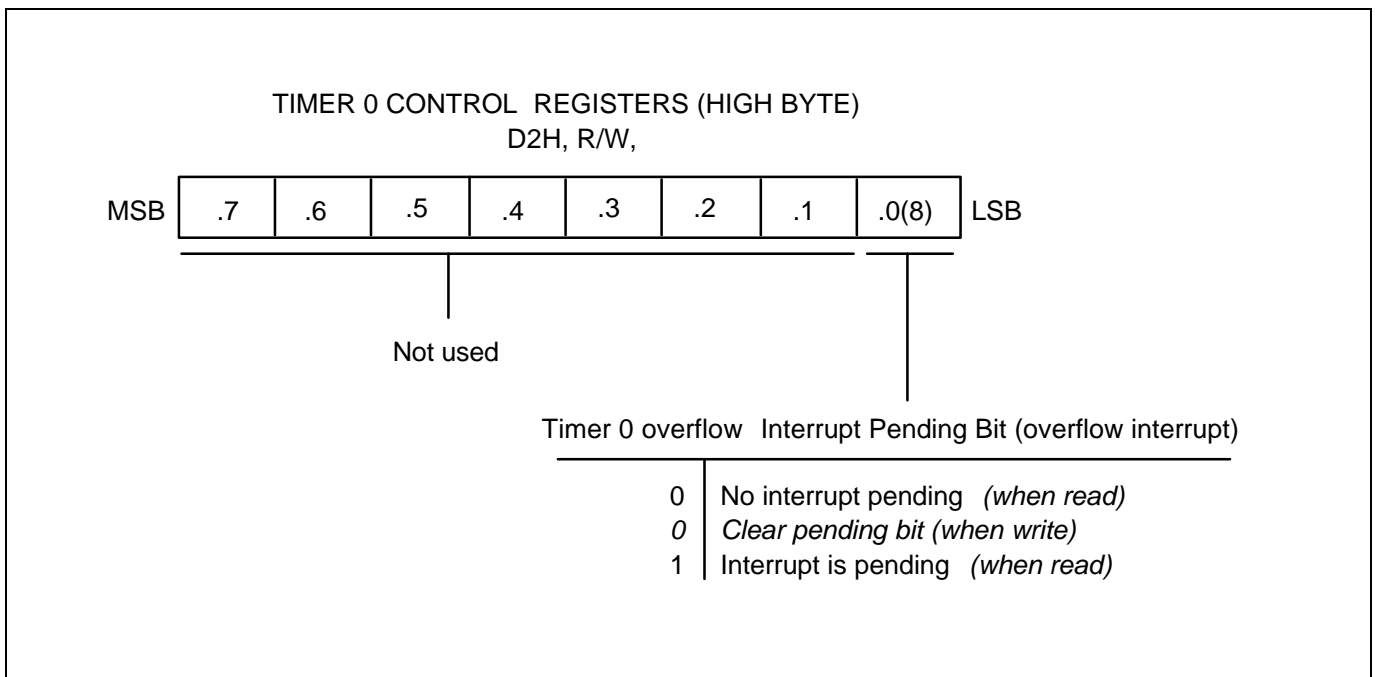


Figure 10-4. Timer 0 Control Registers (T0CONH)

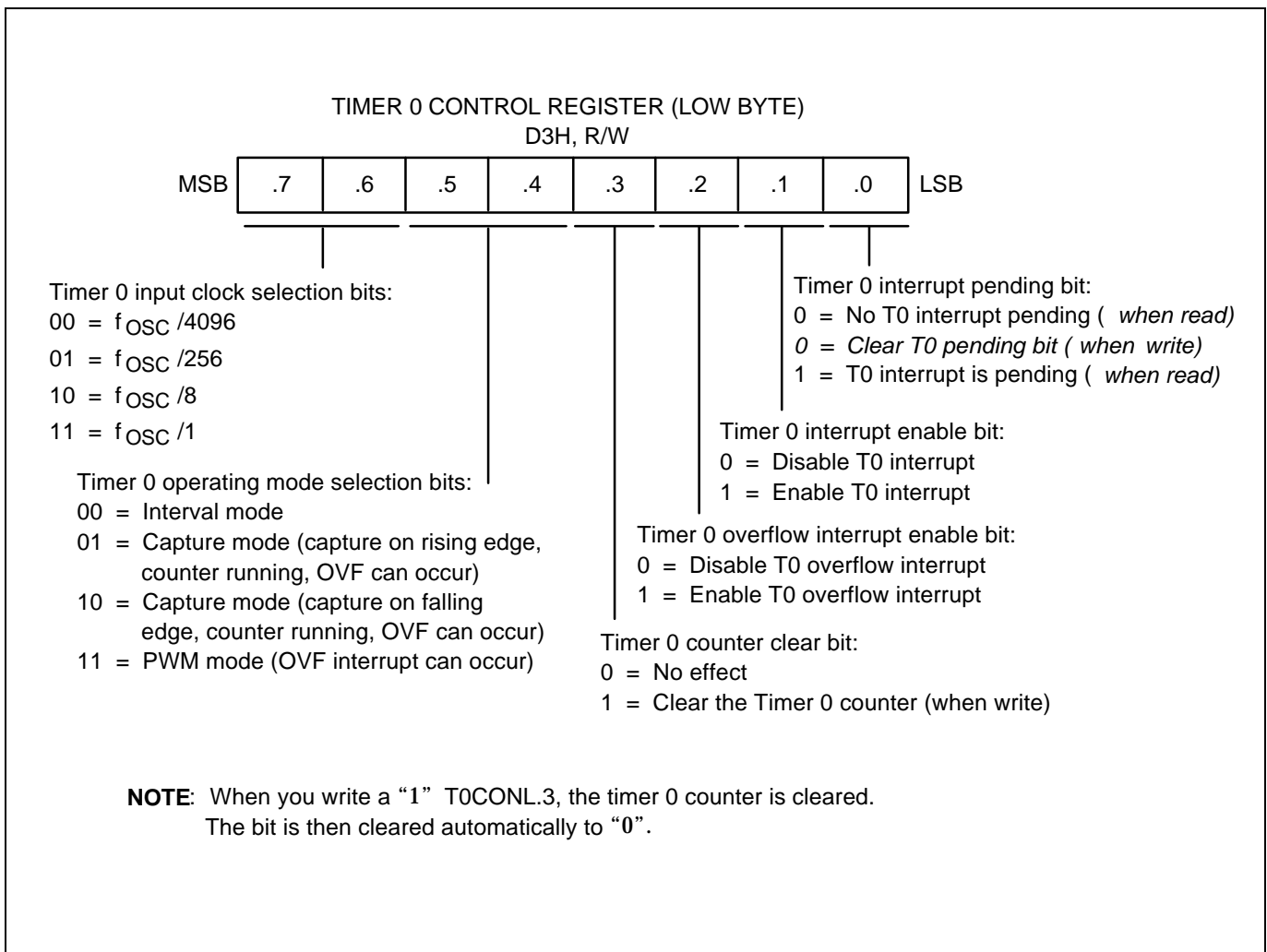


Figure 10-5. Timer 0 Control Registers (T0CONL)

TIMER 0 FUNCTION DESCRIPTION

Timer 0 Interrupts (IRQ0, Vectors 00H)

The Timer 0 module can generate two interrupts; the timer 0 overflow interrupt (T0OVF), and the timer 0 match/capture interrupt (T0INT). T0OVF is interrupt level IRQ0, vector 00H; T0INT is also level IRQ0, vector 00H. The T0OVF interrupt pending condition is cleared by setting the T0CONH.0 pending bit to "0". The T0INT pending condition must be cleared by software by writing a "0" to the T0CONL.0 pending bit.

Interval Timer Mode

In interval timer mode, a match signal is generated when the counter value is identical to the value written to the Timer 0 reference data register, T0DATA. The match signal generates a Timer 0 match interrupt (T0INT, vector 00H) and then clears the counter. If, for example, you write the value '10H' to T0DATA, the counter will increment until it reaches '10H'. At this point, the Timer 0 interrupt request is generated, the counter value is reset and counting resumes. With each match, the level of the signal at the Timer 0 output pin is inverted.

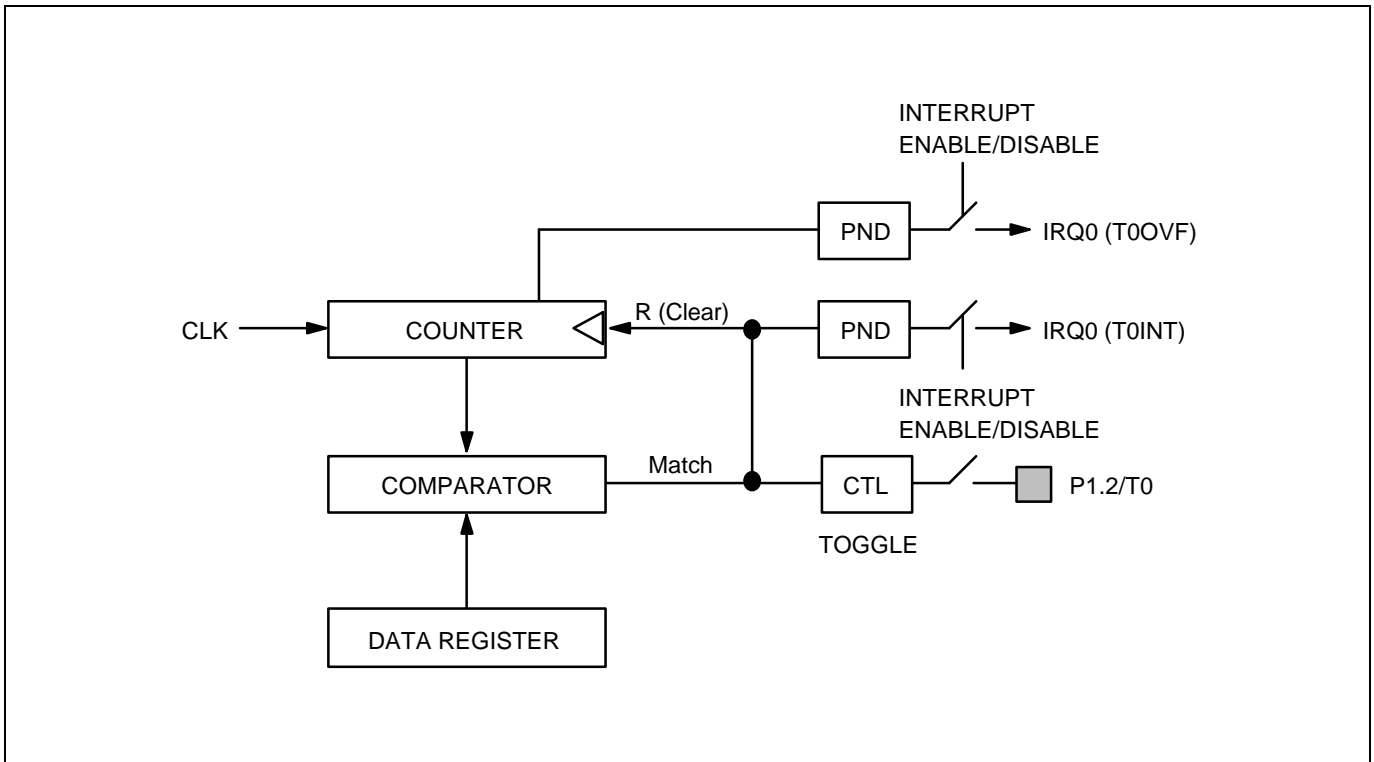


Figure 10-6. Simplified Timer 0 Function Diagram (Interval Timer Mode)

Pulse Width Modulation Mode

Pulse width modulation (PWM) mode lets you program the width (duration) of the pulse that is output at the T0 pin. As in interval timer mode, a match signal is generated when the counter value is identical to the value written to the T0 data register (T0DATA). In PWM mode, however, the match signal does not clear the counter (it runs continuously, overflowing at 'FFH', and continues incrementing from '00H').

Although it is possible to use the match signal to generate a T0INT interrupt, an interrupt is typically not used in PWM-type applications. Instead, the pulse at the T0 pin is held to High level as long as the data register (T0DATA) value is *greater than* the counter (T0CNT) value for 8-bit PWM operation. For 2-bit extension control logic operation (10-bit PWM) the pulse is held to High level when the data value is *equal to* the counter value. One frame width is equal to $t_{CLK} \times 1024$. (See Figure 10–9.)

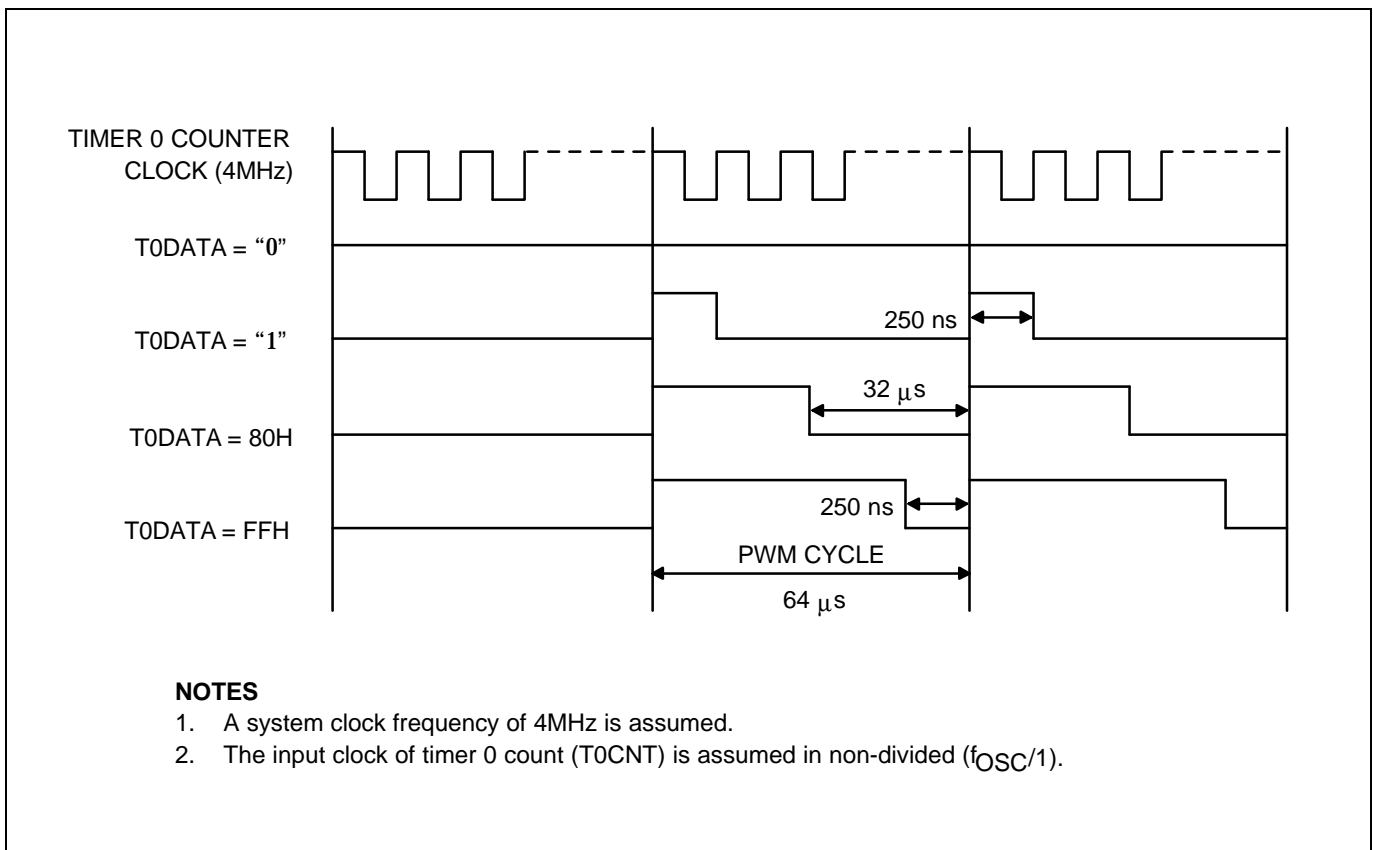


Figure 10–7. Simplified Timer 0 Function Diagram (PWM Mode)

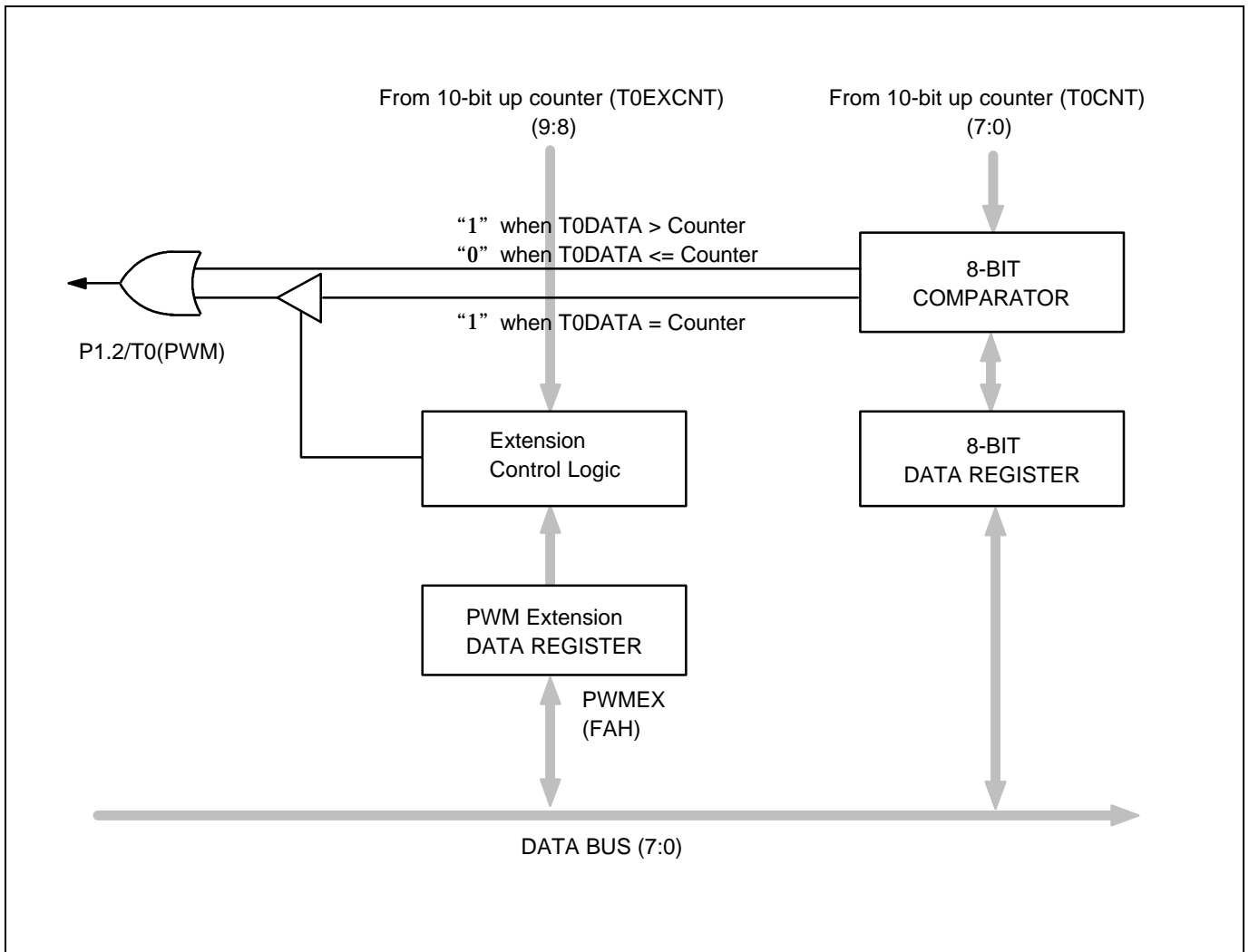


Figure 10-8. PWM Block Function Diagram

Table 10-1. PWM Output “Stretch” Values for Extension Register PWMEX

PWMEX	Cycle Number That is “Stretched”
00H	Not Stretched
01H	2
02H	1, 3
03H	1, 2, 3

NOTE: Bits 0 and 1 of the PWM extension register PWMEX are used only.

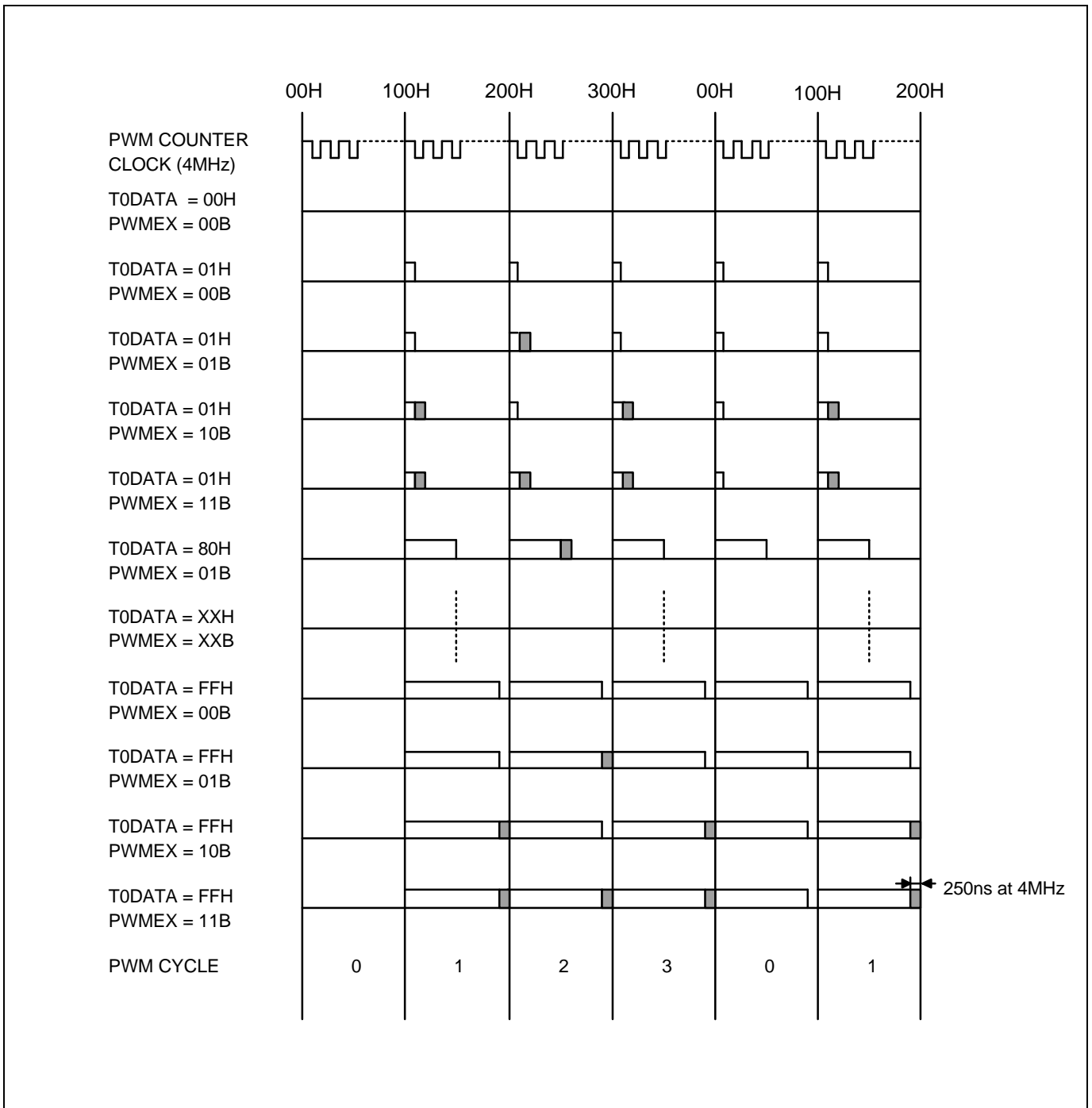


Figure 10-9. 10-Bit PWM Waveforms with Various T0DATA and PWMEX

Capture Mode

In capture mode, a signal edge that is detected at the T0 pin opens a gate and loads the current counter value into the T0 data register. Rising edges or falling edges can be selected to trigger this operation. Both kinds of T0 interrupts can be used in capture mode: T0OVF is generated when a counter overflow occurs, and T0INT is generated when the counter value is loaded into the data register. By reading the captured data value in T0DATA, and assuming a specific value for t_{CLK} , you can determine the pulse width (duration) of the signal being input at the T0 pin. (See Figure 10–10.)

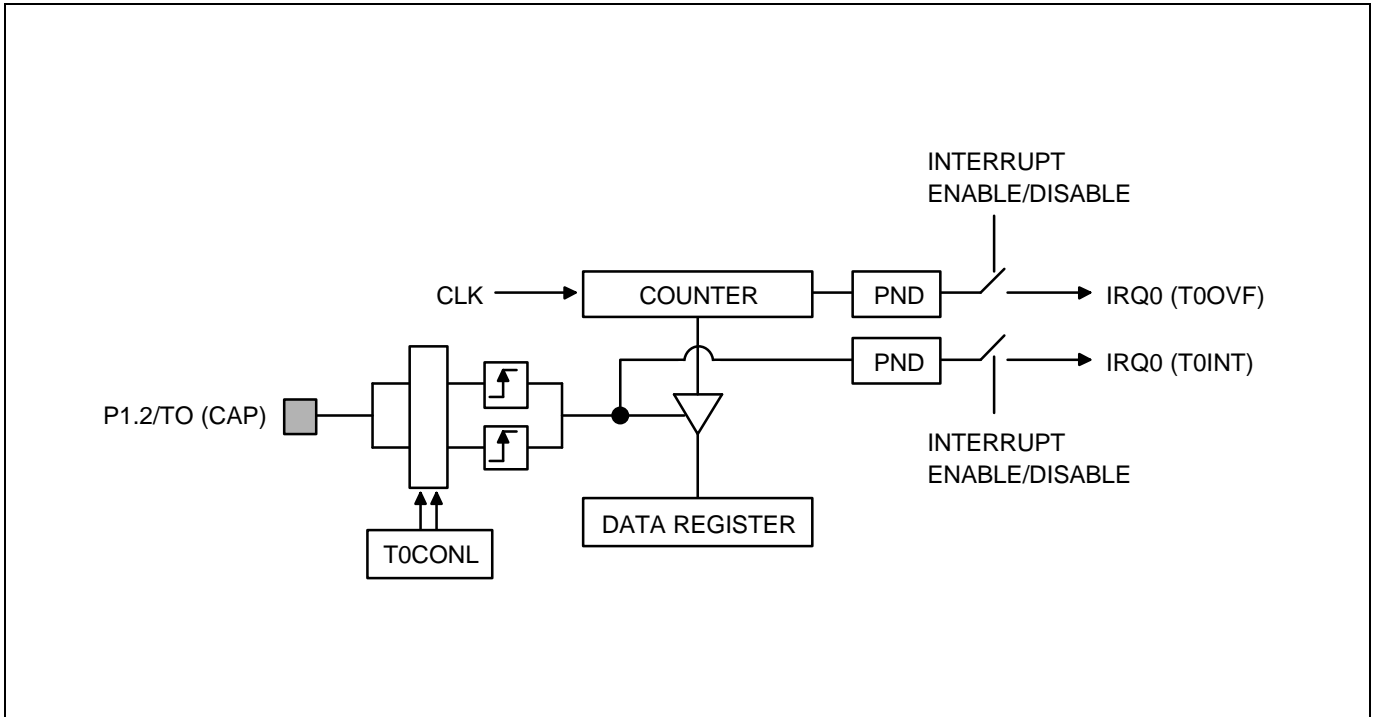


Figure 10–10. Simplified Timer 0 Function Diagram (Capture Mode)

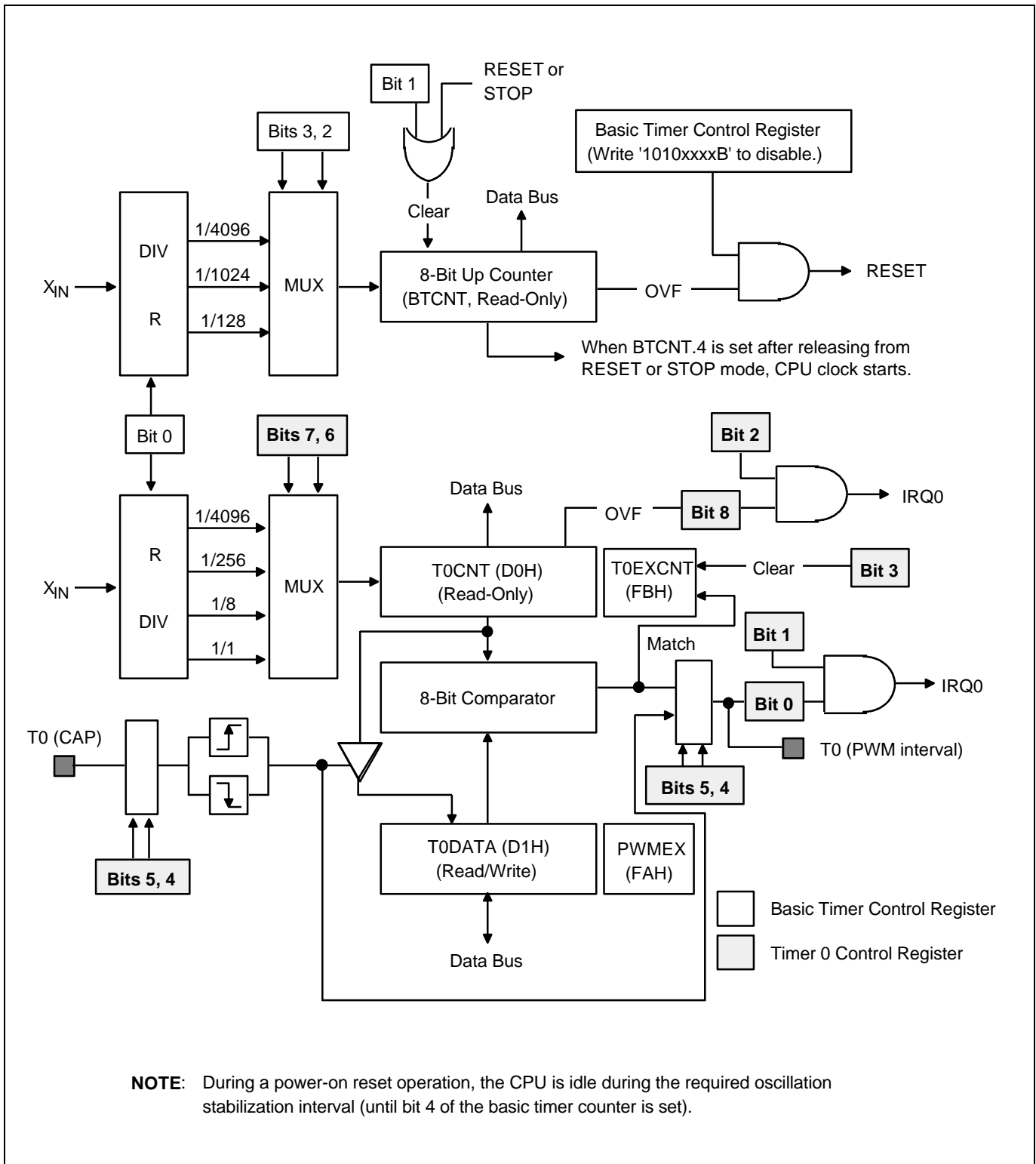


Figure 10-11. Basic Timer and Timer 0 Block Diagram

PROGRAMMING TIP1 -- Configuring Timer 0 (Interval Mode)

The following sample program sets Timer 0 to interval timer mode.

;-----<< Interrupt vector address >>

```
ORG 0000H
VECTOR 00H, COMMON_INT ; IRQ0 / Interrupt vector address
```

;-----<< Initialize system and peripherals >>

```
RESET      ORG 0100H ; Reset start address
           DI ; disable interrupt
           LD BTCON,#00000010B ; enable watch-dog function
           ; clock source:fosc/4096(104ms overflow at 10MHz)
           LD CLKCON,#00011000B ; CPU clock source select(non-divided)
           LD SP,#0C0H ; KS86C4004 Stack pointer initial
           .
           .
           .
           LD T0DATA,#26H ; 1ms interval at 10MHz system clock
           ; 100ns x 256 x 39 = 998.4us
           LD T0CONH,#00H ; timer 0 overflow interrupt disable
           LD T0CONL,#01001010B ; timer 0 match interrupt enable
           ; clock source : fosc/256
           EI ; enable interrupt
```

;-----<< Main loop >>

```
MAIN      .
           .
           .
           LD BTCON,#02H ; enable watch-dog function
           ; basic counter(BTCNT) clear
           JP T,MAIN ; for main loop
           .
           .
```

;-----<< Interrupt service routine >>

```
COMMON_INT
          TM T0CONL,#00000001B
          JP NZ,TIMER0_INT
          .
          .
          .
          IRET
```

```

TIMER0_INT  AND  T0CONL,#11111110B      ; Timer0 pending bit clear
            INC  TIMER_MODE
            CP   TIMER_MODE,#5
            JR   ULT,TMODE_JP
            CLR  TIMER_MODE

TMODE_JP    LD   R9,TIMER_MODE
            RCF
            RLC  R9                      ; TIMER_MODE X 2
            CLR  R8
            LDC  R10,#TBL_TMODE[RR8]
            LDC  R11,#TBL_TMODE+1[RR8]
            CALL @RR10                   ; MULTI CALL
            .
            .
            IRET

TBL_TMODE   DW   DSP_7SEGMENT           ; MULTI CALL ADDRESS
            .
            .
            DW   KEY_SCAN              ; TA_MODE = 4

DSP_7SEGMENT ; display
            .
            .
            RET

KEY_SCAN    ; key scanning
            .
            RET
            .
            .
            .
            END

```

PROGRAMMING TIP2 -- Configuring Timer 0 (PWM Mode)

The following sample program sets Timer 0 to 10-bit PWM mode.

```

      .
      .
      .
RESET  ORG  0100H          ; reset start address
      DI                    ; disable interrupt
      LD  BTCON,#10100010B ; disable watch-dog function
      LD  CLKCON,#00011000B ; CPU clock source select(non-divided)
      LD  SP,#0C0H         ; KS86C4004 Stack pointer initial
      .
      .
      LD  T0CONH,#00H      ; timer0 overflow interrupt disable
      LD  P1CON,#10111010B ; P1.2 PWM output mode
      .
      .
;-----<< MAIN LOOP >>
MAIN   .
      .
      .
      LD  BTCON,#02H       ; enable watch-dog function
      .
      .
      JP  T,MAIN           ; basic counter(BTCNT) clear
      .
      .
      .
      .
      .
      .
      LD  T0DATA,#34        ; duty = 34/256(High width)
      LD  PWMEX,#02H        ; total duty = 138/1024(High width)
      LD  T0CONL,#11111000B ; timer0 PWM mode / non-divided
      .
      .
      .
      .
      LD  T0DATA,#80H       ; duty = 128/256 (High width)
      LD  PWMEX,#00H        ; total duty = 512/1024(High width)
      .
      .
      .

```


TIMER 1

TIMER 1 CONTROL REGISTER (T1CON)

The timer 1 control register, T1CON, located at F3H operates in interval timer mode. By setting the appropriate bits in T1CON you can select the input clock frequency and enable the Timer 1 interrupt. T1CON also contains a pending bit for Timer 1 interrupt.

A reset clears T1CON to '00H'. This sets timer 1 to normal interval mode and selects an input clock frequency of $f_{osc}/512$ and disables the Timer 1 interrupt.

You may clear the timer 1 counter by either setting T1CON.2 to "1" or enable ZCD clear signal to clear the timer 1 counter by setting T1CON.3 to "1".

To enable Timer 1 match interrupt (IRQ0, vector 00H) you must set T1CON.1 to "1". The interrupt service routine must clear the pending condition by writing a "0" to the Timer 1 interrupt pending bit, T1CON.0.

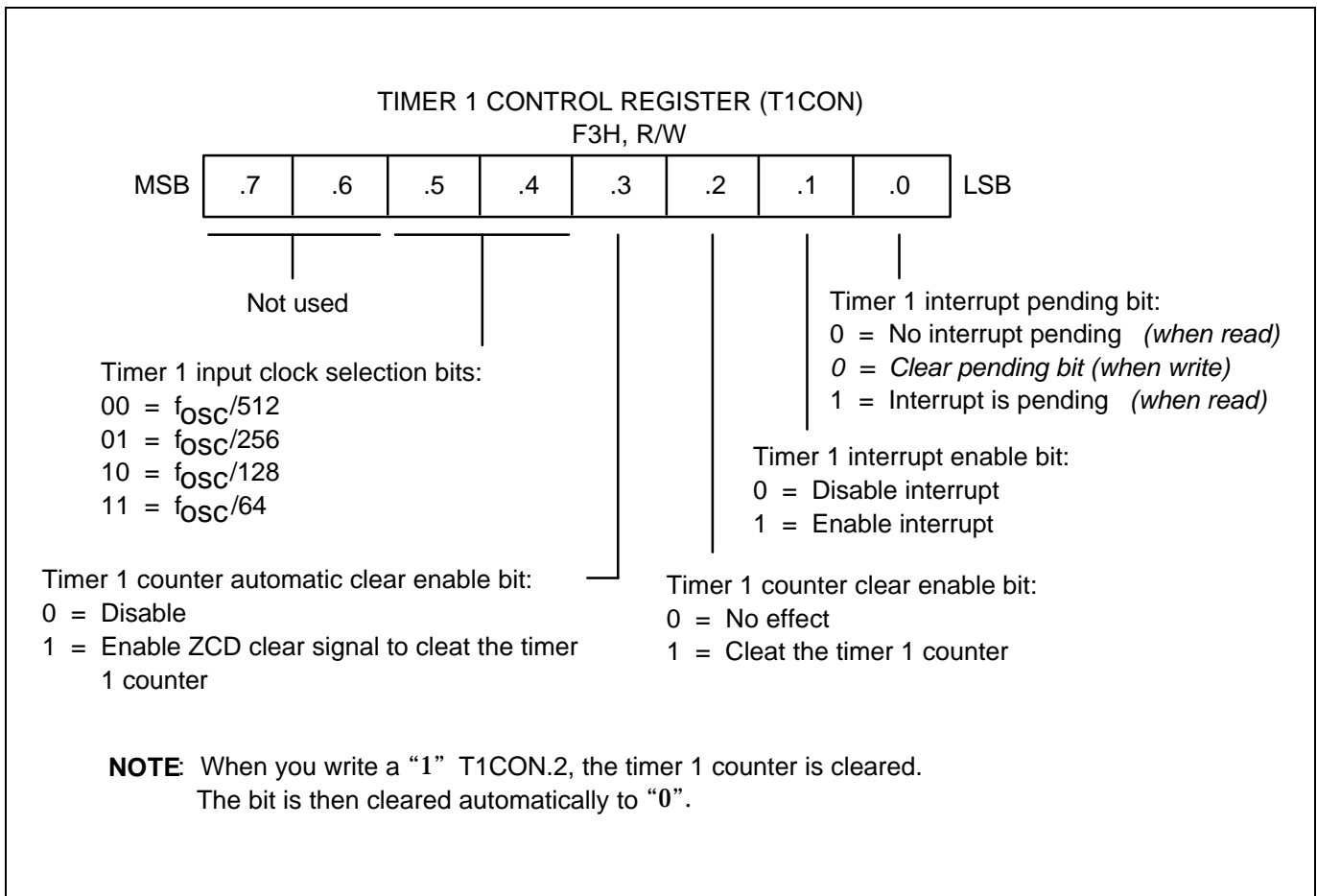


Figure 10-12. Timer 1 Control Register (T1CON)

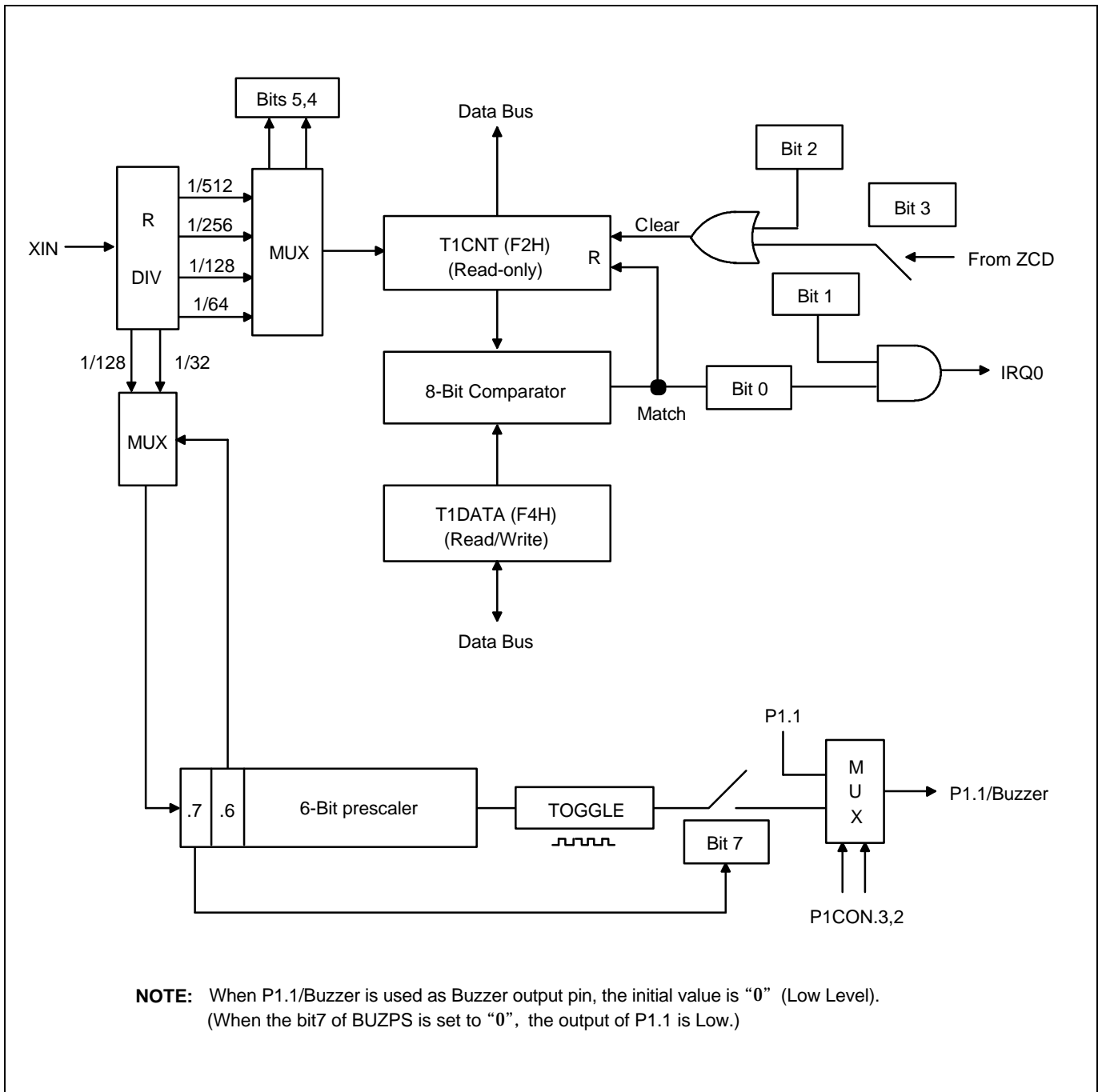


Figure 10-13. Timer 1 Block Diagram

BUZZER OUTPUT CONTROL REGISTER (BUZPS)

Buzzer output control register is used to select the frequency from 200 Hz to 20 K Hz. And these various frequency can be used to generate the melody signal. By selecting the clock source (bit of BUZPS) and the value of prescaler, the desire frequency can be obtained. The BUZPS.7 can be used to control the buzzer output when P1.1 is set to buzzer output mode (configure P1CON).

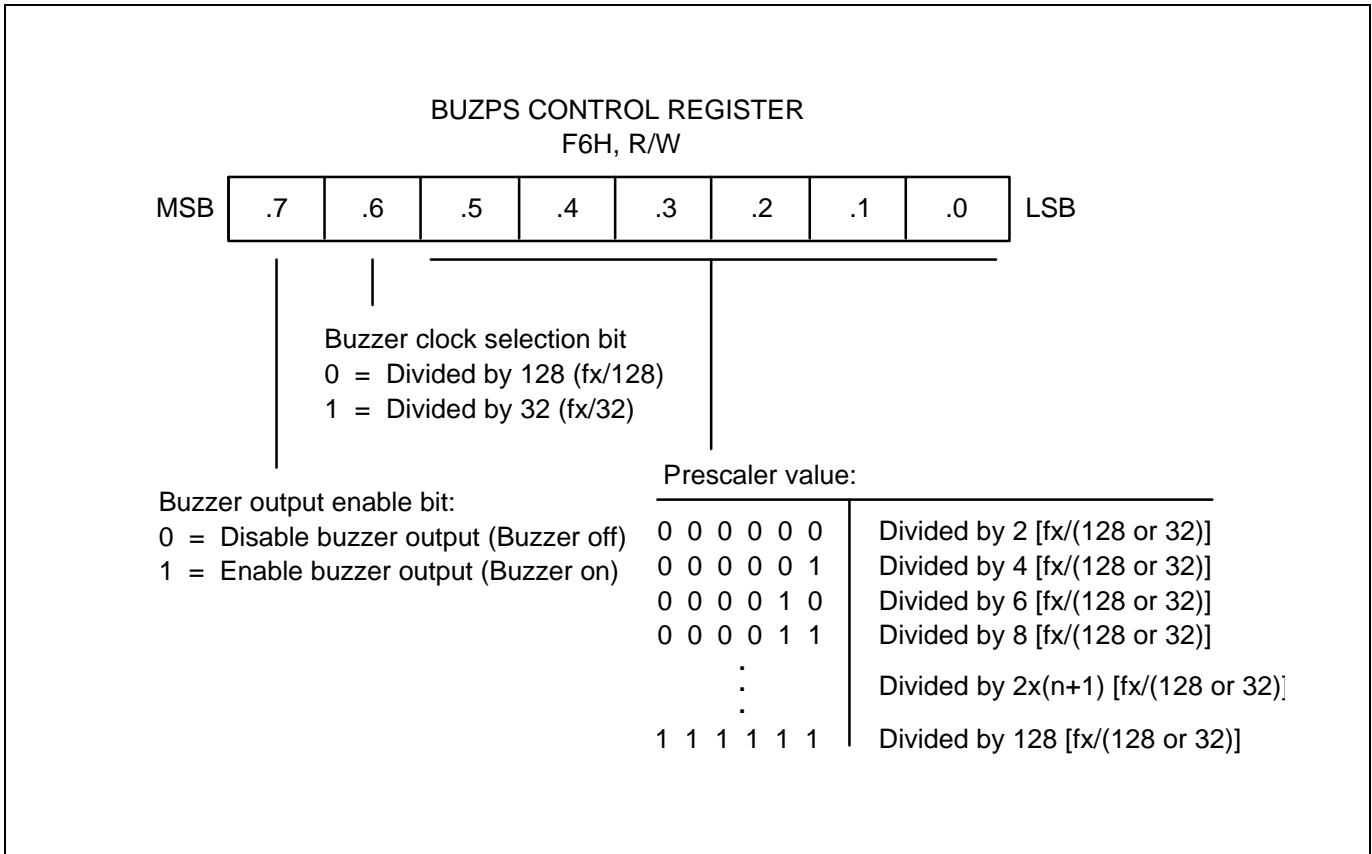


Figure 10-14. Buzzer output control register (BUZPS)

PROGRAMMING TIP – Configuring Timer 1

The following sample program sets Timer 1 to interval timer mode and generates the melody signal by using buzzer.

```

.INCLUDE "C:\SMDS2P\INCLUDE\REG\86C4004.REG"

;-----<< Interrupt vector address >>

.ORG 0000H
.VECTOR 00H, COMMON_INT ; IRQ0 / Interrupt vector address

;-----<< Initialize system and peripherals >>

RESET: .ORG 0100H ; reset start address
        DI ; disable interrupt
        LD BTCON,#00000010B ; enable watch-dog function
        ; clock source:fosc/4096(262ms overflow
        ; at 4MHz)
        LD CLKCON,#00011000B ; CPU clock source select(non-divided)
        LD SP,#0C0H ; KS86C4004 Stack pointer initial
        LD P1CON,#00001100B ; P1.1 buzzer output mode
        LD T1DATA,#4DH ; 10ms interval at 4MHz system clock
        ; 250ns x 512 x 78 = 9.984ms
        LD T1CON,#00000110B ; timer1 match interrupt enable
        ; clock source : fosc/512
        .
        .
        EI ; enable interrupt

;-----<< Main loop >>

MAIN: .
      .
      .
      LD BTCON,#02H ; enable watch-dog function
      ; basic counter(BTCNT) clear
      JP T,MAIN ; for main loop

MELODY_CHK:
      OR TIMER_CHK,#00000100B
      OR MELODY_FLAG,#00000001B ; melody signal enable
      LD MUSIC_INTERVAL,#1
      CLR BEEP_POINTER
      RET

;-----<< Interrupt service routine >>

COMMON_INT:
      TM T1CON,#00000001B ; timer 1 interrupt pending check

```

```

    JP      NZ,TIMER1_INT
    .
    .
    .
    IRET

TIMER1_INT:
    AND    T1CON,#11111110B      ; timer 1 pending bit clear
    TM     MELODY_FLAG,#00000001B ; MELODY_FLAG.bit0 == 0 melody disable
    JP     NZ,MELODY_ENABLE      ; == 1 melody enable
    IRET

MELODY_ENABLE:
    DEC    MUSIC_INTERVAL
    CP     MUSIC_INTERVAL,#0     ; note duration check
    JP     EQ,MELODY_LOAD       ; next DB data load?
    IRET

MELODY_LOAD:
    ; melody interval reload
    LD     R15,BEEP_POINTER
    RL     R15                   ; BEEP_POINTER X 2
    CLR    R14
    LDC    R8,#MELODY_DB[RR14]   ; R8 <-- Note duration
    LDC    R9,#MELODY_DB+1[RR14] ; R9 <-- tone
    LD     MUSIC_INTERVAL,R8
    LD     BUZPS,R9              ; P1.1 buzzer signal on, clock selection
    INC    BEEP_POINTER
    CP     MUSIC_INTERVAL,#0FFH  ; Special code check
    JP     EQ,MELODY_OFF
    IRET

MELODY_OFF:
    AND    MELODY_FLAG,#11111110B ; melody signal disable
    LD     BUZPS,#0              ; Buzzer off
    IRET

MELODY_DB:
    ; DW wxyzH (wx:Note duration
    ; yz:Frequency,Tone)
    ; 4MHz OSC Clock
    ; 500ms interval
    DW    32BCH,32B6H,32AFH,32ADH ; C4 -- C5
    DW    32A8H,32A3H,32A0H,329EH
    DW    3200H
    ; 1000ms interval
    DW    649EH,649BH,6498H,6496H
    DW    6494H,6492H,64FFH,64FBH ; C5 -- C6
    DW    3200H
    ; 500ms interval
    DW    32FBH,32F5H,32F0H,32ECH
    DW    32E8H,32E4H,32E0H,32DFH ; C6 -- C7
    DW    0FF00H                 ; special code (melody end)
    .
    .

```

11

A/D CONVERTER

OVERVIEW

The A/D converter (ADC) module uses successive approximation logic to convert analog levels at one of the eight input channels (KS86C4104 has five input channels, ADC0–ADC4) 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:

- Eight multiplexed analog input pins (ADC0–ADC7)
- Analog comparator with successive approximation logic
- 8-bit A/D conversion data output registers (ADDATAH)
- ADC control register (ADCON)

An analog-to-digital conversion procedure is initiated when the CPU writes a value to the ADCON register at address F7H to select one of the eight available input pins. You select the desired input channel by setting the appropriate bits in the ADCON register.

The KS86C4004/4104 microcontroller performs 8-bit conversions for only one input channel at a time. You can dynamically select different analog input channels during program execution by manipulating selection bits in the ADCON register.

The A/D conversion process requires 4 steps (4 clock edges) to convert each bit and therefore requires 32 clocks to complete an 8-bit conversion: With a 8-MHz CPU clock frequency, one clock cycle is 125 ns. Therefore, if each bit requires 4 clocks, you calculate the conversion rate as follows:

$$125 \text{ ns} \times 4 \text{ clocks} \times 8 \text{ bits} + 8 \text{ clocks (setup time)} = 40 \text{ clocks (5 } \mu\text{s) at 8 MHz}$$

The digital result is dumped into the output registers, ADDATAH (F8H). The A/D converter unit then enters an idle state. Because the ADC module does not generate an interrupt to signal a completed conversion, the contents of ADDATAH must first be read out before another conversion starts. Otherwise, the previous result will be overwritten.

NOTE

Because the ADC does not use sample-and-hold circuitry, it is important that any fluctuations 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 circuit noise, will invalidate the result.

INTERNAL REFERENCE VOLTAGE LEVELS

In the ADC function block, the analog input voltage level is compared to the reference voltage. The analog input level must remain within the range AV_{SS} to AV_{REF} (usually, $AV_{REF} = V_{DD}$).

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 $1/2 AV_{REF}$.

USING A/D PINS FOR STANDARD DIGITAL INPUT

The ADC module's input pins are alternatively used as digital input in port 3 and port 2. The ADC0–ADC5 share pin names are P3.0–P3.5 and ADC6–ADC7 share pin names are P2.2–P2.3, respectively

A/D CONVERTER CONTROL REGISTER (ADCON)

The A/D converter control register, ADCON, is located at address F7H. Only bits 6–3 and 0 are used in the KS86C4004/4104 implementation. ADCON has three functions:

- Bits 6–4 select an analog input pin (ADC0–ADC7).
- Bit 3 indicates the status of the A/D conversion.
- Bit 0 starts the A/D conversion.

Only one analog input channel can be selected at a time. You can dynamically select any one of the eight analog input pins (ADC0–ADC7) by manipulating the 3-bit value for ADCON.6–ADCON.4

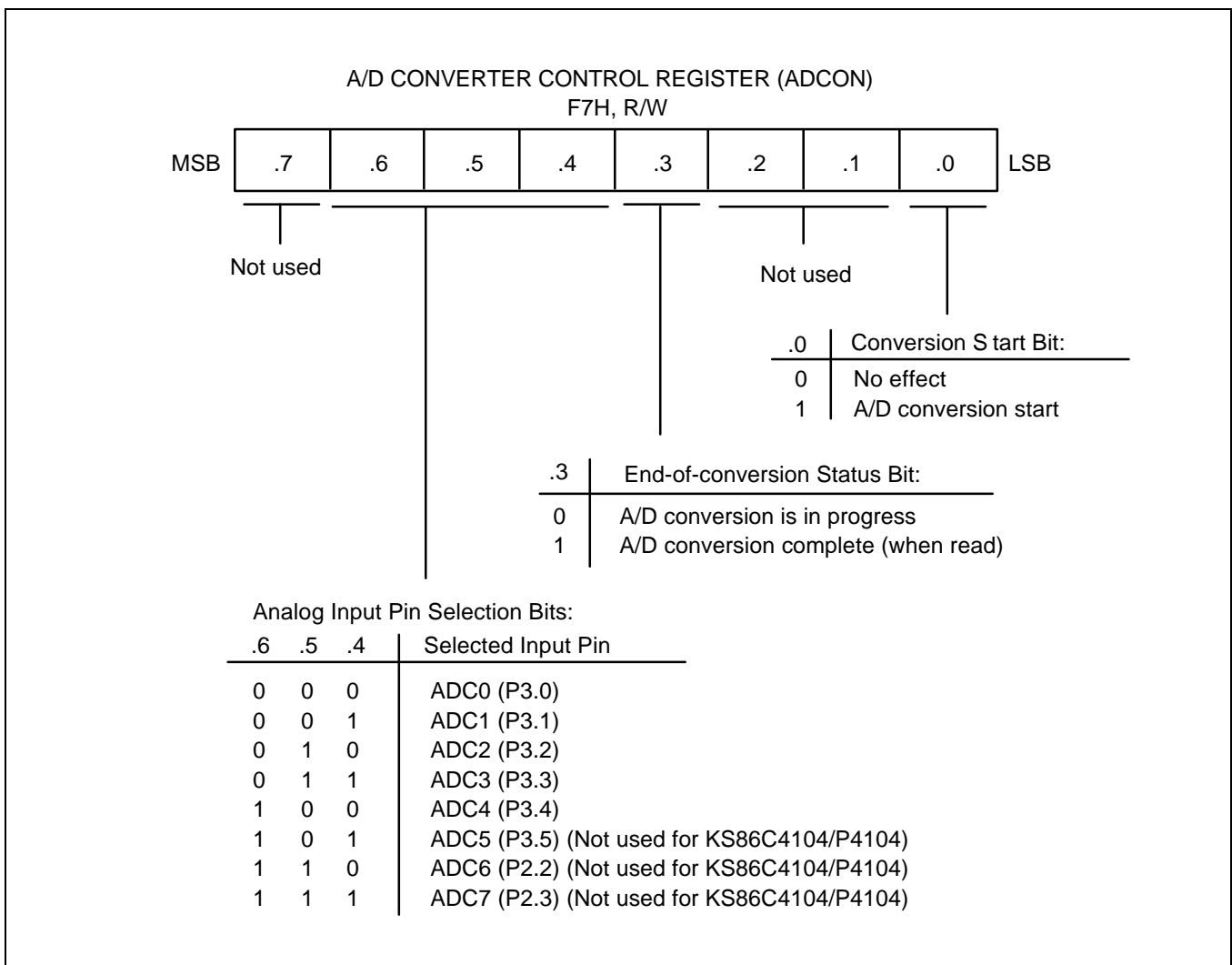


Figure 11-1. A/D Converter Control Register (ADCON)

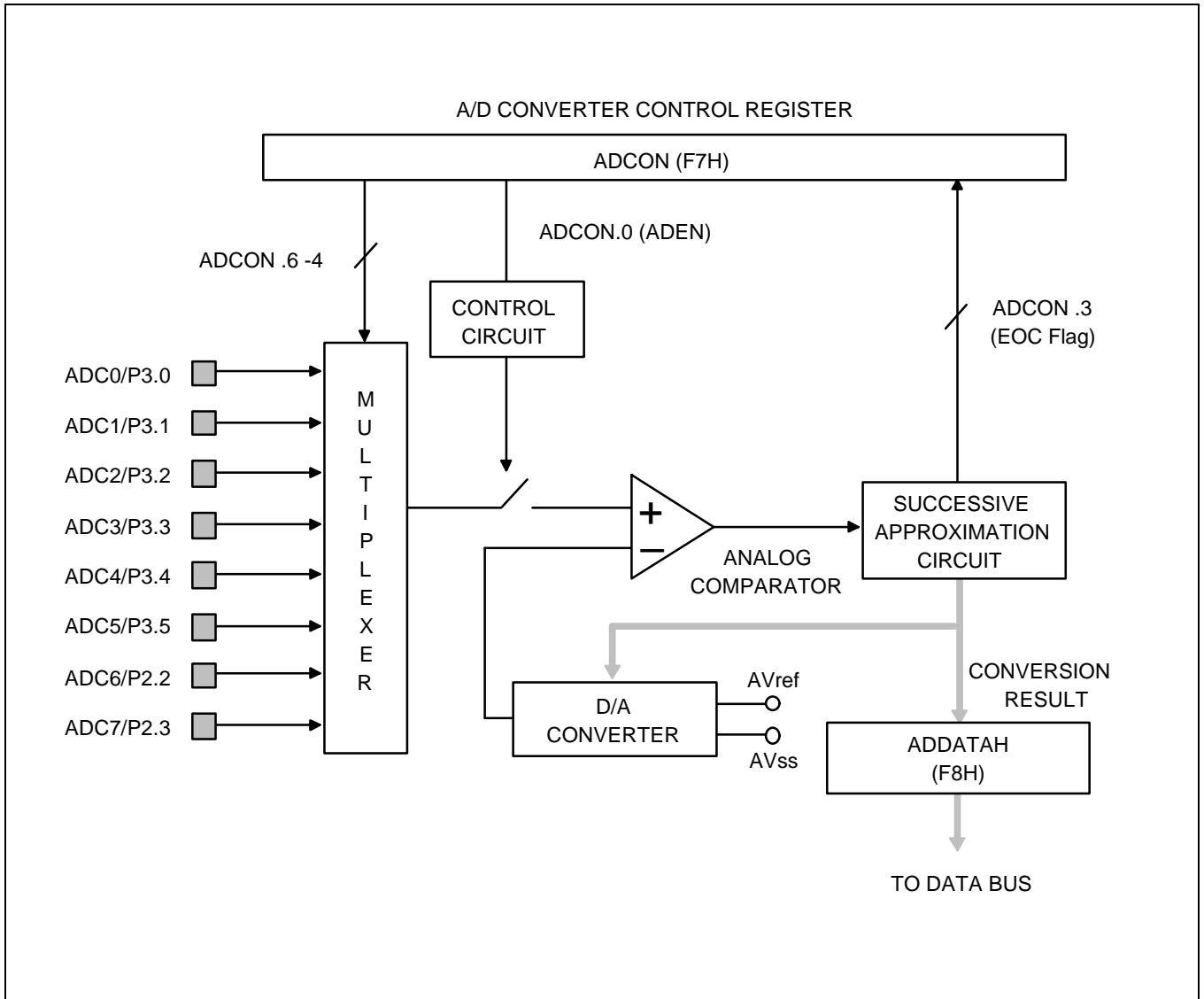


Figure 11-2. A/D Converter Circuit Diagram

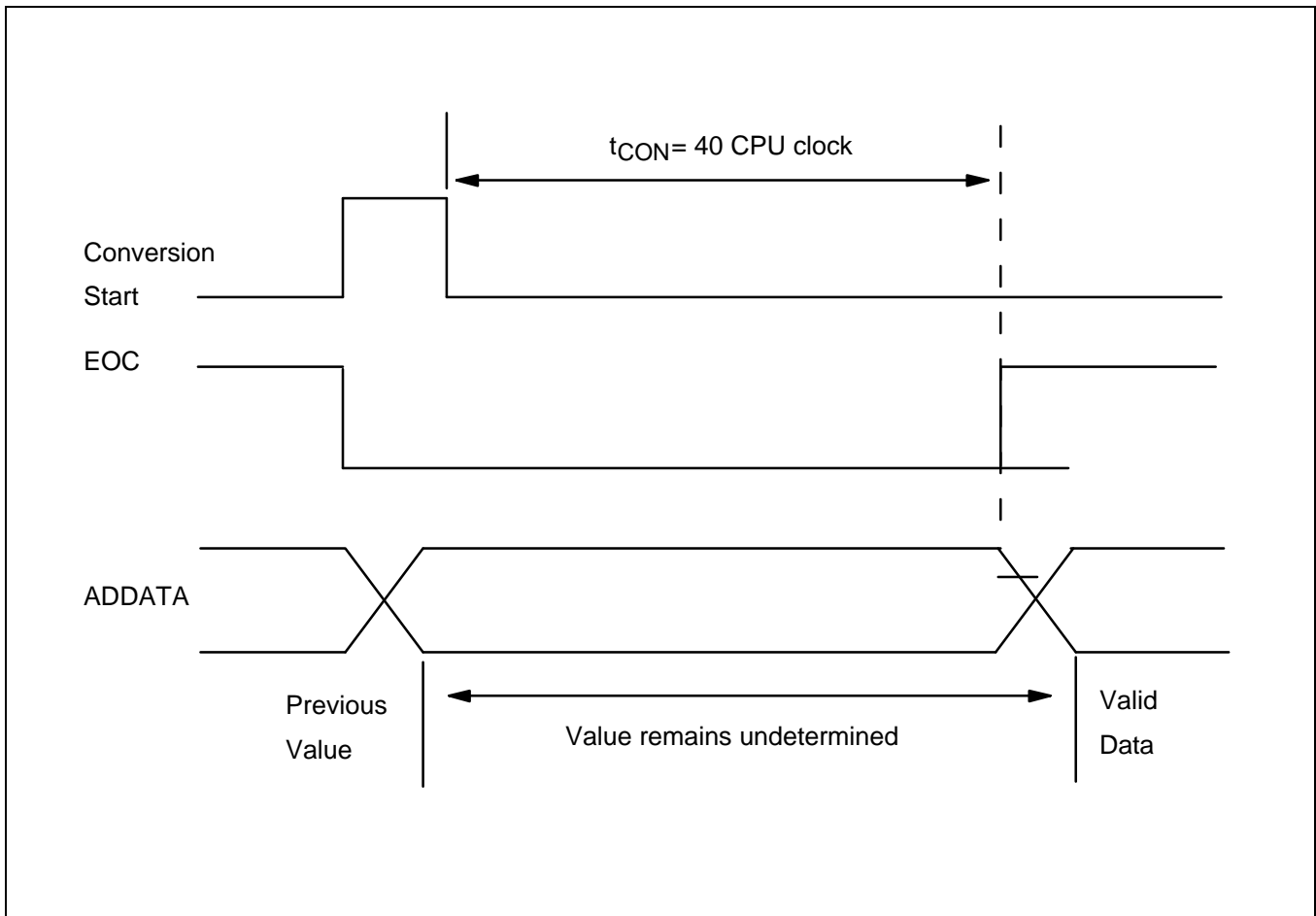


Figure 11-3. A/D Converter Timing Diagram

INTERNAL A/D CONVERSION PROCEDURE

1. Analog input must remain between the voltage range of AV_{SS} and AV_{REF} .
2. Configure the analog input pins to input mode by making the appropriate settings in P3CONH, P3CONL and P2CON registers.
3. Before the conversion operation starts, you must first select one of the eight input pins (ADC0–ADC7) by writing the appropriate value to the ADCON register.
4. When conversion has been completed, (40 CPU clocks have elapsed), the EOC flag is set to “1”, so that a check can be made to verify that the conversion was successful.
5. The converted digital value is loaded to the output register, ADDATAH, than the ADC module enters an idle state.
6. The digital conversion result can now be read from the ADDATAH register.

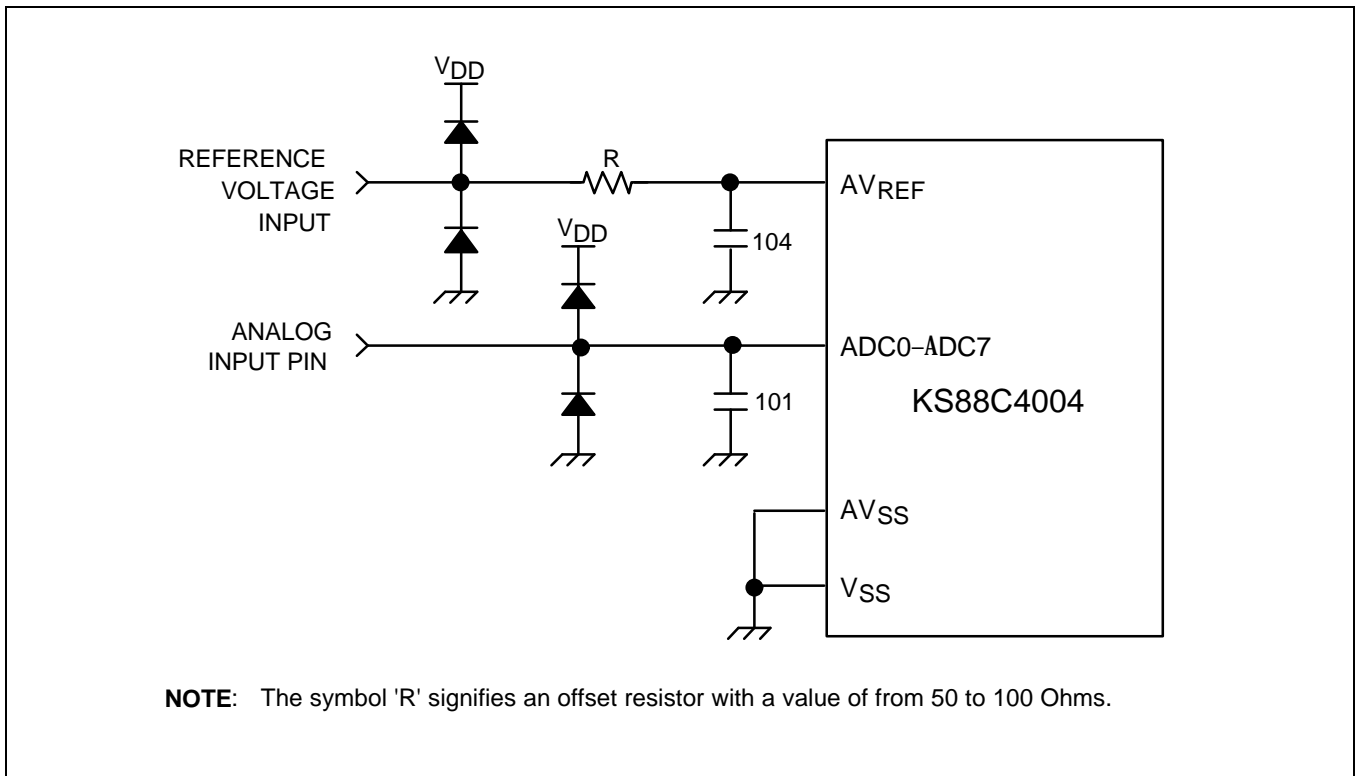


Figure 11-4. Recommended A/D Converter Circuit for Highest Absolute Accuracy

PROGRAMMING TIP PROGRAMMING TIP – Configuring 8-bit A/D Converter

```

.
.
.
LD      P3CONL,#00001111B      ; P3.1-0 A/D Input MODE
LD      P2CON,#11110000B      ; P2.3-2 A/D Input MODE
LD      P2DPUR,#00000000B     ; P2 PULL-UP Disable
.
.
.
LD      ADCON,#00000001B      ; channel ADC0 : P3.0 / conversion start
AD0_CHK: TM      ADCON,#00001000B ; A/D conversion end ? --> EOC check
JR      Z,AD0_CHK              ; no
LD      AD0BUF,ADDATAH        ; 8-bit Conversion data
.
.
.
LD      ADCON,#01100001B      ; channel ADC6 : P2.2 / Conversion start
AD6_CHK: TM      ADCON,#00001000B ; A/D conversion end ? --> EOC check
JR      Z,AD6_CHK              ; no
LD      AD6BUF,ADDATAH        ; 8-bit Conversion data
.
.
.

```

NOTES

12

ZERO-CROSSING DETECTION CIRCUIT

OVERVIEW

Zero-crossing detection circuit in Samsung's KS86C4004/C4104, generates a digital signal in synchronism with an AC signal input. It provides the timing signal for operations which are synchronized with the AC line. The zero crossing detection circuit digitizes the AC signal it receives from the power supply.

By setting bits 1 and 0 in port 1 control register (P1CON), you can enable zero-crossing detection. Zero-crossing detector is shown in Figure 12-1.

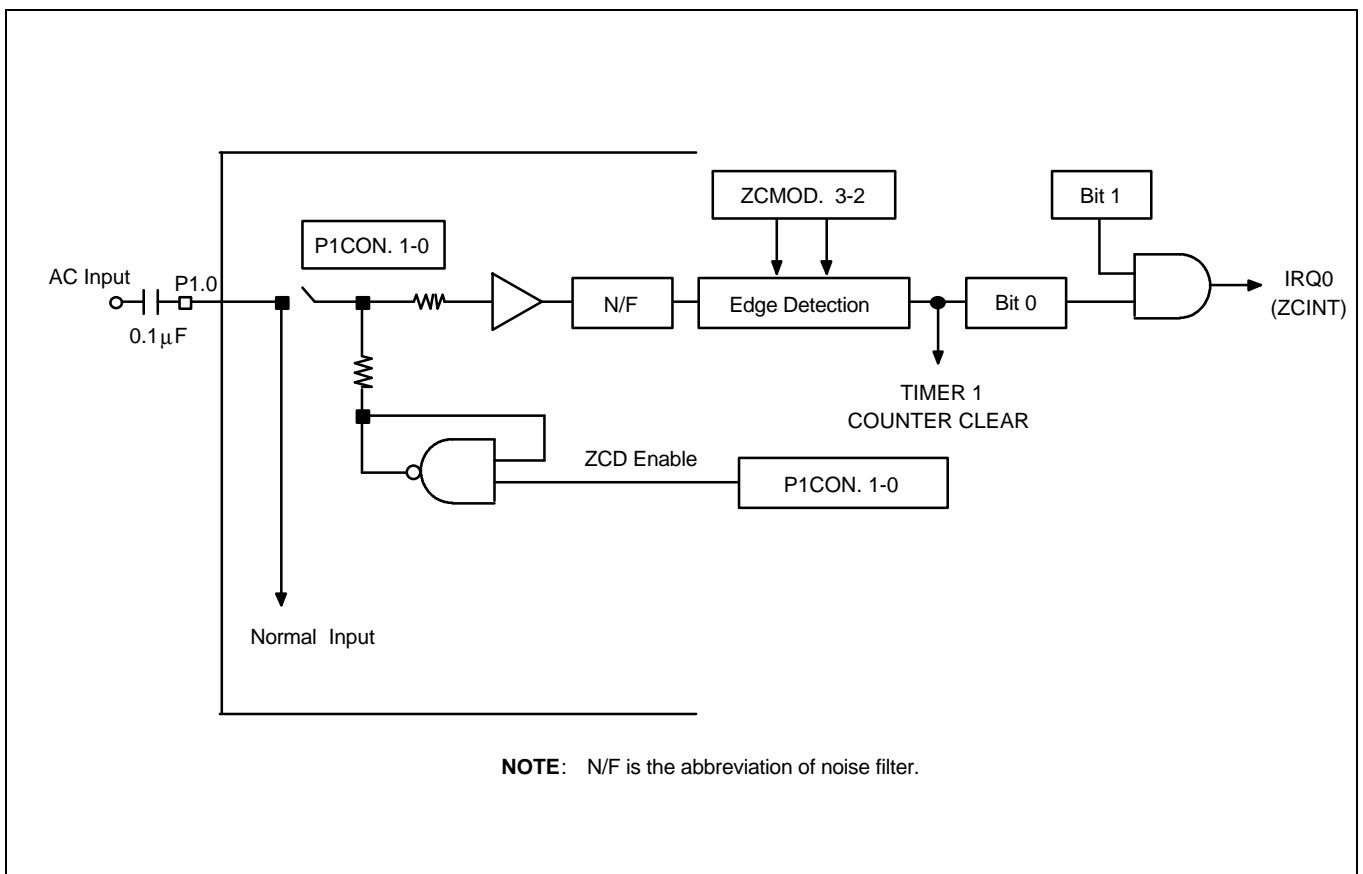


Figure 12-1. Zero-Crossing Detector Diagram

ZERO-CROSSING DETECTOR CONTROL REGISTER

The zero crossing detector control register, ZCMOD, is used to select interrupt mode (interrupt on falling edge, rising edge or both).

Reset clears ZCMOD to '00H', and configures interrupt selection mode to falling edge and disables ZCD interrupt. The interrupt pending bit must be cleared by writing "0" to ZCMOD.0

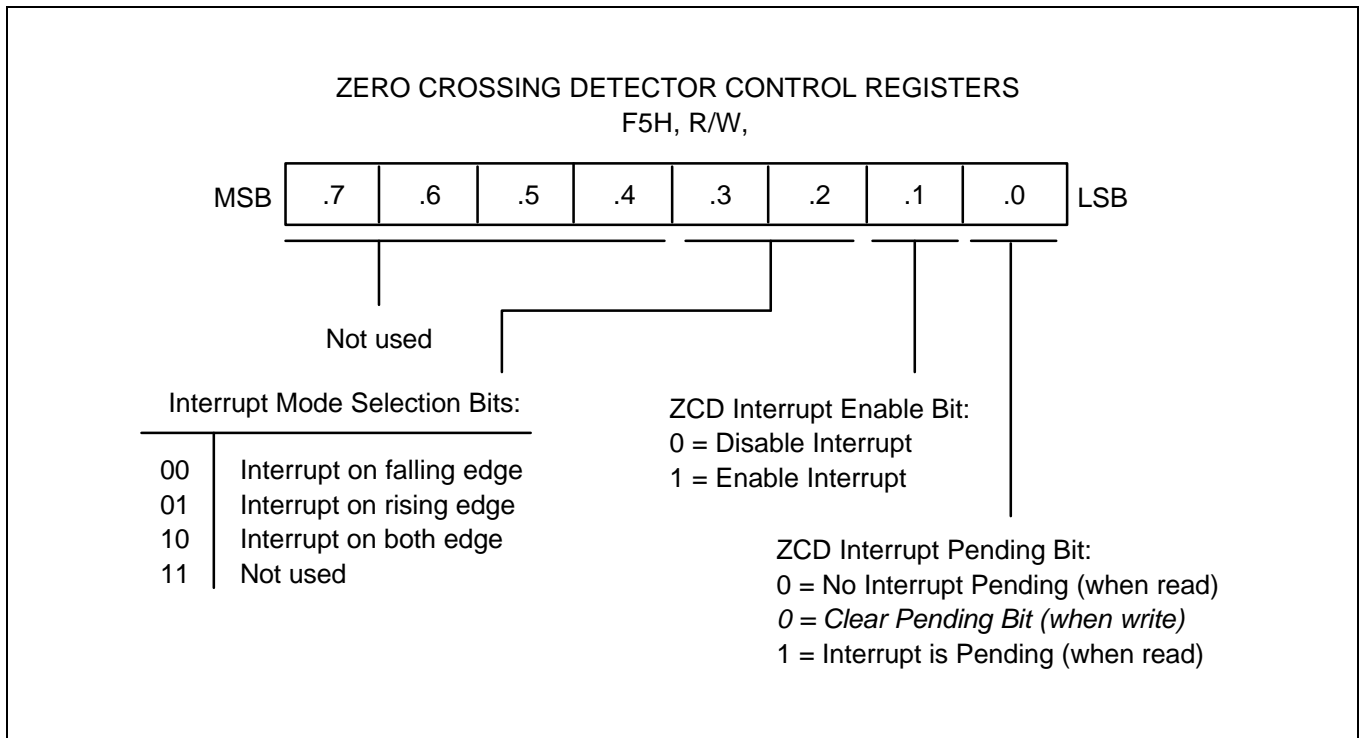


Figure 12-2. Zero-Crossing Detector Control Register (ZCMOD)

ZERO CROSS DETECTOR

ZCD circuit detects the zero-cross point of the AC waveform. Three types of detection can be selected, the point from positive to negative, the point from negative to positive, and both.

The zero cross detection circuit has the noise filter circuit in it. The detected zero cross point can be used to clear the timer 1 counter (T1CON.3=1).

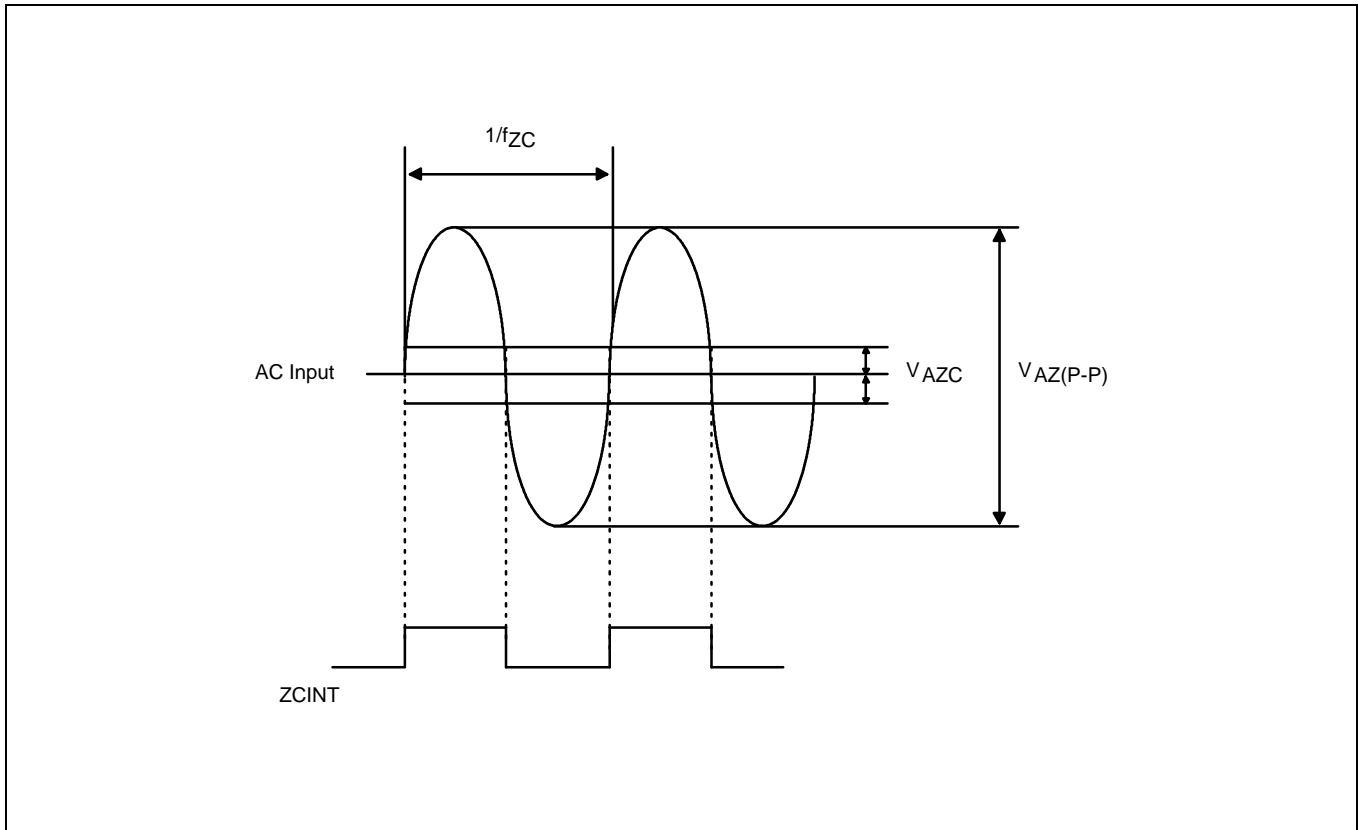



Figure 12-2. Zero-Crossing Waveform Diagram

 **PROGRAMMING TIP – Configuring ZCD and Timer1**

Programming procedure

1. Configure port 1 input pin to alternative function mode by making the appropriate bit setting to P1CON.
2. Select the correct clock source for timer 1 and set bit 3 (T1CON.3=1) to enable the ZCD signal to clear the timer 1. Load the proper data to timer 1 data register.
3. Enable the timer 1 interrupt in ZCD interrupt.
4. Set the relay on signal in timer 1 interrupt and disable the timer 1 interrupt.

```

                .ORG      0000H
                .VECTOR   00H, COMMON_INT          ; IRQ0 / Interrupt vector address
                .ORG      0100H                    ; Reset start address
RESET:         DI                          ; Disable interrupt
                LD        BTCON, #00000010B       ; Enable watch-dog function
                                                ; clock source:fosc/4096
                                                ; (104ms overflow at 10MHz)
                LD        CLKCON, #00011000B      ; CPU clock source select(non-divided)
                LD        SP, #0C0H               ; KS86C4004 Stack pointer initial
                LD        P1CON, #1010101011B    ; P1.0 ZCD input enable / P1.1-3 push-pull
                LD        ZCMOD, #00001010B      ; Enable both edge interrupt
                LD        T1DATA, #81H
                LD        T1CON, #00001100B      ; ZCD clear enable (fosc/512)
                                                ; Timer1 interrupt disable
                .
                .
                EI                          ; Enable interrupt
                .
MAIN:          .
                LD        BTCON, #02H            ; Enable watch-dog function
                                                ; Basic counter(BTCNT) clear
                .
                .
                JP        T, MAIN                ; For main loop
                .
                .
                .
COMMON_INT:   TM        ZCMOD, #00000001B
                JP        NZ, ZCD_INT
                TM        T1CON, #00000001B
                JP        NZ, TIMER1_INT
                .
                .
                .
    
```

```
TIMER1_INT:
    AND    T1CON, #11111100B    ; timer1 pending bit clear / t1 int disable
    XOR    P1, #00000100B      ; P1.2 toggle
    IRET
ZCD_INT:
    AND    ZCMOD, #11111110B   ; pending bit clear
    XOR    P1, #00001000B      ; P1.3 toggle
    LD     T1CON, #00001110B   ; Timer1 interrupt enable (fosc/512)
                                           ; Enable ZCD clear signal to clear the
                                           ; timer 1 counter
    IRET
:
:
```


NOTES

13

ELECTRICAL DATA

OVERVIEW

In this section, the following KS86C4004/C4104 electrical characteristics are presented in tables and graphs:

- Absolute maximum ratings
- D.C. electrical characteristics
- A.C. electrical characteristics
- Input timing measurement points
- Oscillator characteristics
- Oscillation stabilization time
- Data retention supply voltage in Stop mode
- Stop mode release timing when initiated by a reset
- A/D converter electrical characteristics
- Zero-crossing detector
- Schmitt trigger input characteristics
- Characteristic curves

Table 13-1. Absolute Maximum Ratings

 $(T_A = 25^\circ\text{C})$

Parameter	Symbol	Conditions	Rating	Unit
Supply voltage	V_{DD}	–	–0.3 to +6.5	V
Input voltage	V_I	All input ports	–0.3 to $V_{DD} + 0.3$	V
Output voltage	V_O	All output ports	–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 1, 2, 3	+100	
		Total pin current for ports 0	+200	
Operating temperature	T_A	–	–40 to +85	$^\circ\text{C}$
Storage temperature	T_{STG}	–	–65 to +150	$^\circ\text{C}$

Table 13-2. DC Electrical Characteristics

 $(T_A = -40^\circ\text{C}$ to $+85^\circ\text{C}$, $V_{DD} = 2.7\text{ V}$ to 5.5 V)

Parameter	Symbol	Conditions		Min	Typ	Max	Unit
Input high voltage	V_{IH1}	Ports 1,2,3 and RESET	$V_{DD} = 2.7$ to 5.5 V	$0.8 V_{DD}$	–	V_{DD}	V
	V_{IH2}	Port 0		$0.7 V_{DD}$			
	V_{IH3}	X_{IN} and X_{OUT}		$V_{DD} - 0.5$			
Input low voltage	V_{IL1}	Ports 1,2,3 and RESET	$V_{DD} = 2.7$ to 5.5 V	–	–	$0.2 V_{DD}$	V
	V_{IL2}	Port 0				$0.3 V_{DD}$	
	V_{IL3}	X_{IN} and X_{OUT}				0.4	
Output high voltage	V_{OH}	$I_{OH} = -1\text{ mA}$ ports 0, 1, 2, 3	$V_{DD} = 4.5$ to 5.5 V	$V_{DD} - 1.0$	–	–	V
Output low voltage	V_{OL1}	$I_{OL} = 15\text{ mA}$ port 0	$V_{DD} = 4.5$ to 5.5 V	–	0.4	2.0	V
	V_{OL2}	$I_{OL} = 4\text{ mA}$ port 1,2,3	$V_{DD} = 4.5$ to 5.5 V		0.4	2.0	

Table 13-2. DC Electrical Characteristics (Continued)

(T_A = -40°C to +85°C, V_{DD} = 2.7 V to 5.5 V)

Parameter	Symbol	Conditions		Min	Typ	Max	Unit		
Input high leakage current	I _{LIH1}	All inputs except I _{LIH2}	V _{IN} = V _{DD}	-	-	1	μA		
	I _{LIH2}	X _{IN} , X _{OUT}	V _{IN} = V _{DD}			5			
Input low leakage current	I _{LIL1}	All inputs except I _{LIL2}	V _{IN} = 0 V	-	-	-1	μA		
	I _{LIL2}	X _{IN} , X _{OUT}	V _{IN} = 0 V			-5			
Output high leakage current	I _{LOH}	All outputs	V _{OUT} = V _{DD}	-	-	2	μA		
Output low leakage current	I _{LOL}	All outputs	V _{OUT} = 0 V	-	-	-2	μA		
Pull-up resistors	R _P	V _{IN} = 0 V Ports 0 – 3 and RESET	V _{DD} = 5 V	30	47	70	kΩ		
			V _{DD} = 3 V	30	280	350			
Supply current	I _{DD1}	Run mode 10 MHz CPU clock	V _{DD} = 5 V ± 10%	-	7.5	15	mA		
		8 MHz CPU clock	V _{DD} = 3 V ± 10%			3		6	
	I _{DD2}	Idle mode 10 MHz CPU clock	V _{DD} = 5 V ± 10%			2		5	
		8 MHz CPU clock	V _{DD} = 3 V ± 10%			0.7		2.5	
	I _{DD3}	Stop mode	V _{DD} = 5 V ± 10%			0.1		5	μA
			V _{DD} = 3 V ± 10%						

NOTE: D.C. electrical values for Supply current (I_{DD1} to I_{DD3}) do not include current drawn through internal pull-up resistors, output port drive current, ZCD and ADC.

Table 13-3. AC Electrical Characteristics

(T_A = -20°C to +85°C, V_{DD} = 4.5 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Interrupt input high, low width	t _{INTH} , t _{INTL}	Port 2 V _{DD} = 5V ± 10%	–	200	–	ns
RESET input low width ZCD noise filter	t _{RSL} –	Input V _{DD} = 5V ± 10%	–	1	–	μs

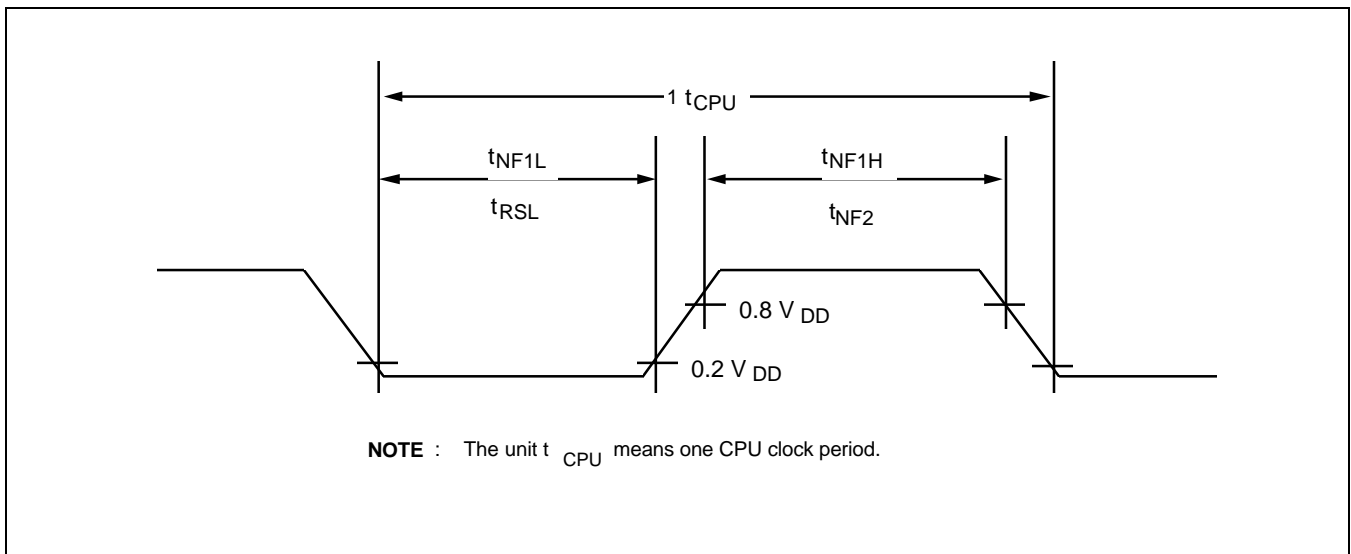


Figure 13-1. Input Timing Measurement Points

Table 13-4. Oscillator Characteristics

(T_A = -40°C to +85°C)

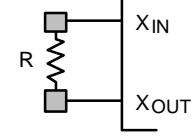
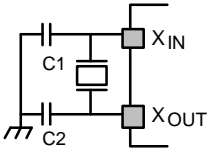
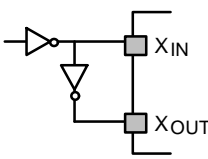
Oscillator	Clock Circuit	Test Condition	Min	Typ	Max	Unit
RC oscillator		V _{DD} = 4.75 to 5.25 V	-	4	-	MHz
Main crystal or ceramic		V _{DD} = 4.5 to 5.5 V V _{DD} = 2.7 to 4.5 V	1 1	- -	10 8	
External clock		V _{DD} = 4.5 to 5.5 V V _{DD} = 2.7 to 4.5 V	1 1	- -	10 8	

Table 13-5. Oscillation Stabilization Time

(T_A = -40°C to +85°C, V_{DD} = 2.7 V to 5.5 V)

Oscillator	Test Condition	Min	Typ	Max	Unit
Main crystal	f _{OSC} > 1.0 MHz	-	-	20	ms
Main ceramic	Oscillation stabilization occurs when V _{DD} is equal to the minimum oscillator voltage range.	-	-	10	
External clock (main system)	X _{IN} input high and low width (t _{XH} , t _{XL})	25	-	500	ns
Oscillator stabilization	t _{WAIT} when released by a reset ⁽¹⁾	-	2 ¹⁶ / f _{OSC}	-	ms
wait time	t _{WAIT} when released by an interrupt ⁽²⁾	-	-	-	

NOTES

- f_{OSC} is the oscillator frequency.
- The duration of the oscillator stabilization wait time, t_{WAIT}, when it is released by an interrupt is determined by the settings in the basic timer control register, BTCON.

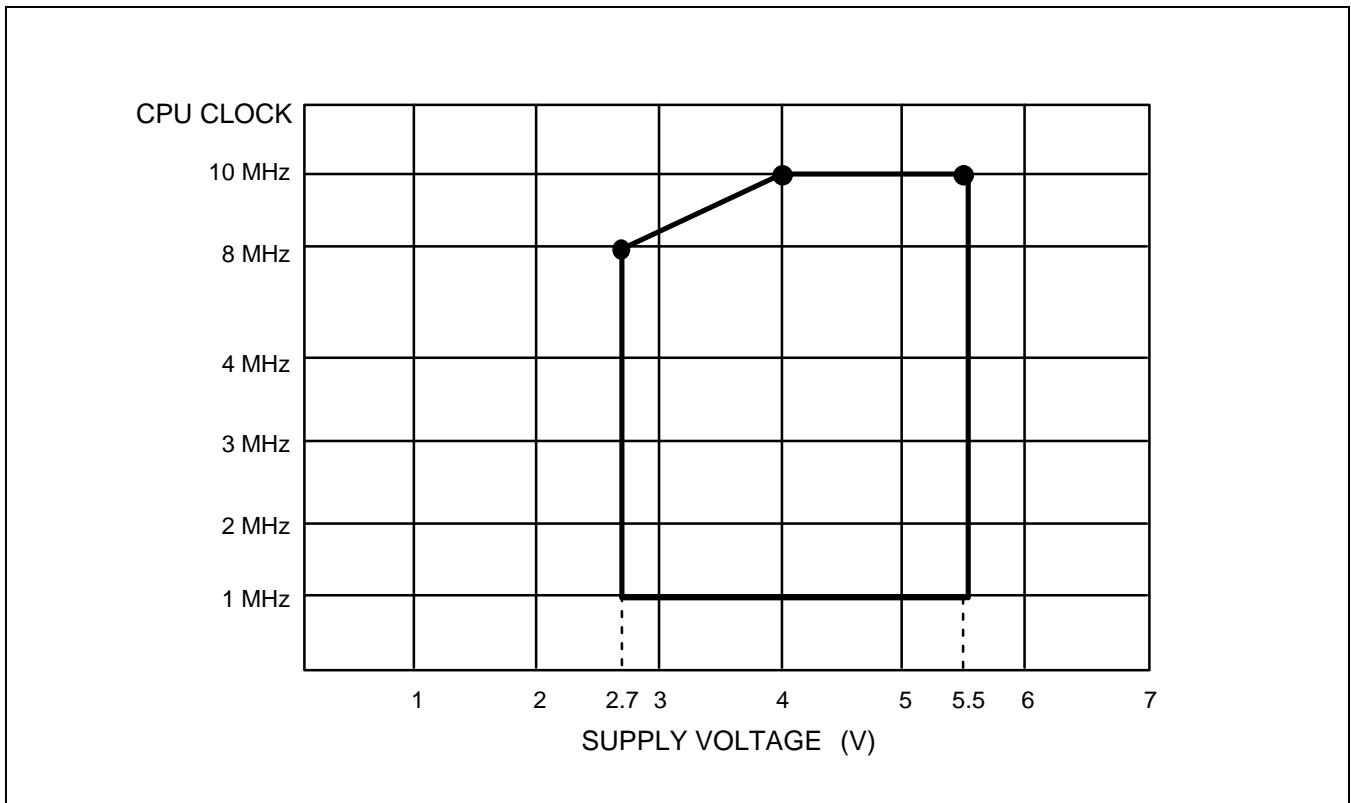


Figure 13-2. Operating Voltage Range (KS86C4104)

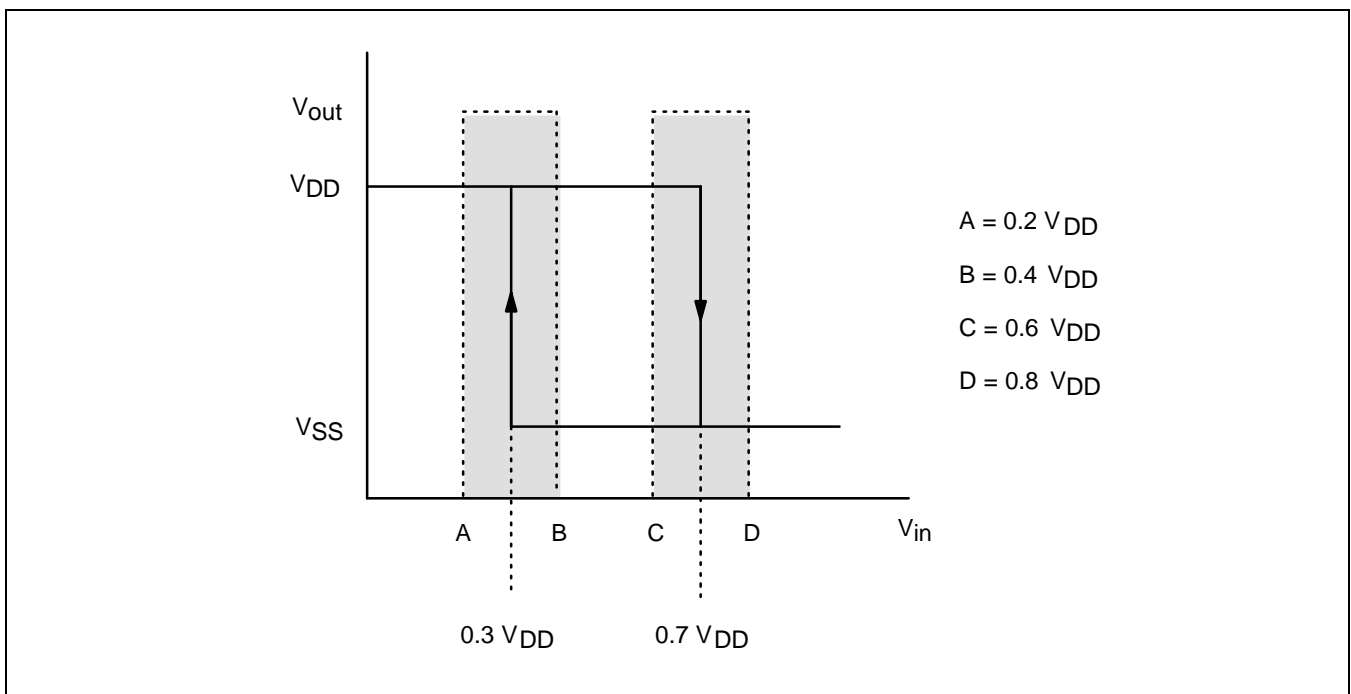


Figure 13-3. Schmitt Trigger Input Characteristics Diagram

Table 13-6. Data Retention Supply Voltage in Stop Mode

($T_A = -40^{\circ}\text{C}$ to $+85^{\circ}\text{C}$, $V_{DD} = 2.7\text{ V}$ to 5.5V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Data retention supply voltage	V_{DDDR}	Stop mode	2.0	–	5.5	V
Data retention supply current	I_{DDDR}	Stop mode; $V_{DDDR} = 2.0\text{ V}$	–	0.1	5	μA

NOTE: Supply current does not include current drawn through internal pull-up resistors or external output current loads.

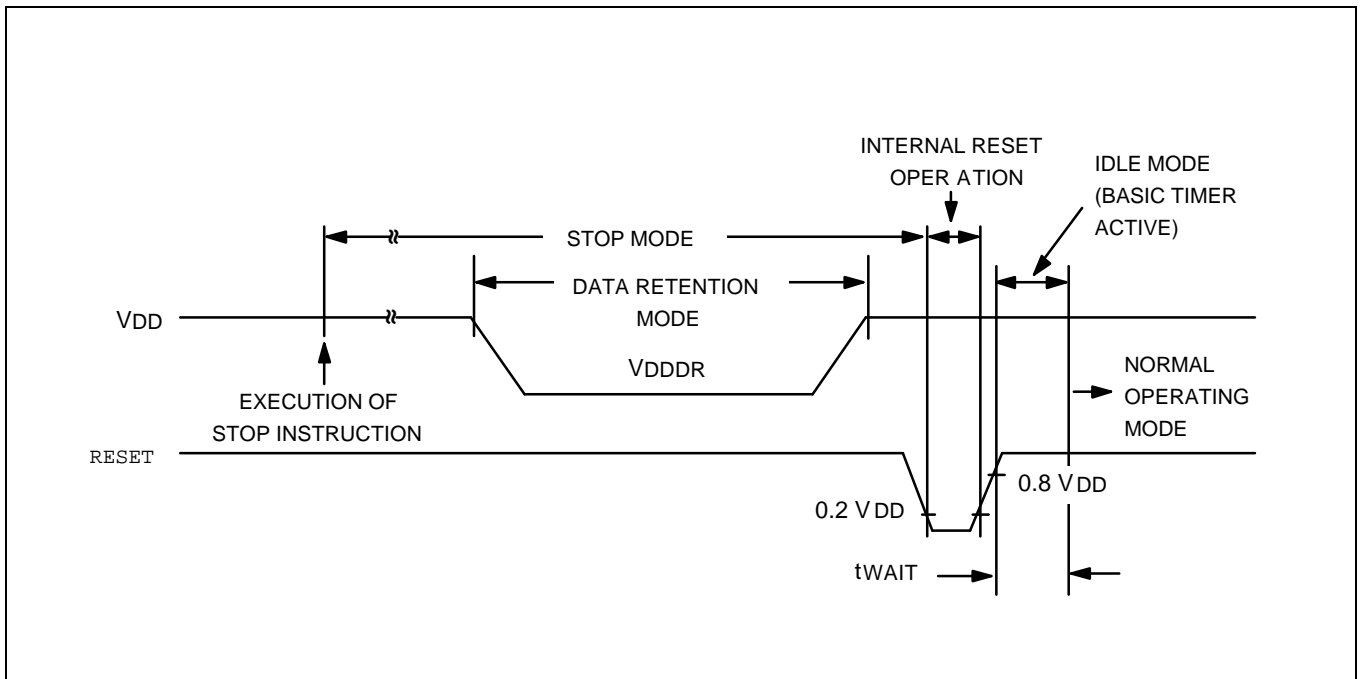


Figure 13-4. Stop Mode Release Timing When Initiated by a RESET

Table 13-7. A/D Converter Electrical Characteristics

(T_A = -40°C to +85°C, V_{DD} = 2.7 V to 5.5 V, V_{SS} = 0 V)

Parameter	Symbol	Test Conditions	Min	Typ	Max	Unit
Total accuracy		V _{DD} = 5.12 V CPU clock = 10 MHz AV _{REF} = 5.12 V AV _{SS} = 0 V	-	-	± 2	LSB
Integral linearity error	ILE	"	-	-	± 1.5	LSB
Differential linearity error	DLE	"	-	-	± 1	
Offset error of top	EOT	"	-	-1	±2	
Offset error of bottom	EOB	"	-	-1	± 2	
Conversion time ⁽¹⁾	t _{CON}	f _{cpu} = 10 MHz	4	-	-	μs
Analog input voltage	V _{IAN}	-	AV _{SS}	-	AV _{REF}	V
Analog input impedance	R _{AN}	-	2	-	-	MΩ
ADC reference voltage	AV _{REF}	-	3.0	-	V _{DD}	V
ADC reference ground	AV _{SS}	-	V _{SS}	-	V _{SS} + 0.3	V
Analog input current	I _{ADIN}	AV _{REF} = V _{DD} = 5 V	-	-	10	μA
ADC block current ⁽²⁾	I _{ADC}	AV _{REF} = V _{DD} = 5 V	-	1	3	mA
		AV _{REF} = V _{DD} = 3 V	-	0.5	1.5	
		AV _{REF} = V _{DD} = 5 V Power down mode	-	100	500	nA

NOTES:

- 'Conversion time' is the time required from the moment a conversion operation starts until it ends.
- I_{ADC} is operating current during A/D conversion.

Table 13-8. Zero Crossing Detector

($T_A = -40^{\circ}\text{C}$ to $+85^{\circ}\text{C}$, $V_{DD} = 4.5\text{ V}$ to 5.5 V , $V_{SS} = 0\text{ V}$)

Parameter	Symbol	Test Conditions	Min	Typ	Max	Unit
Zero-crossing detection input voltage	V_{ZC}	AC connection $c = 0.1\ \mu\text{F}$	1.0	–	3.0	Vp-p
Zero-crossing detection accuracy	V_{AZC}	$f_{ZC} = 60\text{ Hz}$ (sine wave) $V_{DD} = 5\text{ V}$ $f_{OSC} = 10\text{ MHz}$	–	–	± 100	mV
Zero-crossing detection input frequency	f_{ZC}	–	40	–	200	Hz

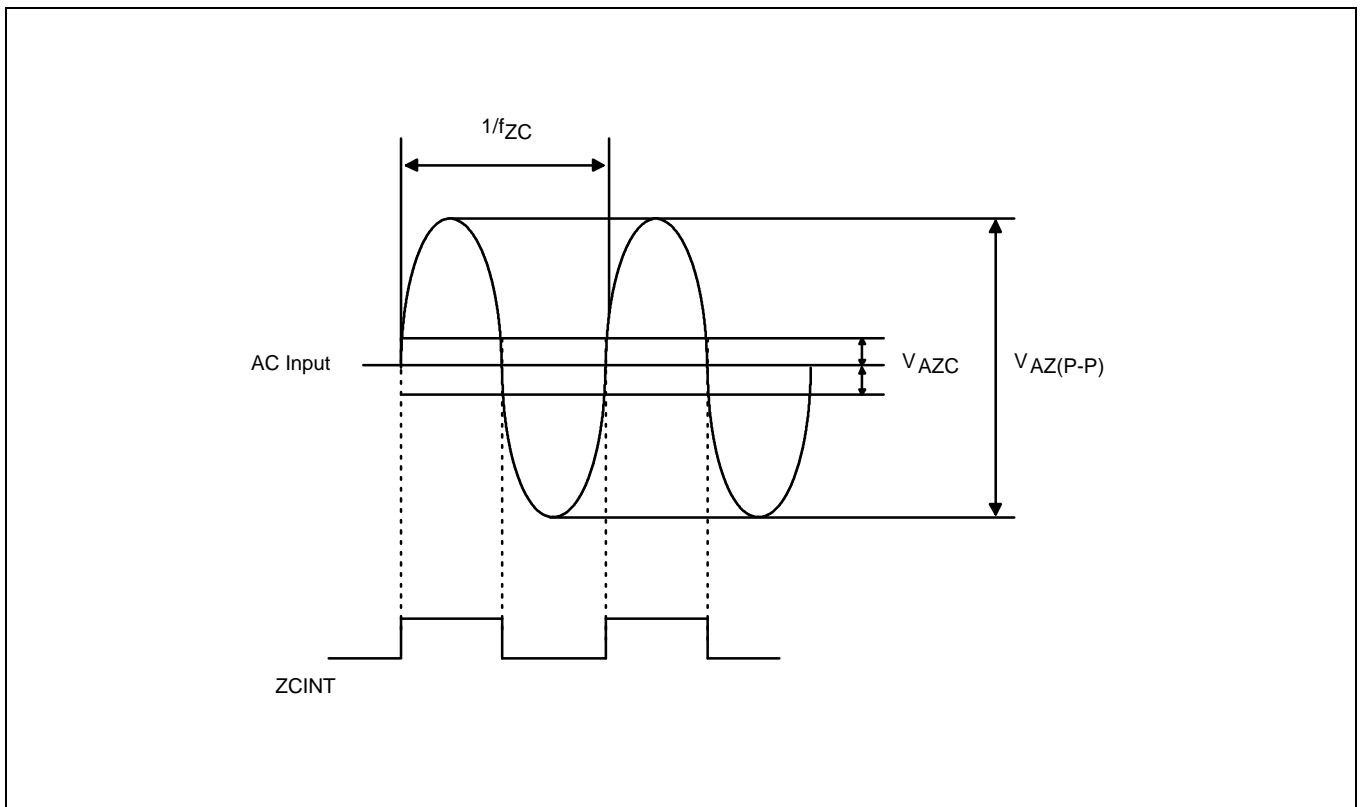


Figure 13-2. Operating Voltage Range

NOTES

15

KS86P4004/P4104 OTP

OVERVIEW

The KS86P4004/P4104 single-chip CMOS microcontroller is the OTP (One Time Programmable) version of the KS86C4004/C4104 microcontroller. It has an on-chip OTP ROM instead of masked ROM. The EPROM is accessed by serial data format.

The KS86P4004/P4104 is fully compatible with the KS86C4004/C4104, both in function and in pin configuration. Because of its simple programming requirements, the KS86P4004/P4104 is ideal for use as an evaluation chip for the KS86C4004/C4104.

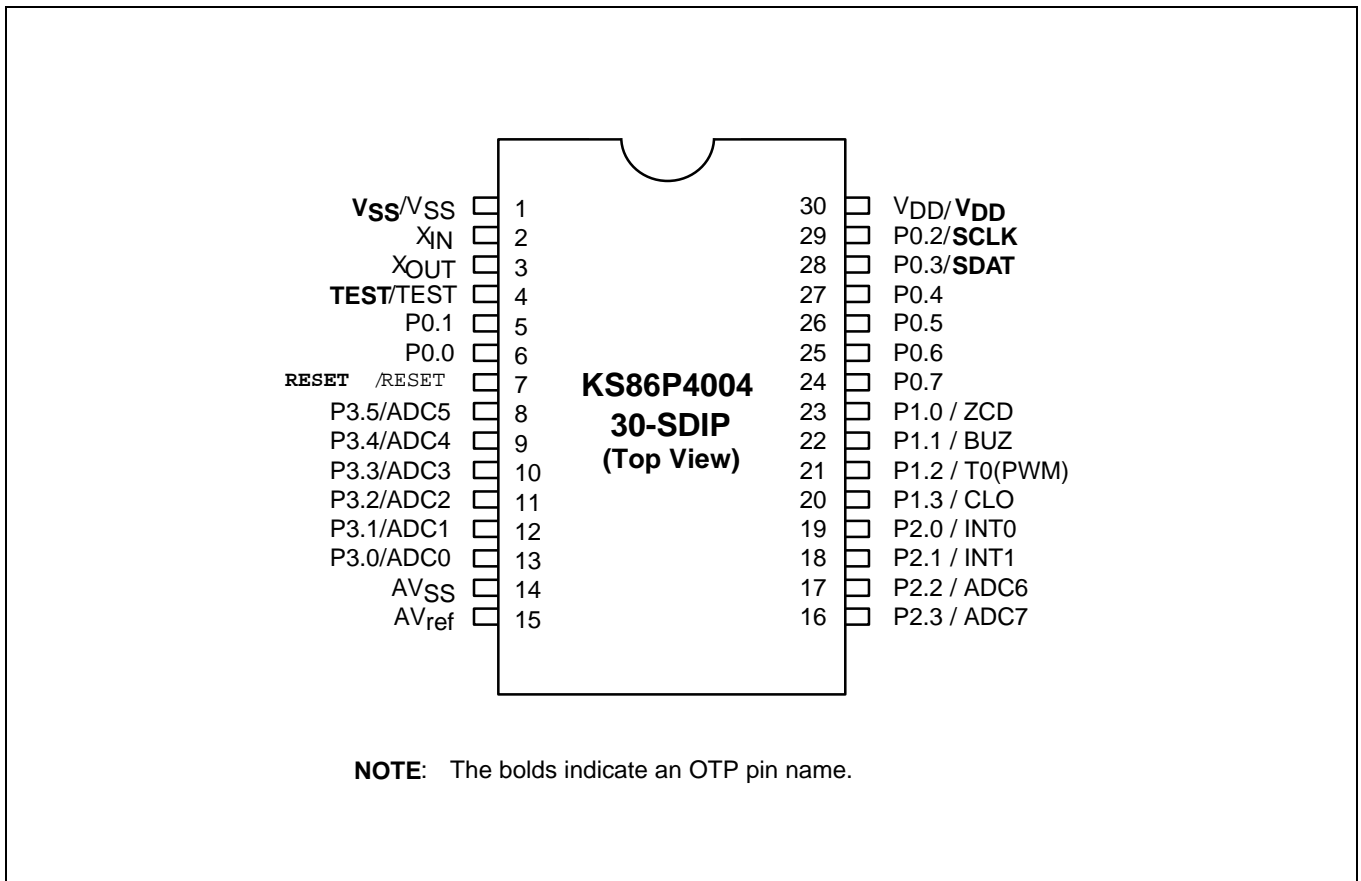


Figure 15-1. Pin Assignment Diagram (30-Pin SDIP Package)

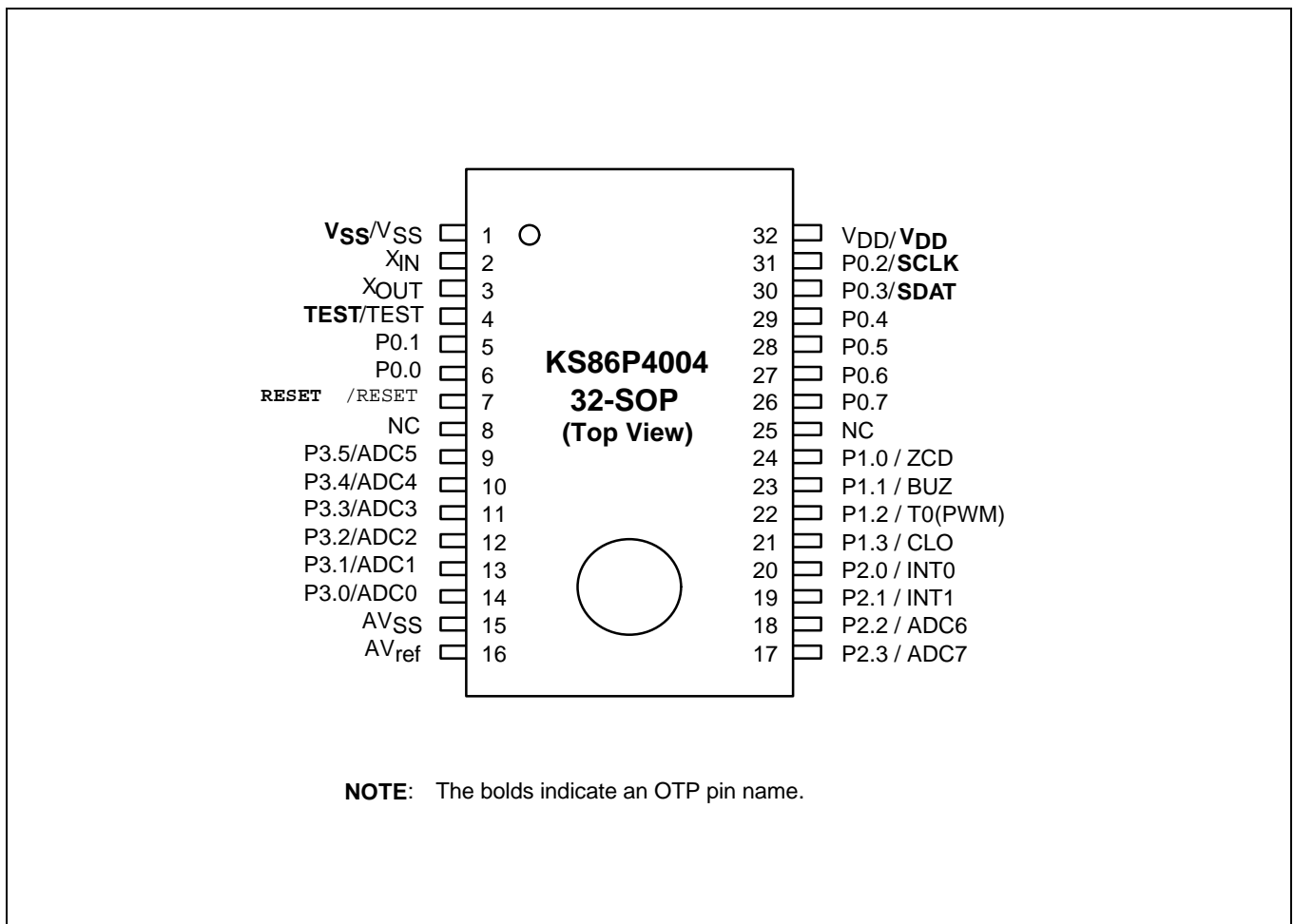


Figure 15-2. Pin Assignment Diagram (32-Pin SOP Package)

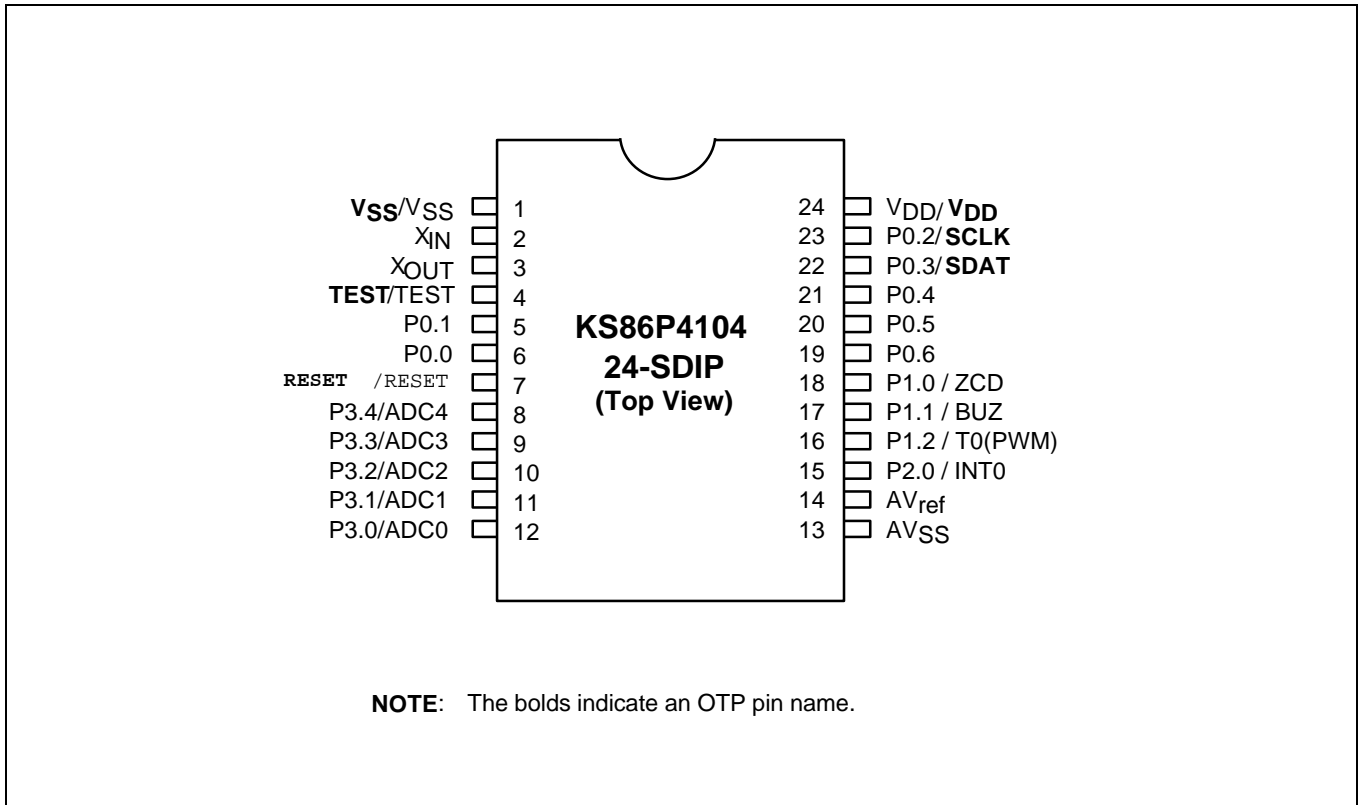


Figure 15-3. Pin Assignment Diagram (24-Pin SDIP Package)

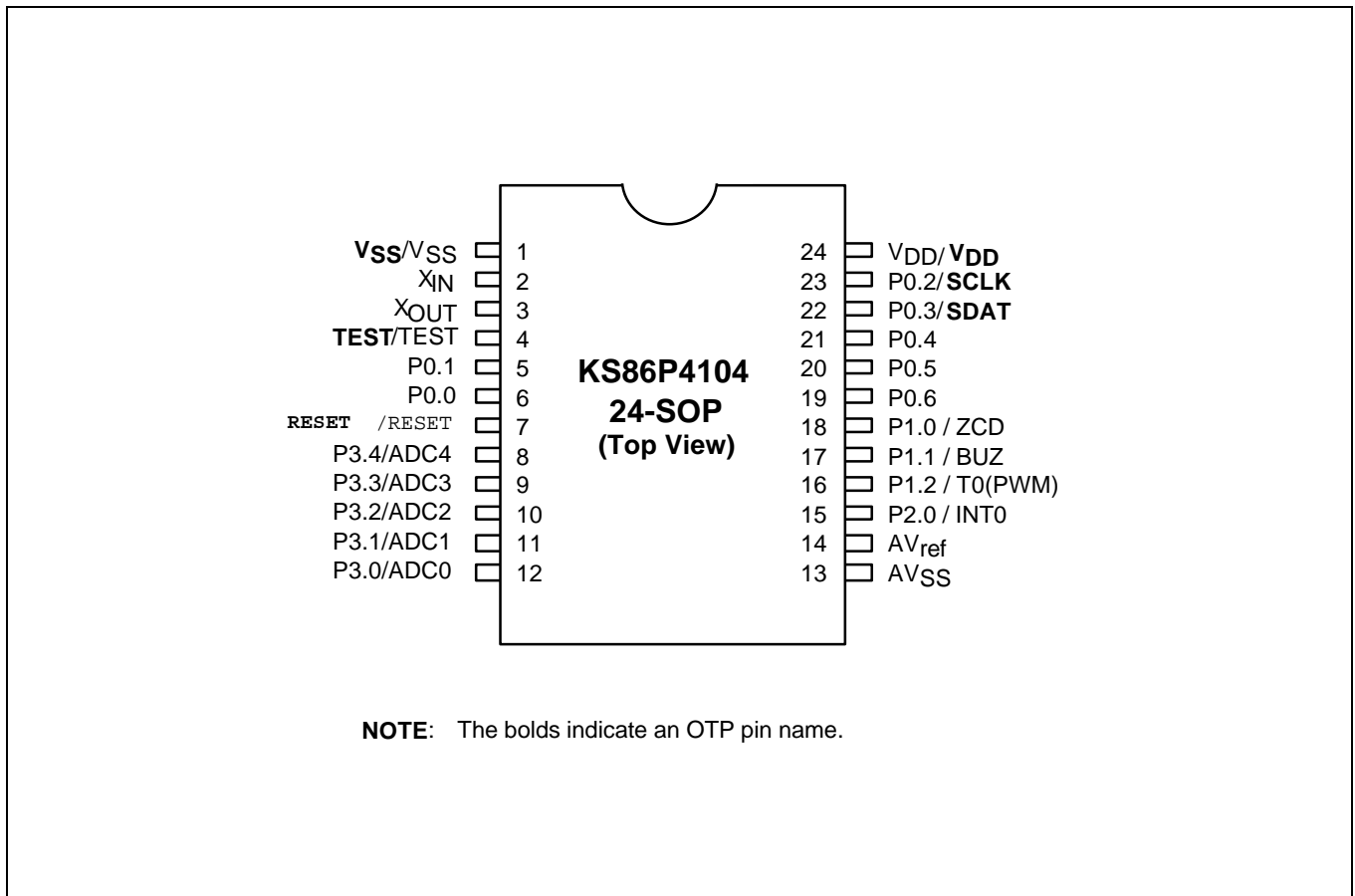


Figure 15-4. Pin Assignment Diagram (24-Pin SOP Package)

Table 15-1. Descriptions of Pins Used to Read/Write the EPROM

Main Chip Pin Name	During Programming			
	Pin Name	Pin No.	I/O	Function
P0.3	SDAT	KS86P4004: 28 (30) KS86P4104: 22 (22)	I/O	Serial data pin (output when reading, Input when writing) Input and push-pull output port can be assigned
P0.2	SCLK	KS86P4004: 29 (31) KS86P4104: 23 (23)	I/O	Serial clock pin (input only pin)
TEST	V _{PP} (TEST)	4	I	Power supply pin for EPROM cell writing (indicates that OTP enters into the writing mode). When 12.5 V is applied, OTP is in writing mode and when 5 V is applied, OTP is in reading mode. (Option)
RESET	RESET	7	I	Chip Initialization
V _{DD} / V _{SS}	V _{DD} / V _{SS}	KS86P4004: 30 (32) / 1 KS86P4104: 24 (24) / 1	I	Logic power supply pin.

NOTE: () means the SOP OTP pin number.

Table 15-2. Comparison of KS86P4004/P4104 and KS86C4004/C4104 Features

Characteristic	KS86P4004/P4104	KS86C4004/C4104
Program Memory	4 K byte EPROM	4 K byte mask ROM
Operating Voltage (V _{DD})	2.7 V to 5.5 V	2.7 V to 5.5 V
OTP Programming Mode	V _{DD} = 5 V, V _{PP} (TEST)=12.5V	
Pin Configuration	30 SDIP / 32 SOP / 24 SDIP / 24 SOP	
EPROM Programmability	User Program 1 time	Programmed at the factory

OPERATING MODE CHARACTERISTICS

When 12.5 V is supplied to the V_{PP} (TEST) pin of the KS86P4004/P4104, the EPROM programming mode is entered. The operating mode (read, write, or read protection) is selected according to the input signals to the pins listed in Table 15-3 below.

Table 15-3. Operating Mode Selection Criteria

V _{DD}	V _{PP} (RESET)	REG/ MEM	ADDRESS (A15-A0)	R/W	MODE
5 V	5 V	0	0000H	1	EPROM read
	12.5 V	0	0000H	0	EPROM program
	12.5 V	0	0000H	1	EPROM verify
	12.5 V	1	0E3FH	0	EPROM read protection

NOTE: "0" means Low level; "1" means High level.

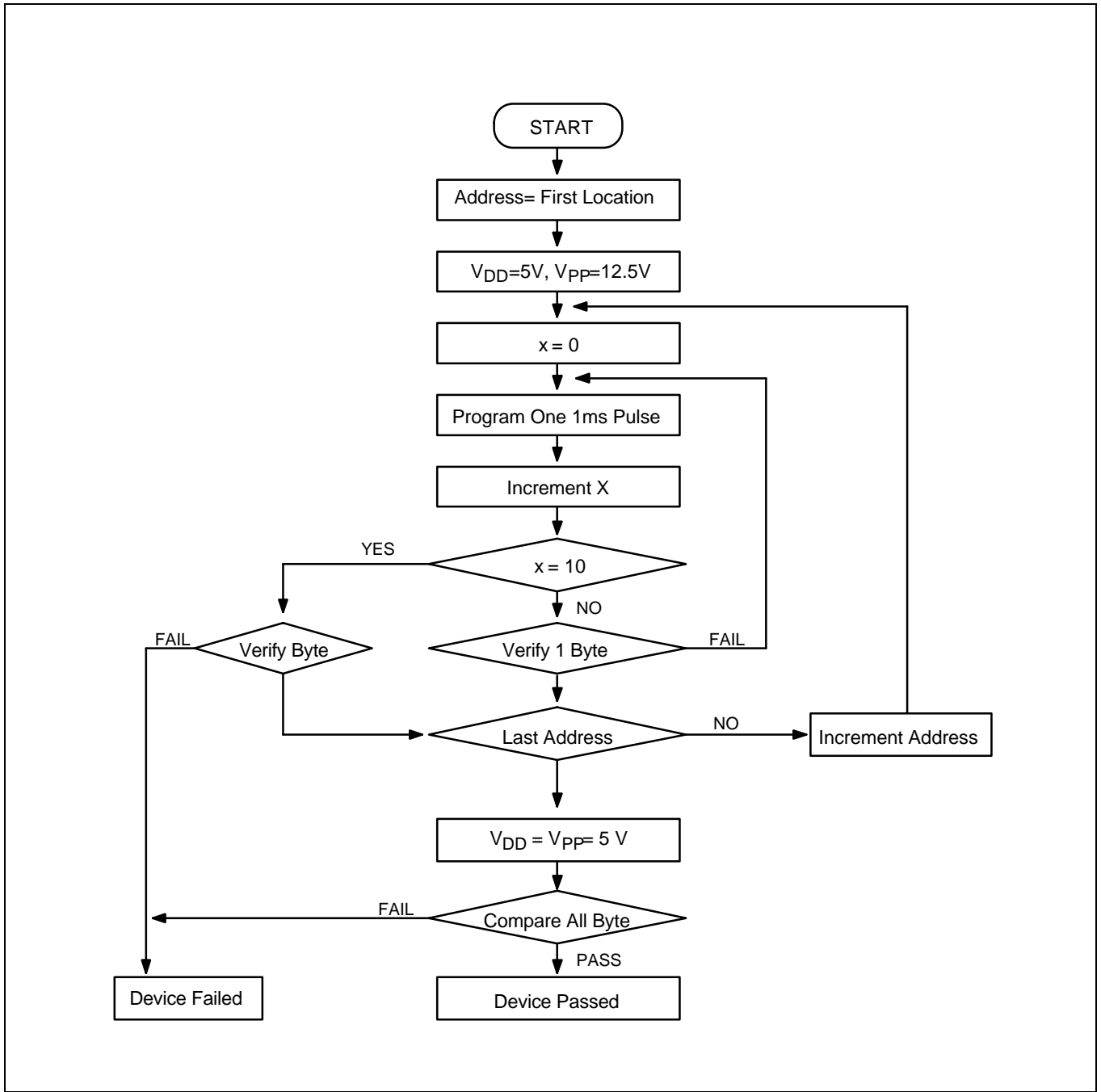


Figure 15-5. OTP Programming Algorithm

Table 15-4. D.C. Electrical Characteristics

(T_A = -20°C to +85°C, V_{DD} = 4.5 V to 5.5 V)

Parameter	Symbol	Conditions		Min	Typ	Max	Unit
Supply current	I _{DD1}	Run mode; 10 MHz CPU clock	V _{DD} = 5 V ± 10%	-	7.5	15	mA
		8 MHz CPU clock	V _{DD} = 3 V ± 10%		3	6	
	I _{DD2}	Idle mode; 10 MHz CPU clock	V _{DD} = 5 V ± 10%		2	5	
		8 MHz CPU clock	V _{DD} = 3 V ± 10%		0.7	2.5	
	I _{DD3}	Stop mode	V _{DD} = 5 V ± 10%		0.1	5	μA
			V _{DD} = 3 V ± 10%				

NOTE: D.C. electrical values for Supply current (I_{DD1} to I_{DD3}) do not include current drawn through internal pull-up resistors, output port drive current, ZCD and ADC.

NOTES



16

DEVELOPMENT TOOLS

OVERVIEW

Samsung provides a powerful and easy-to-use development support system in turnkey form. The development support system is configured with a host system, debugging tools, and support software. For the host system, any standard computer that operates with MS-DOS as its operating system can be used. One type of debugging tool including hardware and software is provided: the sophisticated and powerful in-circuit emulator, SMDS2+, for KS57, KS86, KS88 families of microcontrollers. The SMDS2+ is a new and improved version of SMDS2. Samsung also offers support software that includes debugger, assembler, and a program for setting options.

SHINE

Samsung Host Interface for in-circuit Emulator, SHINE, is a multi-window based debugger for SMDS2+. SHINE provides pull-down and pop-up menus, mouse support, function/hot keys, and context-sensitive hyper-linked help. It has an advanced, multiple-windowed user interface that emphasizes ease of use. Each window can be sized, moved, scrolled, highlighted, added, or removed completely.

SAMA ASSEMBLER

The Samsung Arrangeable Microcontroller (SAM) Assembler, SAMA, is a universal assembler, and generates object code in standard hexadecimal format. Assembled program code includes the object code that is used for ROM data and required SMDS program control data. To assemble programs, SAMA requires a source file and an auxiliary definition (DEF) file with device specific information.

SASM86

The SASM86 is an relocatable assembler for Samsung's KS86-series microcontrollers. The SASM86 takes a source file containing assembly language statements and translates into a corresponding source code, object code and comments. The SASM86 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 KS86-series microcontrollers. All required target system cables and adapters are included with the device-specific target board.

OTPs

One times programmable microcontrollers (OTPs) are under development for KS86C4004/C4104 microcontroller.

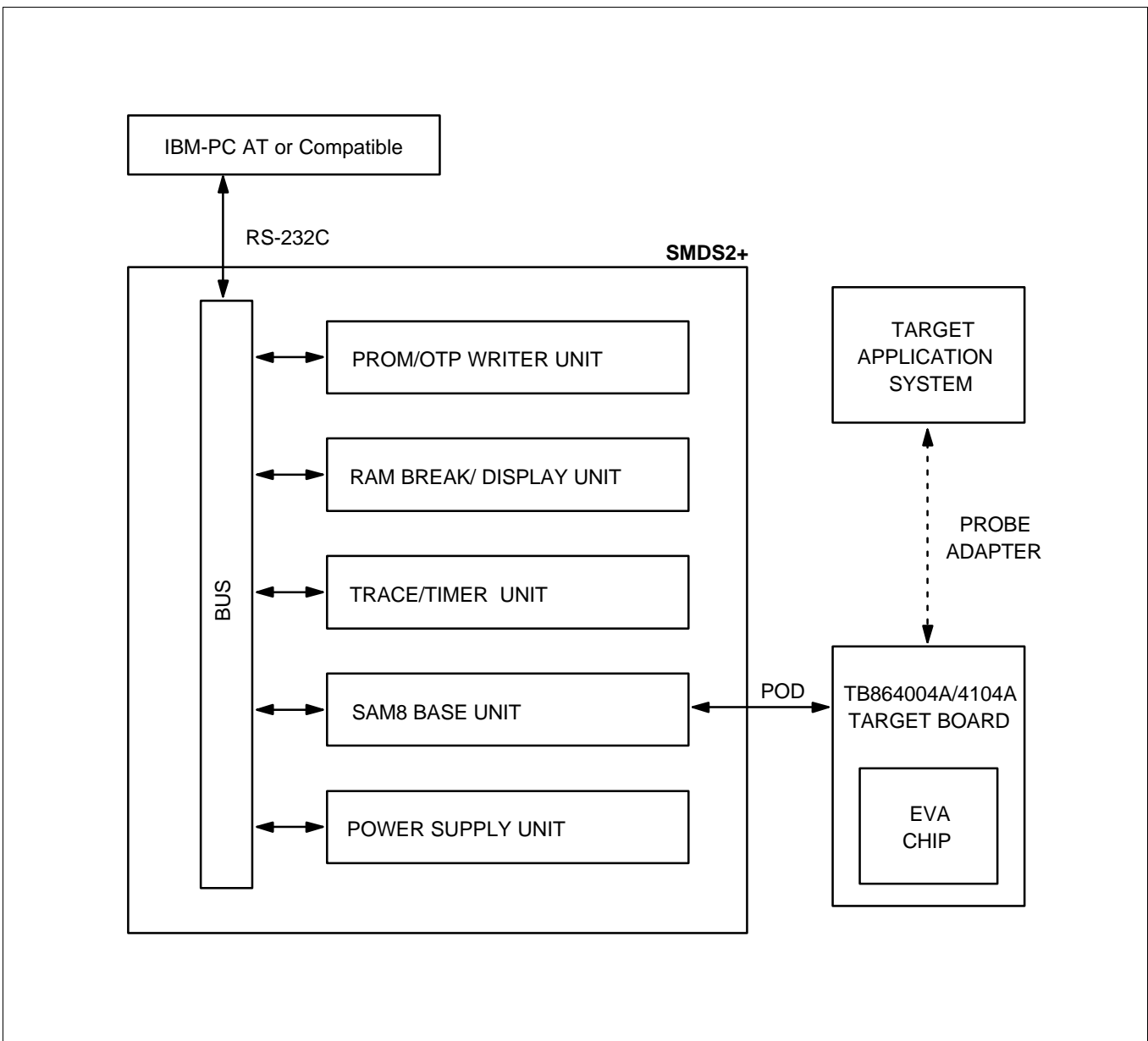


Figure 16-1. SMDS Product Configuration (SMDS2+)

TB864004A/4104A TARGET BOARD

The TB864004A/4104A target board is used for the KS86C4004/C4104 microcontrollers. It is supported by the SMDS2+ development systems. The TB864004A/4104A target board can also be used for KS86C4004/C4104.

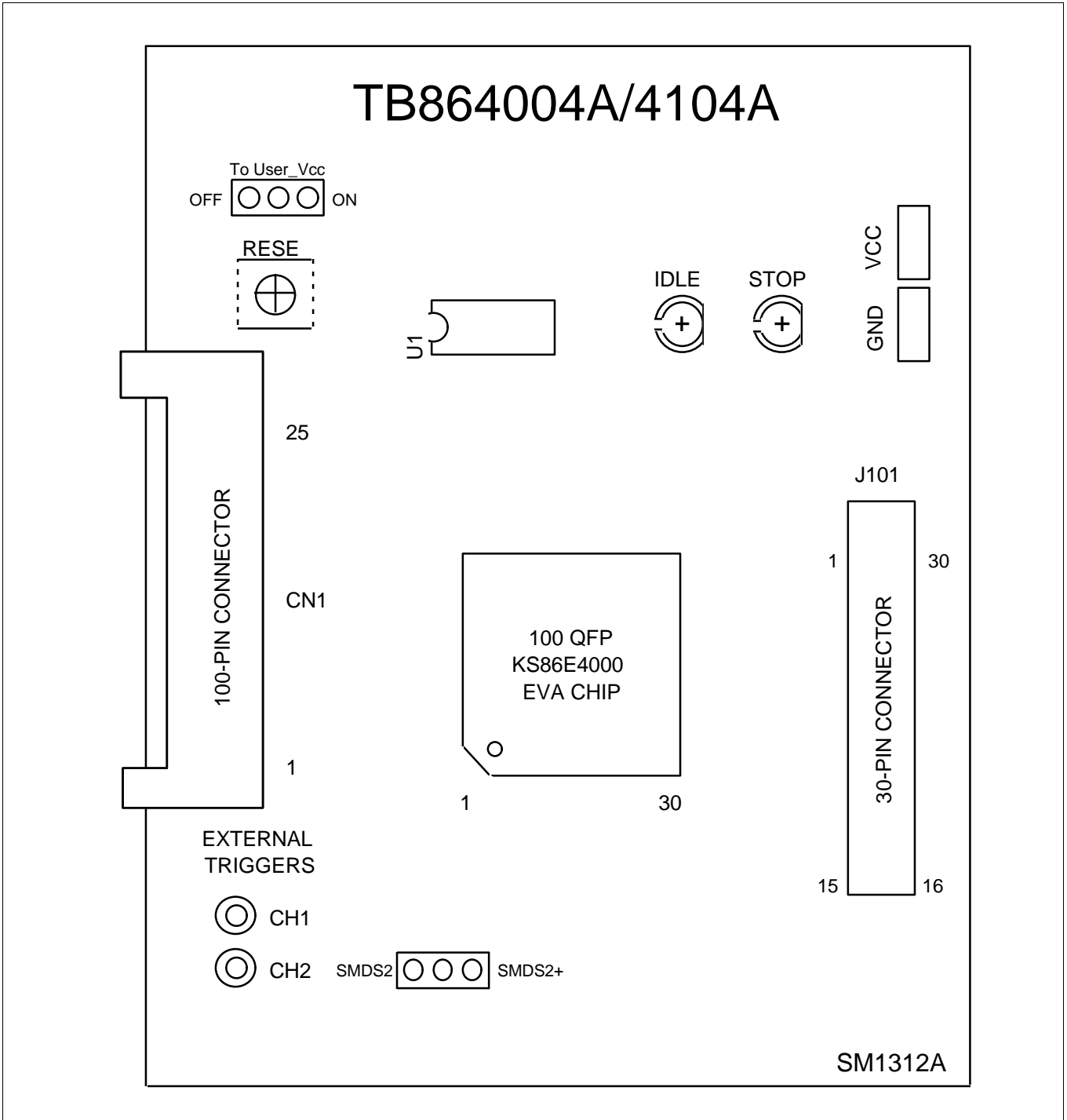

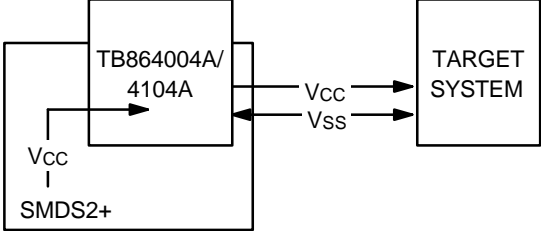

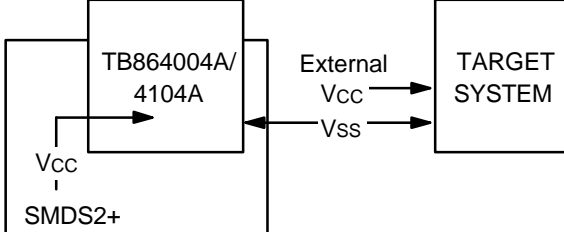


Figure 16-2. TB864004A/4104A Target Board Configuration

Table 16-1. Power Selection Settings for TB864004A/4104A

'To User_Vcc' Settings	Operating Mode	Comments
<p>To User_Vcc</p> <p>OFF  ON</p>		<p>The SMDS2+ main board supplies V_{CC} to the target board (evaluation chip) and the target system.</p>
<p>To User_Vcc</p> <p>OFF  ON</p>		<p>The SMDS2+ main board supplies V_{CC} only to the target board (evaluation chip). The target system must have its own power supply.</p>

NOTE: The following symbol in the 'To User_Vcc' Setting column indicates the electrical short (off) configuration:



SMDS2+ Selection (SAM8)

In order to write data into program memory that is available in SMDS2+, the target board should be selected to be for SMDS2+ through a switch as follows. Otherwise, the program memory writing function is not available.

Table 16-2. The SMDS2+ Tool Selection Setting


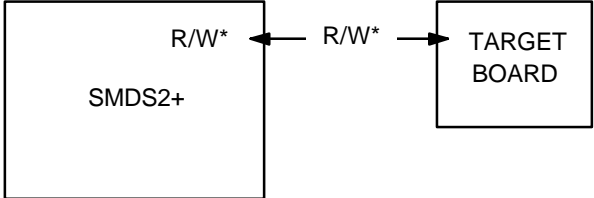
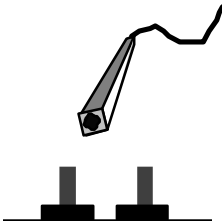
'SW1' Setting	Operating Mode
<p>SMDS2  SMDS2+</p>	

Table 16-3. Using Single Header Pins as the Input Path for External Trigger Sources

Target Board Part	Comments
<p>EXTERNAL TRIGGERS</p> <p>○ CH1</p> <p>○ CH2</p>	<div style="display: flex; align-items: center;">  <div style="margin-left: 20px;"> <p>Connector from external trigger sources of the application system</p> </div> </div> <p>You can connect an external trigger source to one of the two external trigger channels (CH1 or CH2) for the SMDS2+ breakpoint and trace functions.</p>

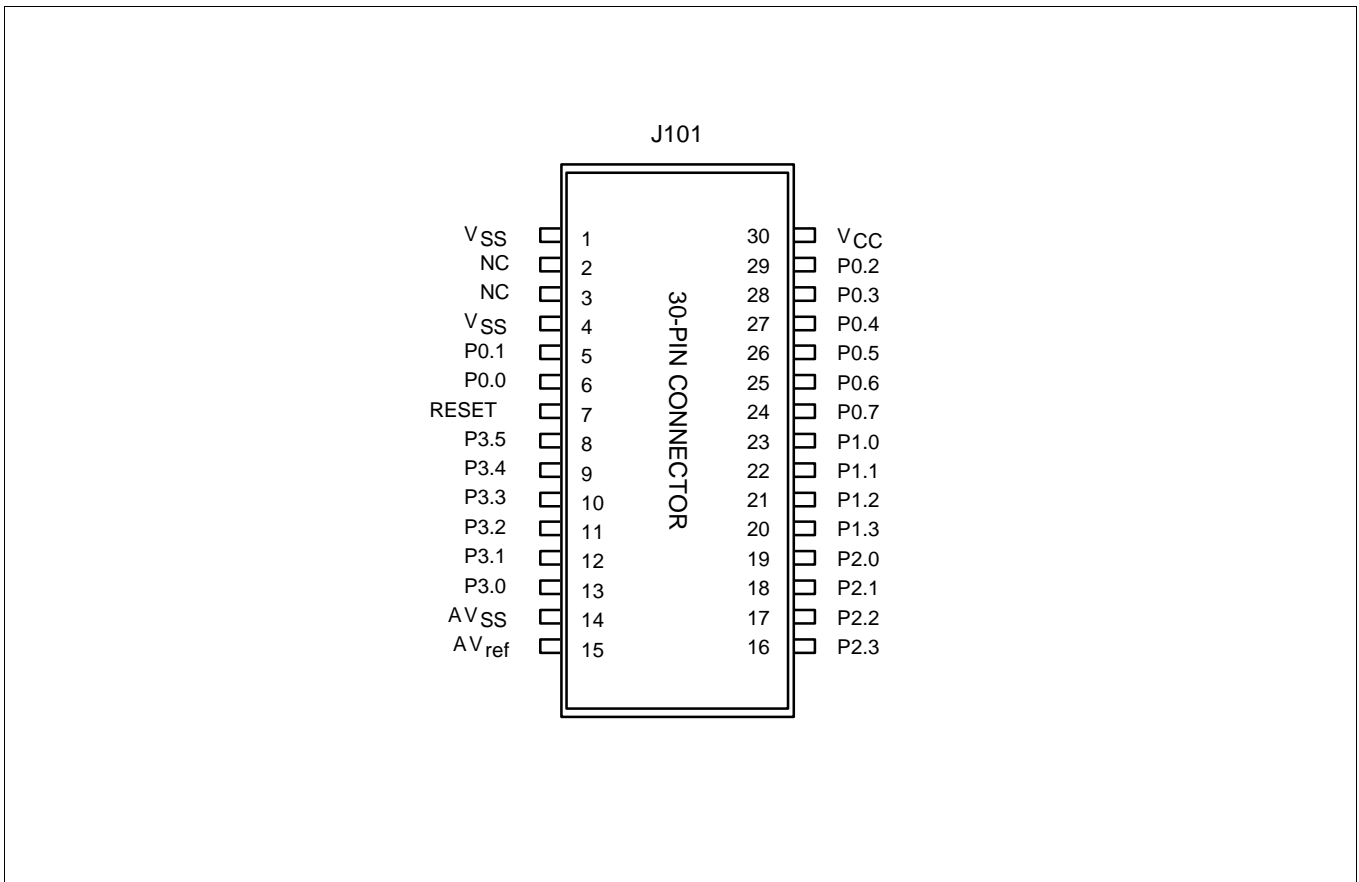


Figure 16-3. 30-Pin Connector for TB864004A/4104A

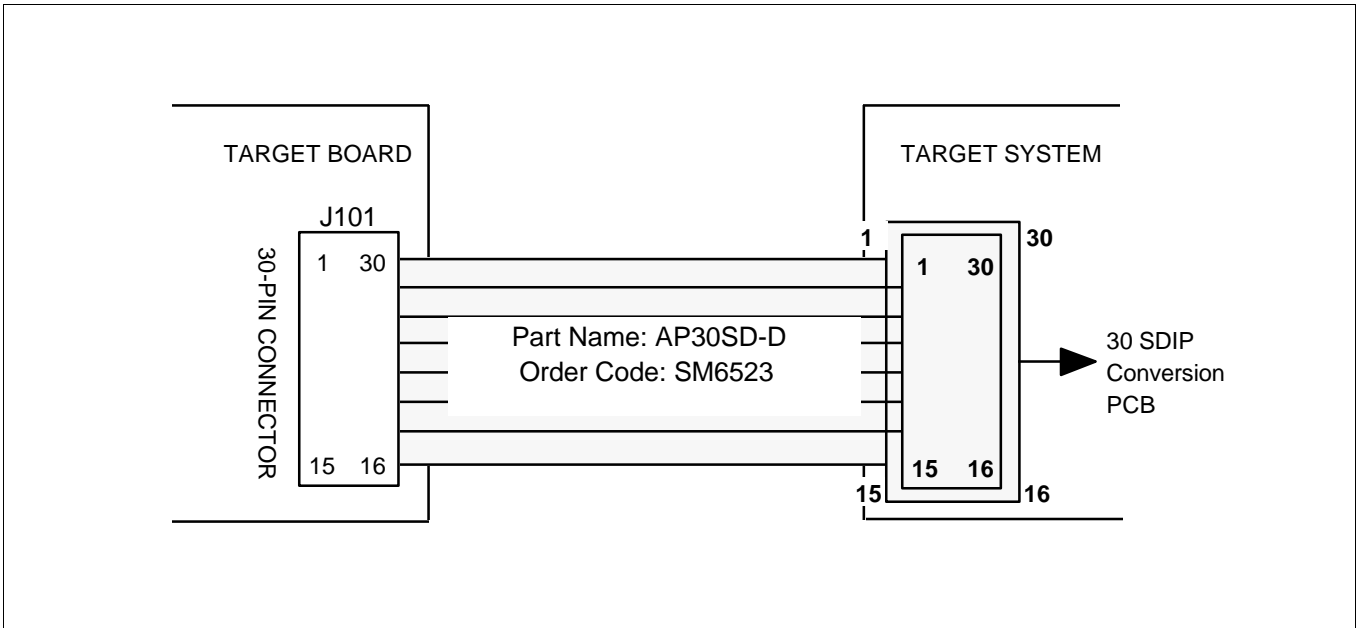


Figure 16-4. KS86C4004 Probe Adapter for 30-SDIP Package

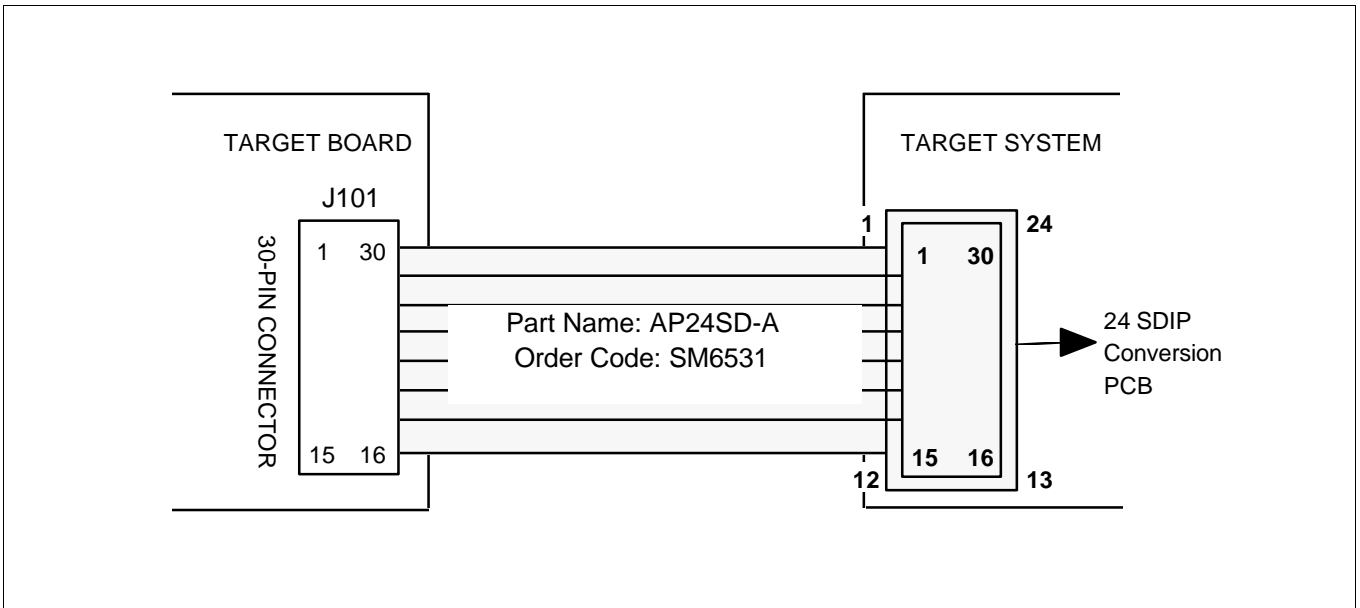


Figure 16-5. KS86C4104 Probe Adapter for 24-SDIP Package

NOTES