NEC Electronics Inc.

NEC

USER'S MANUAL

April 1987

μPD7810/11, 7810H/11H, 78C10/C11/C14 Microcomputers

8-Bit, Single-Chip, With A/D Converter

NEC Electronics Inc.

NEC

μPD7810/11, 7810H/11H, 78C10/C11/C14 Microcomputers

8-Bit, Single-Chip, With A/D Converter

April 1987
Stock No. 500375

Revision History

June 1985	Original issue was μPD7810/7811 NMOS Microcomputers.
April 1987	Added μPD7810H/11H (NMOS) and μPD78C10/C11/C14 (CMOS) Microcomputers. Because this is a major revision, additions and changes are not highlighted by marginal arrows.

Preface

The μPD7810/11, 7810H/11H, 78C10/C11/C14 Microcomputers User's Manual is directed toward design, field, and applications engineers and assembly language programmers.

The organization of each section and appendix of this manual follows:

- Section 1, Introduction, summarizes the microcomputer's important features, including registers, on-chip peripherals, instruction set, part differences, pin configurations, and block diagram.
- Section 2, Pin Functions, describes in detail the function of signals carried by each pin on the microcomputer packages.
- Section 3, Functional Description, gives overview information about architecture, registers, memory, an on-chip peripherals.
- Section 4, Port Operation, details the microcomputer ports, port operating modes, control registers, memory mapping, and timing for opcode fetch, memory read, and memory write. This section also discusses port emulation mode.
- Section 5, Interval Timer, describes the organization and operation of the on-chip interval timer and discusses the timer's comparator registers, clock sources, and operation.
- Section 6, Multifunction Timer/Event Counter, describes the features, organization, and operation of the 16-bit timer/event counter. The section explains the counter registers, capture registers, and interrupt control and output control circuitry. It also gives examples of program code for operating the timer/event counter.
- Section 7, Serial Interface, describes the organization, operation, and programming of the serial interface. This section also discusses the receiver, serial mode registers, and transmission modes and gives programming examples.
- Section 8, Analog-to-Digital Converter, explains the A/D converter's configuration and operating modes and gives a programming example.
- Section 9, Interrupt Control Structure, describes the hardware configuration of the interrupt controller, lists interrupt priorities and vector locations, details the functional elements and operation of the structure, and discusses interrupt wait times.
- Section 10, Reset, Halt, Stop, and Standby, discusses the microcomputer's operating modes. This section details the standby mode, reset and halt operations, and software and hardware stop modes and discusses releasing both stop and halt modes.
- Section 11, Instruction Format, lists instruction formats and conventions for writing code. The section also discusses program sequencing, addressing modes, skip operation timing, and overlay instructions.
- Section 12, Instruction Set, lists the instructions alphabetically with detailed descriptions and programming examples.
- Appendix A, Alphabetical List of Instructions, gives a short description of each instruction.
- Appendix B, Abbreviated Instruction Set, organizes the instructions by function in a table format with coded information.
- Appendix C, Package Drawings, provides dimensional drawings of the four types of microcomputer packages.

CONTENTS

CONTENTS (cont)

CONTENTS (cont)

CONTENTS (cont)

Description

The μPD7810/7811, μPD7810H/7811H, μPD78C10/78C11, and μPD78C14 are advanced high-end, stand-alone microcomputers implemented from a powerful enhancement of basic 8-bit 7800 architecture. These microcomputers combine the consistent and simple 8-bit 7800 architecture with the processing capability of a 16-bit ALU to provide superior speed and efficiency in a straightforward, easy-to-use configuration. In addition to containing all the functional blocks required for a complete integrated microcomputer, these devices have unique peripheral capability usually assigned to external components. This ability allows the microcomputers to have extensive processing capability in a single self-contained unit.

Features

- ☐ Complete, stand-alone microcomputer on a single chip
- ☐ 8-bit external/16-bit internal bus architecture fabricated using MOS silicon technology
- ☐ Instruction time
  - — 1 μs shortest instruction time (7810/7811)
  - — 0.8 μs shortest instruction time (7810H/7811H, 78C10/78C11, and 78C14)
- ☐ Clock
  - — 12-MHz clock, 0.25-μs T-states (7810/7811)
  - — 15-MHz clock, 0.20-μs T-states (7810H/7811H, 78C10/78C11, and 78C14)
- ☐ 256-byte RAM
- ☐ ROM
  - — 4K bytes (7811, (7811H, and 78C11)
  - — 16K bytes (78C14)
- ☐ ROMless version (7810, 7810H, and 78C10)
- ☐ Direct addressing of up to 64K bytes
- ☐ 40 port lines, plus four edge-sensing inputs
- ☐ Comprehensive, powerful instruction set
  - — 16-bit data transfers
  - — 16-bit arithmetic and logic operations
  - — 8-bit x 8-bit multiply operation with 16-bit results
  - — Divide operation (8-bit divisor, 16-bit dividend, 16-bit quotient, 8-bit remainder)
  - — 16-bit shift and rotate operations
  - — 16-bit index operations
- ☐ 12 addressing modes
- ☐ 8-bit and 16-bit data manipulation capability
- ☐ Three external and nine internal interrupts
- ☐ Multilevel subroutine nesting limited only by memory size
- ☐ Seven-level prioritized, vectored interrupt structure
- ☐ Full-duplex USART
- ☐ Eight-channel, 8-bit A/D converter
- ☐ Two programmable 8-bit interval timers
- ☐ Multifunction 16-bit timer/event counter
- ☐ Zero-cross detection with two inputs
- ☐ 8085A-like bus
- ☐ Standby function
- ☐ Single +5-volt power supply
- ☐ Available in 64-pin plastic QUIP, 64-pin plastic shrink DIP, 64-pin plastic miniflat, and 68-pin PLCC

Registers

The dual register set organization consists of the following main and alternate registers.

- 8-bit accumulator
- 8-bit vector register
- 16-bit extended accumulator
- 8-bit general-purpose registers (six)

These registers and an 8-bit program status word (PSW), 16-bit program counter, 16-bit stack pointer, 16-bit arithmetic logic unit (ALU), and a 16-bit internal data bus constitute the main functional units of the 7810/7811, 7810H/7811H, 78C10/78C11, and 78C14. The 8-bit registers can be used individually or paired as 16-bit registers, enabling operations on 8-and 16-bit data. The dual register set organization permits swapping data between registers and their alternates for operation in foreground/background mode. The microcomputers each contain 40 I/O port lines.

Peripherals

Peripheral functions include a full-duplex USART, eight-channel/8-bit A/D converter, 16-bit multifunction timer/event counter, two programmable 8-bit interval timers that may be paired as a single 16-bit timer or used as two discrete 8-bit timers, and two zero-cross detection inputs.

Instruction Set

A powerful, comprehensive instruction set features 16-bit data transfers, 16-bit arithmetic and logic operations, 16-bit shift and rotate operations, 8-bit by 8-bit multiply operations with 16-bit results, divide operations (16-bit dividend, 8-bit divisor, 16-bit quotient, 8-bit remainder), and 16-bit index operations. Operating with a 12-MHz clock, a timing state (T-state) equals one 0.25-μs $\phi 3$ clock period (frequency of $\phi 3 = f_{XTAL}/3$). Multiply operations require 8 μs to execute; divide operations, less than 15 μs. Programming efficiency and flexibility are further enhanced by 12 addressing modes.

Additional features include a seven-level prioritized, vectored interrupt structure, 256 bytes of RAM, 4K bytes of ROM (7811, 7811H, 78C11) or 16K bytes (78C14), direct addresing of up to 64K bytes of memory, multilevel subroutine nesting limited only by memory size, a standby function, and a single +5-volt power supply.

Part Differences

The following describes differences among the various parts.

(1) 7810 is the same as 7811 except 7810 does not contain ROM.

(2) 7810H/7811H (15 MHz) is the same as 7810/7811 (12 MHz) except for the maximum crystal input frequency.

(3) 7810/7811 and 7810H/7811H are NMOS parts.

(4) 78C10/78C11 and 78C14 are CMOS parts.

(5) 78C11 (4K-byte ROM) is the same as 78C14 (16K-byte ROM) except for ROM size.

(6) 7811, 7811H, 78C11, and 78C14 are "ROM parts."

(7) 7810, 7810H, and 78C10 are "ROMless parts."

Table 1-1 shows differences between 7810/7811 and 78C10/78C11/78C14. The various sections of this manual discuss these differences in detail.

Pin Configurations

Figures 1-1, 1-2, and 1-3 are the package pin configuration drawings. They apply to all versions of the microcomputer.

Block Diagram

Figure 1-4 is the block diagram of the 7810/7811, 7810H/7811H, 78C10/78C11, and 78C14 microcomputers.

Table 1-1. Differences Between 7810/7811 and 78C10/78C11/78C14

Features	78C10/78C11/78C14	7810/11
Instruction set	159 instructions (addition of STOP instruction) Additional sr register operand (ZCM)	158 instructions
	HALT instruction (12 states)	HALT instruction (11 states)
Standby modes		
Halt mode	Yes	Yes
Software stop mode	Yes	No
Hardware stop mode	Yes	No
Backup RAM	256 bytes	32 bytes Separate backup power supply for 7811 RAM.
A/D converter	Power down capability. When V_{AREF} is reduced to 0 V, A/D converter power is turned off, reducing power during standby modes.	No power-down capability
Noise protection		
$\overline{NMI}$	3 μs (typical) Analog delay circuit removes noise independent of internal states.	12 states
$\overline{RESET}$	3 μs (typical)	20 states
$\overline{STOP}$	500 ns (typical)	
Zero-cross bias control	Yes Bias on or off by writing to ZCM register	No
State of pins at reset (ROMless parts)	High impedance	High or low

Figure 1-1. Pin Configuration, 64-Pin Plastic QUIP or Shrink DIP

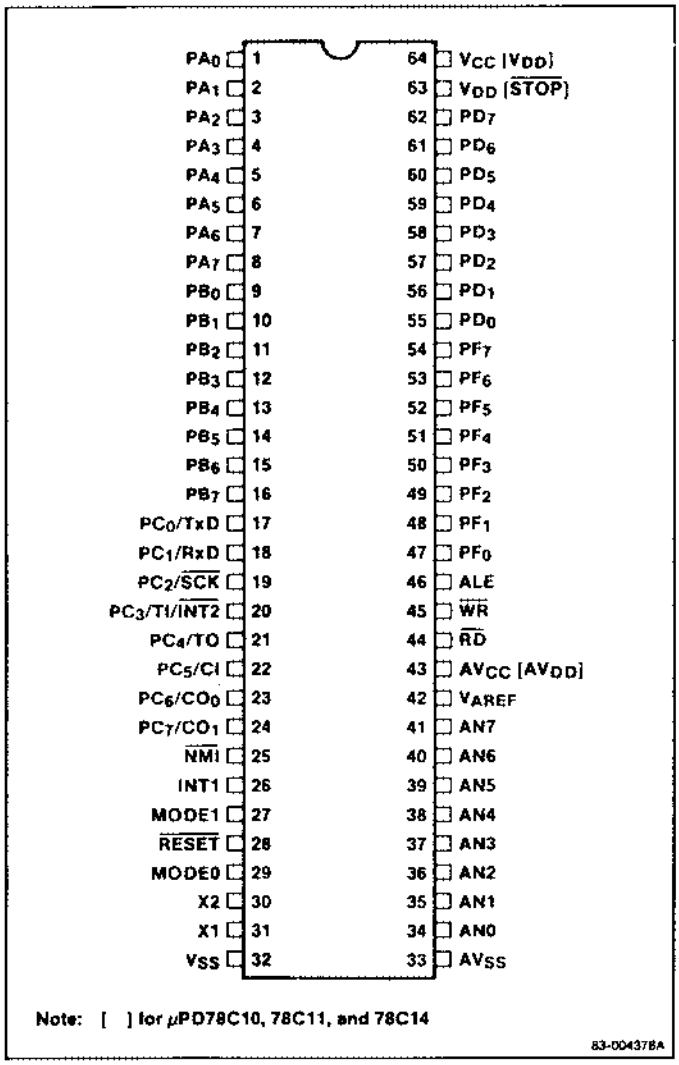


Figure 1-2. Pin Configuration, 64-Pin Plastic Miniflat

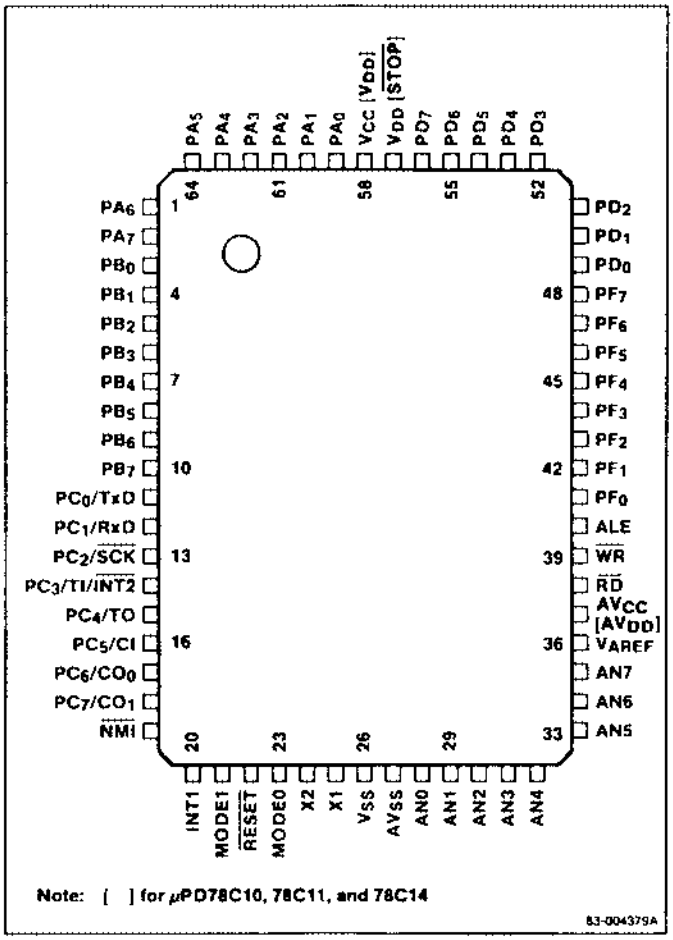
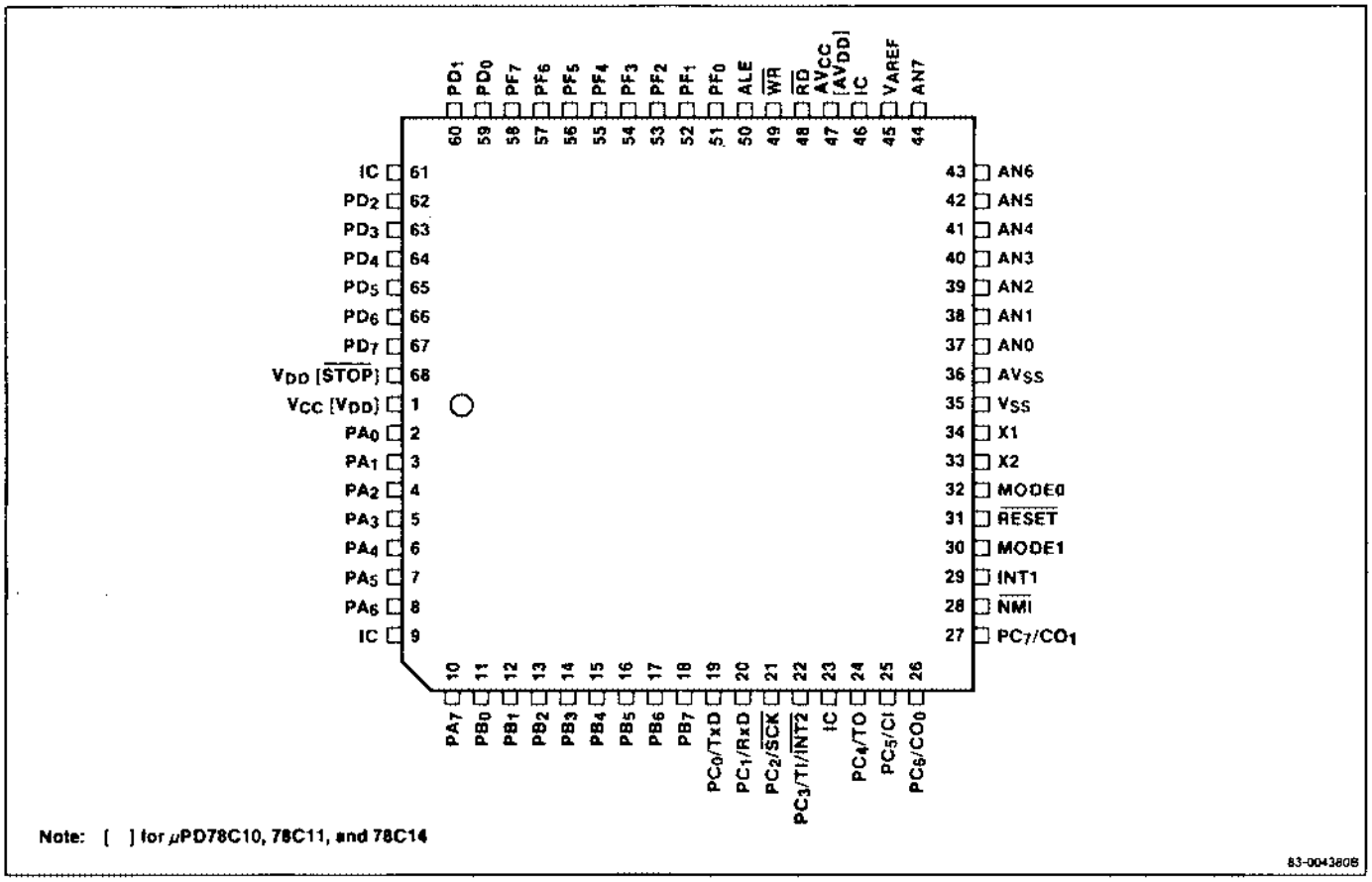


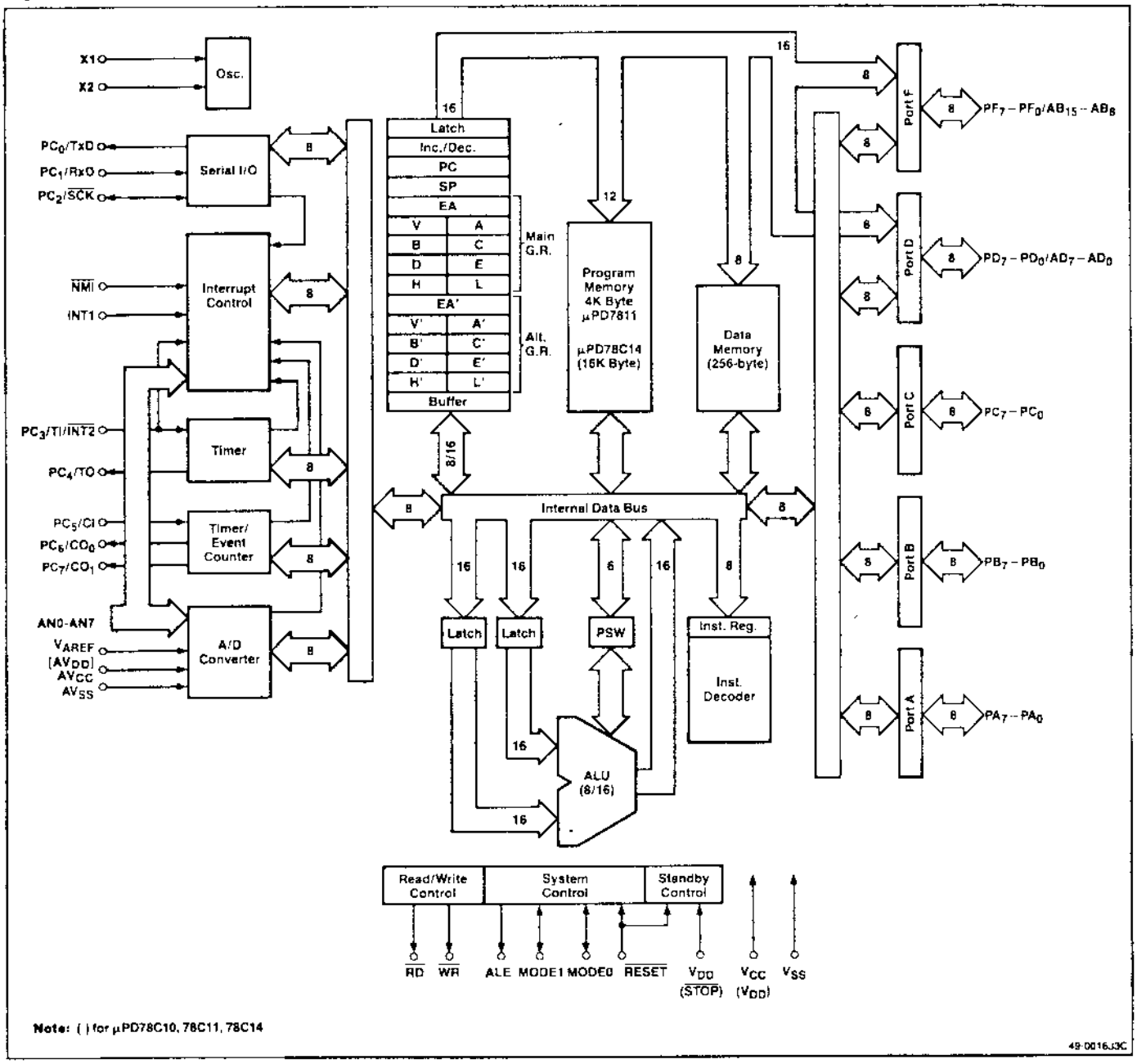
Figure 1-3. Pin Configuration, 68-Pin Plastic Leaded Chip Carrier (PLCC)



Note: [ ] for μPD78C10, 78C11, and 78C14

83-0043808

Figure 1-4. Microcomputer Block Diagram



Note: ( ) for µPD78C10, 78C11, 78C14

This section describes the functions of signals carried by each 7810/7811, 7810H/7811H, 78C10/ 78C11, and 78C14 pin. The descriptions apply to all parts unless otherwise specified.

Port A [PA_7-PA_0]

These eight lines constitute port A, a programmable 8-bit I/O port with a latched, three-state output. Each line may be used independently as an input or an output. Any line can be placed in input or output mode by the mode A register (MA). Writing a one to the corresponding bit of register MA sets the port line to input mode; writing a zero sets the line to output. A reset places all port A lines in input mode by writing ones to all bits of the MA register.

Port B [PB_7-PB_0]

These eight lines constitute port B, a programmable 8-bit I/O port with a latched, three-state output. Port B operates identically to port A, using mode register MB to set lines to input or output.

Port C [PC_7-PC_0]

These eight lines constitute port C, a programmable 8-bit I/O port with a latched, three-state output. Port C operates in two modes: port mode and control mode.

In port mode, port C operates identically to ports A and B, using mode register MC to set lines to input or output. In control mode, each line has a set function as a specific control signal. Each line can be independently set by the mode control C register (MCC) to operate in either port mode, by setting the corresponding bit of MCC to 0, or control mode by setting the corresponding bit to 1. A reset places all port C lines in port mode, with all lines designated as input (MCC set to all 0's, MC set to all 1's).

The function of each port C line in control mode is shown in table 2-1.

Port D [PD_7-PD_0]

These eight lines constitute port D. Port D operates as either multiplexed data/address lines or as a programmable 8-bit I/O port, depending on whether or not external memory is used.

When no external memory is connected, port D is a programmable 8-bit I/O port with a latched, three-state output. It may be enabled as an 8-bit wide output port or input port or as multiplexed data/address lines by the memory mapping register (MM). Unlike ports A, B, C, and F, the individual lines of port D cannot be independently enabled as input or output.

Table 2-1. Port C Control Mode

Symbol	Function
PC_0	Serial data output line (TxD).
PC_1	Serial data input line (RxD).
PC_2	Serial clock I/O line ($\overline{SCK}$). Provides an output clock for the internal clock mode or an input line for an external clock mode.
PC_3	Timer input line (TI), interrupt request line ($\overline{INT2}$), or one of two zero-cross detect inputs for an ac signal.
PC_4	Timer output line (TO). Outputs a square wave from Timer 0, Timer 1, or the $\phi 3$ internal clock.
PC_5	Counter input line (CI). Provides an external clock pulse input to the timer/event counter.
PC_6-PC_7	Counter output lines (CO_0 and CO_1, respectively). Programmable outputs from the timer/event counter.

In the 7811, a reset initializes port D as an 8-bit input port (all lines go to a high-impedance state). For the 7810, 7810H, and 78C10, a reset initializes port D as eight multiplexed data/address lines.

Port F [PF_7-PF_0]

These eight lines constitute port F. Groups of lines of port F can be enabled as either port lines or address lines (depending on the amount of external memory being used) by writing to the memory mapping register (MM).

When 256 or fewer bytes of external memory are used, port F can be used as a programmable 8-bit I/O port in which each individual line may be designated independently as an input or an output by programming the mode F register. The three-state outputs are latched.

When more than 256 bytes of external memory are used, specific lines become dedicated address lines as shown in table 2-2. Any port F line not used as a dedicated address line in a particular memory configuration is free to be used as a programmable port line, as designated by the mode F register. When more than 16K of external memory is used, port F becomes the upper half of a 16-bit address bus (the lower half is port D).

In the 7811, 7811H, 78C11, and 78C14, a reset initializes port F as an 8-bit input port (all lines go to a high-impedance state). Writing to the MM register reconfigures port F to the appropriate combination of address and I/O lines.

Table 2-2. External Memory/Port Configuration

External Memory	Port	Configuration
None	PD_7-PD_0	I/O port lines
	PF_7-PF_0	I/O port lines
1-256 bytes	PD_7-PD_0	Multiplexed data/address lines (2)
	PF_7-PF_0	I/O port lines
1-4096 bytes	PD_7-PD_0	Multiplexed data/address lines (2)
	PF_3-PF_0	Address lines (2)
	PF_7-PF_4	I/O port lines
1-16,384 bytes	PD_7-PD_0	Multiplexed data/address lines (2)
	PF_5-PF_0	Address lines (2)
	PF_7-PF_6	I/O port lines
1-65,536 bytes (1)	PD_7-PD_0	Multiplexed data/address lines (2)
	PF_7-PF_0	Address lines

Notes:

(1) 48K bytes maximum memory expansion for the 78C14; 60K bytes for the 7811, 7811H, and 78C11; 64K bytes maximum for the 7810, 7810H, and the 78C10.

(2) PF_7-PF_0 = address lines AB_{15}-AB_8
PD_7-PD_0 = address lines AD_7-AD_0

In the 7810, 7810H, and 78C10, the mode pins (MODE0 and MODE1) determine the memory configuration. The appropriate number of port F lines required to support the memory configuration are initialized as address lines, and the remaining lines are initialized as input port lines.

Nonmaskable Interrupt [$\overline{NMI}$]

The nonmaskable interrupt is a falling-edge, Schmitt-triggered input. It is always acknowledged at the end of the current instruction, regardless of the status of the mask register or interrupt enable flip-flop. $\overline{NMI}$ requires an external pullup resistor.

Interrupt Request 1 [INT1]

This input line is the maskable interrupt request INT1. It is rising-edge triggered, and has a priority level of 3. This line is also one of two zero-cross detection inputs.

Reset [$\overline{RESET}$]

This active-low, Schmitt-triggered input signal initializes the microcomputer. On reset, all ports are designated as input ports (on the 7810, 7810H, and 78C10, port D is designated as an 8-bit address/data bus), and the $\overline{WR}$ and $\overline{RD}$ signals go high. Program execution begins from address 0000H upon removal of the $\overline{RESET}$ input signal.

Mode 0 [MODE0]

This line should be grounded in the 7811, 7811H, 78C11, and 78C14. In the 7810, 7810H, and 78C10, this line is used in conjunction with MODE1 to specify the maximum external memory capacity. MODE0 requires an external pullup resistor. When MODE0 is pulled up, it is used to output the $\overline{IO}$ signal during a special register reference.

The maximum value of the pullup resistor is critical. It is specified in the data sheet, and the value is a function of the crystal frequency and the capacitance at the input pin.

Mode 1 [MODE1]

This line should be pulled up in the 7811, 7811H, 78C11, and 78C14 and is used in conjunction with MODE0 to specify whether the part has ROM or is ROMless. In the ROMless parts, MODE0 and MODE1 specify the maximum external memory capacity as shown in table 2-3.

Table 2-3. Maximum External Memory

MODE 1	MODE 0	External Memory
1	0	7811, 7811H, 78C11, and 78C14. Memory mapping register determines amount of external memory.
0	0	7810, 7810H, and 78C10 with 4K bytes memory (address 0-4095)
0	1	7810, 7810H, and 78C10 with 16K bytes memory (address 0-16,383)
1	1	7810, 7810H, and 78C10 with 64K bytes memory (address 0-65,536)

When pulled up, MODE1 also outputs the $\overline{M1}$ (machine cycle 1) signal to indicate the first cycle of each instruction. This signal is asserted during the opcode fetch phase of an instruction.

The maximum value of the pullup resistor is critical. It is specified in the data sheet, and the value is a function of the crystal frequency and the capacitance at the input pin.

Crystal 1 [X1]

This line is one of two crystal connections for the internal clock-generator circuit. It may also be used as an input line for an external clock source.

Crystal 2 [X2]

This line is one of two crystal connections for the internal clock-generator circuit.

Ground [V_{SS}]

This line is the ground potential.

Analog Ground [AV_{SS}]

This line is the ground terminal for the resident analog-to-digital converter.

Analog Inputs [AN0-AN7]

These eight lines are the analog inputs to the resident A/D converter. Lines AN4-AN7 may also be used as four edge-sensing inputs. Whenever a falling edge is sensed, a test flag (AN4-AN7) is set.

Analog Reference Voltage [V_{AREF}]

This line is the reference voltage input for the resident A/D converter.

Analog Power Supply [AV_{CC}]

This is the power supply for the resident A/D converter. Note that the symbol for the 78C10/78C11/78C14 is AV_{DD}.

Read [$\overline{RD}$]

This active-low line strobes data from external memory or I/O devices onto the data bus. $\overline{RD}$ is asserted to indicate that the CPU is requesting data from external memory or I/O devices. The line stays high for all other conditions.

Write [$\overline{WR}$]

This active-low line strobes data from the data bus to external memory or I/O devices. $\overline{WR}$ is asserted to indicate that the data bus holds valid data. The line stays high for all other conditions.

Address Latch Enable [ALE]

This line strobes the address on address lines PD_7-PD_0 (AD_7-AD_0) and the necessary lines of PF_7-PF_0 (AB_{15}-AB_8) to external memory, enabling access of external memory. ALE is asserted when the address bus holds a valid external memory address.

Power Supplies [V_{CC} and V_{DD}]

The 7810/7811 and 7810H/7811H have two +5-volt power inputs. V_{CC} is the primary supply. V_{DD} supplies power to the entire 256-byte resident RAM in normal mode; in standby mode, it maintains the contents of the upper 32 bytes only.

The 78C10/78C11 and 78C14 have one +5-volt power input and it is labeled V_{DD}.

Stop [$\overline{STOP}$]

In the 78C10/78C11 and 78C14, this is the control signal for the hardware stop mode. A low level on this Schmitt-triggered input stops the oscillator.

REGISTER SET

The microcomputer architecture is organized with a dual register set (figure 3-1), which includes a main register set and an identical alternate register set. This dual register set includes a pair of 16-bit extended accumulators (EA and EA'), a pair of 8-bit accumulators (A and A'), a pair of 8-bit vector registers (V and V'), and six pairs of 8-bit general-purpose registers (B, C, D, E, H, L, and B', C', D', E', H', L').

The 8-bit general-purpose registers may be used independently or may be coupled to function as 16-bit registers (BC, DE, HL, or B'C', D'E', 'H'L').

An 8-bit program status word (PSW), a 16-bit stack pointer (SP), and a 16-bit program counter (PC) complete the register set.

A 16-bit ALU, 16-bit internal data path, and the 16-bit program counter and stack pointer—plus the ability to pair the general-purpose registers into 16-bit entities—permits full operation of instructions using either 8-bit or 16-bit data.

Powerful 16-bit processing includes 16-bit data transfers, 16-bit shift and rotate operations, 16-bit index operations, 16-bit arithmetic and logic operations, 8-bit by 8-bit multiplies with 16-bit results, and divide operations with 8-bit divisor, 16-bit dividend, 16-bit quotient, and 8-bit remainder. In addition, the ability to access an alternate set of registers permits simplified foreground/background programming.

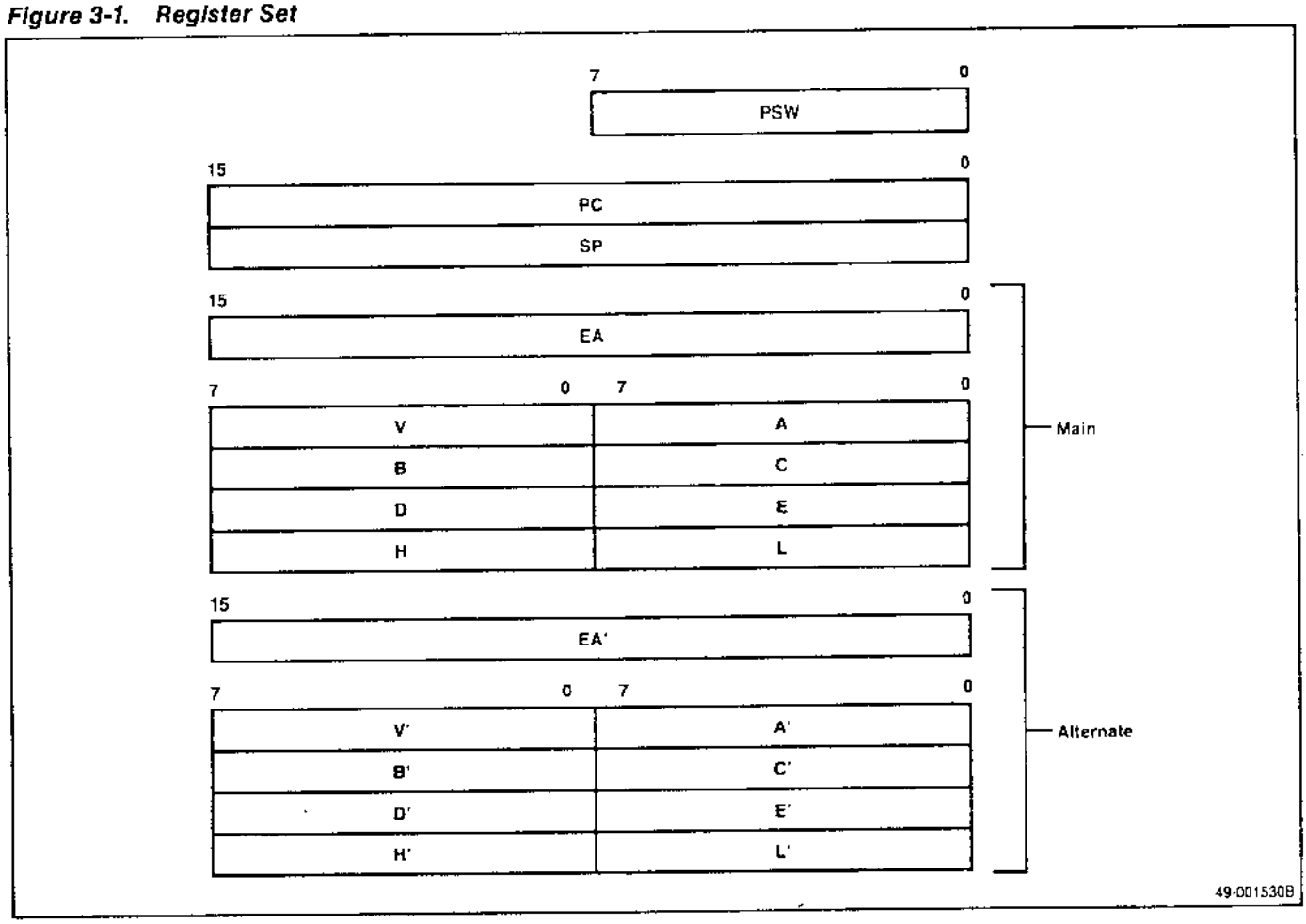
Accumulator [A] and Extended Accumulator [EA]

The microcomputer has an accumulator-oriented architecture in which most arithmetic and logic operations are processed via the accumulator (8-bit operations using accumulator A and 16-bit operations using extended accumulator EA). The contents of the two accumulators, A and EA, can be exchanged with their alternate registers, A' and EA', using the EXA instruction.

Vector Register [V]

The vector register can select a 256-byte work space in memory. The higher-order 8 bits of the address are

Figure 3-1. Register Set



specified by the contents of the vector register. The lower-order 8 bits of the address are specified in the immediate data of the instruction. The contents of V and V' may be exchanged using the EXA instruction.

General-Purpose Registers [B, C, D, E, H, L]

The general-purpose registers, in conjunction with the accumulator, perform the majority of arithmetic and logic operations. Eight-bit operations use accumulator A and individual general registers; 16-bit operations use extended accumulator EA and 16-bit general register pairs. The main register set and the alternate register set can be used for interrupt processing. The contents of one set may be exchanged with the other to facilitate programming and to increase throughput.

The general-purpose registers may be used individually or paired as 16-bit entities as follows: BC, DE, HL or B'C', D'E', H'L'. Either of register pairs DE or HL (or D'E' or H'L') can function as a base register for use in register indirect addressing, which includes the autoincrement, autodecrement, and double autoincrement addressing modes, as well as the base and base-index addressing modes (see Section 11).

The contents of the main and alternate set of general-purpose registers may be exchanged using the EXX instruction. The EXH instruction can be used to exchange the contents of only the HL and H'L' registers.

Program Counter [PC]

The 16-bit program counter holds the address of the next instruction to be executed. Upon instruction execution, the program counter is automatically incremented by the number of bytes contained in the instruction being executed. When the instruction is a branch instruction, the program counter is updated by either the immediate data of the instruction or the contents of a specified register. A reset clears the program counter, beginning program execution from address 0000.

Stack Pointer [SP]

The 16-bit stack pointer holds the address of the initial position of a reserved LIFO stack area in memory. The stack pointer is decremented by a subroutine call or PUSH instruction, or by a vector to an interrupt service routine. It is incremented by a return from subroutine or POP instruction. The stack pointer allows subroutines to be nested up to the limit of memory.

Control Registers

In addition to the basic register set, there are several special mode registers, which control such functions as port/external memory configuration and operating parameters for the peripheral interfaces. These mode registers and their functions are listed in table 3-1.

Table 3-1. Mode Registers

Register	Symbol	Function
Mode A	MA	Designates each line of port A independently as either input or output.
Mode B	MB	Designates each line of port B independently as either input or output.
Mode control C	MCC	Designates either port mode or control mode independently for each line of port C.
Mode C	MC	When register MCC designates a line of port C as a port line, register MC designates it as either input or output.
Memory mapping	MM	7811, 7811H, 78C11, and 78C14 only. Selects the port or external memory configuration for ports D and F. Enables or disables access to the internal RAM.
Mode F	MF	Designates each line of port F independently as either input or output.
Timer mode	TMM	Designates the operating parameters for the TIMER0 and TIMER1 8-bit interval timers.
Timer/event mode	ETMM	Designates the parameters for the 16-bit multifunction timer/event counter.
Timer/event counter output mode	EOM	Designates the parameters for the output control circuit of the 16-bit multifunction timer/event counter.
Serial mode (high byte/low byte)	SMH/SML	Designates the operating parameters for the serial interface.
A/D channel mode	ANM	Designates the operating parameters for the eight-channel, 8-bit analog-to-digital converter.
Interrupt mask (low byte/ high byte)	MKL/MKH	Designates which interrupts are masked and which are not.
Zero-cross mode	ZCM	Selects bias circuitry for ac zero-cross detection.

Program Status Word [PSW]

The 8-bit PSW (shown in figure 3-2) contains the flags that indicate the status of the processor immediately following an instruction execution. The contents of the PSW are automatically saved on the stack in the event of an interrupt (internal, external, or SOFTI) and are restored by the RETI instruction (see Section 9). A reset clears all bits of the PSW.

Table 3-2 describes the flags in the PSW; table 3-3 lists all the intructions that affect the flags.

Figure 3-2. Program Status Word

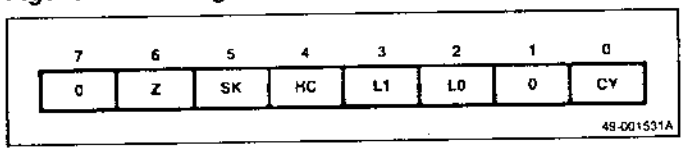


Table 3-2. Program Status Word Flags

Bit	Flag	Description
0	CY	The carry (CY) flag is set whenever a carry from bit 7 or bit 15 to the CY bit, or a borrow from the CY bit to bit 7 or bit 15, occurs. This flag is cleared for all other conditions.
1		This bit is always zero and is not used.
2	L0	The overlay zero (L0) flag is set whenever an MVI L,byte or LXI H,word instruction is executed. Subsequent MVI L,byte or LXI H,word instructions are skipped until another type of instruction is executed. L0 is cleared for all other instructions.
3	L1	The overlay one (L1) flag is set whenever an MVI A,byte instruction is executed, and cleared for all other instructions. When L1 is set, subsequent MVI A,byte instructions are skipped until another type of instruction is executed.
4	HC	The half-carry (HC) flag is set whenever a carry from bit 3 occurs during an operation, or whenever a borrow from bit 4 to bit 3 occurs. This flag is cleared for all other conditions.
5	SK	The skip (SK) flag is set whenever a skip condition occurs, and cleared for all other conditions.
6	Z	The zero (Z) flag is set whenever an operation result equals zero, and cleared for all other conditions.
7		This bit is always zero and is not used.

Arithmetic and Logic Unit [ALU]

The 16-bit ALU performs the arithmetic and logic computations required to execute an instruction. Because of its 16-bit structure, it enables processing of 16-bit data as well as 8-bit data.

MEMORY

The 7811, 7811H, and 78C11 contain 256 bytes of RAM and 4K bytes of ROM (the 7810, 7810H, and 78C10 are ROMless) and feature direct addressing of up to 64K bytes of memory. External memory may be configured up to a maximum of 64K bytes for the 7810, 7810H, and 78C10 and 60K bytes for the 7811, 7811H, and 78C11. The 78C14 contains 256 bytes of RAM and 16K bytes of ROM.

In the 7811, 7811H, and 78C11, addresses 0000H to 0FFFH are reserved for the resident ROM, and addresses FF00H to FFFFH are reserved for the internal RAM. In the 78C14, addresses 0000H-3FFFH are reserved for the resident ROM and FF00H-FFFFH are reserved for the internal RAM. Figure 3-3 shows the memory map.

Figure 3-3. Memory Map

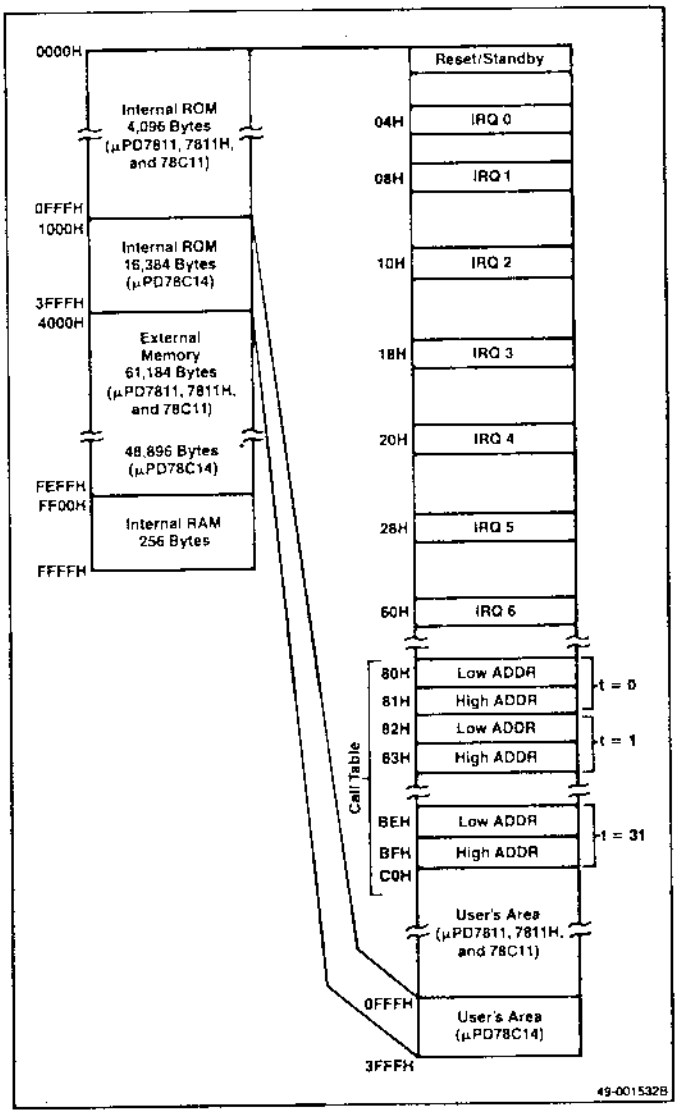


Table 3-3. Flag Operations

Operations						PSW_6	PSW_5	PSW_4	PSW_3	PSW_2	PSW_0
Register/Memory			Immediate		Skip	Z	SK	HC	L1	L0	CY
ADD ADC SUB SBB	ADDW ADCW SUBW SBBW	ADDX ADCX SUBX SBBX	ADI ACI SUI SBI			X	0	X	0	0	X
DADD DADC DSUB DSBB EADD ESUB											
ANA ORA XRA DAN DOR DXR	ANAW ORAW XRAW	ANAX ORAX XRAX	ANI ORI XRI	ANIW ORIW		X	0	•	0	0	•
ADDNC SUBNB GTA LTA DADDNC DSUBNB DGT DLT	ADDNCW SUBNBW GTAW LTAW	ADDNCX SUBNBX GTAX LTAX	ADINC SUINB GTI LTI	 GTIW LTIW		X	X	X	0	0	X
ONA OFFA DON DOFF	ONAW OFFAW	ONAX OFFAX	ONI OFFI	ONIW OFFIW		X	X	•	0	0	•
NEA EQA DNE DEQ	NEAW EQAW	NEAX EQAX	NEI EQI	NEIW EQIW		X	X	X	0	0	X
INR DCR	INRW DCRW					X	X	X	0	0	•
DAA						X	0	X	0	0	X
RLR RLL SLR SLL DRLR DRLL DSLR DSLL						•	0	•	0	0	X
SLRC SLLC						•	X	•	0	0	X
STC						•	0	•	0	0	1
CLC						•	0	•	0	0	0
			MVI A,byte			•	0	•	1	0	•
			LXI H,word			•	0	•	0	1	•
					BIT SK SKN SKIT SKNIT	•	X	•	0	0	•
					RETS	•	1	•	0	0	•
All other instructions						•	0	•	0	0	•

1-Set 0-Reset X-Affected (Set or Reset) •Not Affected

The internal RAM may be disabled by the memory mapping register, allowing this address space to be used for external memory. In the ROMless parts, the entire 64K address space is available for use by external memory, but the internal RAM must be disabled by the memory mapping register in order for this 256-byte address space to be available for external memory.

Interrupt Control

The interrupt structure includes three external interrupts and eight internal hardware interrupts, plus a software (SOFTI) interrupt. These interrupts are organized into seven levels of priority. All interrupts (except $\overline{NMI}$ and SOFTI) are controlled via an interrupt mask flag. Each interrupt category is assigned a specific vector location.

Vector Locations. To process the various interrupts, the following vector locations have been established.

Interrupt Name	Vector Location	Priority
NMI	0004H	IRQ0
INTT0/INTT1	0008H	IRQ1
INT1/INT2	0010H	IRQ2
INTE0/INTE1	0018H	IRQ3
INTEIN/INTAD	0020H	IRQ4
INTSR/INTST	0028H	IRQ5
SOFTI (Soft INT)	0060H	IRQ6

Call Address Table. A 64-byte area from address 0080H to 00BFH holds a table of up to 32 call addresses for use by the CALT instruction.

Reserved Area. Addresses 00H-0BFH are reserved for vector addresses and the call address table and should not be used for data or work areas.

Work Space. Up to 256 bytes of work space can be allocated in any available contiguous memory area using the vector register.

Internal RAM

A 256-byte area of internal RAM is located at addresses FF00H-FFFFH. In standby mode, backup power in the 7810/7811 and 7810H/7811H may be provided for 32 bytes of the internal RAM area at addresses FFE0H-FFFFH.

In the 78C10/78C11, and 78C14 halt and stop modes, the 256-byte RAM data will be maintained if V_{DD} is above 2.5 volts.

External Memory

Up to 64K bytes of external memory may be used with the 7810, 7810H, and 78C10. Up to 60K bytes of external memory at addresses 1000H-FEFFH may be used with the 7811, 7811H, and 78C11. Up to 48K bytes may be used with the 78C14. External memory access is enabled by the $\overline{RD}$, $\overline{WR}$, and ALE signals via port D and port F. Port D multiplexes the low address and data byte and port F outputs the upper 8 address bits.

INPUTS AND OUPUTS

Ports A, B, C, D, and F

The 7811, 7811H, 78C11, and 78C14 contain up to 40 I/O port lines depending on the memory configuration. The 7810, 7810H, and 78C10 contain up to 32 I/O port lines. Each port is 8 bits wide. All data written into the ports by data transfer operations is stored in an output latch, remaining unchanged until new data is written. Since there are no input latches, the input data must be held stable until read.

Serial Interface

A serial interface facilitates communication in a multiprocessor system by allowing the transfer of serial data between different terminals in asynchronous, synchronous, or I/O interface modes.

Analog-to-Digital Converter

An eight-channel, 8-bit successive-approximation type analog-to-digital converter performs high-precision conversion of analog signals into digital format. The conversion values are held in four conversion result registers (CR_0-CR_3). The analog input can be set to either scan mode or select mode.

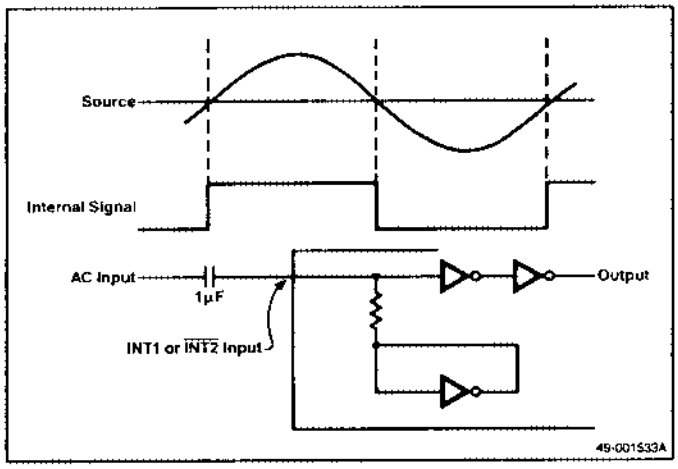
Edge-Sensing Inputs

In addition to providing analog inputs to the A/D converter, lines AN4-AN7 can detect the falling edge of an input signal. When a falling edge is detected, a test flag is set. This flag can be tested by the SKIT or SKNIT instruction.

Zero-Cross Detection, 7810/7811, 7810H/7811H

Special circuits are incorporated for the INT1 and $\overline{INT2}$ lines that allow detection of the zero point (average dc level) being crossed by ac signals on these lines (see figure 3-4). A zero crossing triggers an interrupt. An external capacitor is required as shown in figure 3-4.

Figure 3-4. Zero-Cross Detection



The zero-cross detection circuitry for each line consists of a high-gain self-biasing amplifier that maintains the average dc level of the line. Any level change caused by a rising ac signal crossing above the average dc level is amplified, setting the logic state to one. The logic state of one is held until the falling ac signal crosses below the average dc level. An interrupt is generated when the INT1 line senses a zero crossing from negative to positive (a rising ac signal), or when the INT2 line senses a zero crossing from positive to negative (a falling ac signal).

Since the $\overline{INT2}$ line also functions as the TI (input clock) line in the interval counter, the zero-cross detect pulses on line $\overline{INT2}$ can be used as an input clock source. The zero crossing detection may use the frequency of the 50/60-Hz input power line as a system timing signal.

A primary function of zero-cross detection is the generation of interrupts at the zero-cross point, enabling control of voltage-phase sensing devices such as triacs or silicon controlled rectifiers (SCRs). This feature also applies to shaft-speed measurement and angle measurement. When no external capacitors are connected to the INT1 and $\overline{INT2}$ lines, they function as TTL-compatible input pins.

Zero-Cross Detection, 78C10/78C11 and 78C14

The zero-cross detection for the CMOS 78C10/78C11 and 78C14 is very similar to that of the 7810/7811 and 7810H/7811H. In the CMOS parts, INT1 and $\overline{INT2}$ can be used for zero-cross detection when specified by the zero-cross mode register (ZCM).

Figure 3-5 shows the format for the zero-cross mode register, which controls the self-bias for zero-cross detection. When the ZC_1 and ZC_2 bits are cleared, self-biasing of the zero-cross detection inputs is disabled and the bits function as normal digital inputs. When ZC_1 and ZC_2 are set, self-biasing is enabled and the zero-crossing of an ac signal can be detected by connecting it through an external capacitor to the pin. When ZC_1 or ZC_2 is set, but no external series capacitor is connected, the input functions as a digital input. However, some input load current is required and a suitable external driver circuit must be provided. Also, when the zero-cross detection circuit is activated, power supply current flows constantly, even in the stop modes. This current is relatively small compared to overall consumption in the operating mode. However, the current does become significant in the power-down modes if compared to when the zero-cross detection function is not activated. A reset sets bits ZC_1 and ZC_2 and activates zero-cross detection.

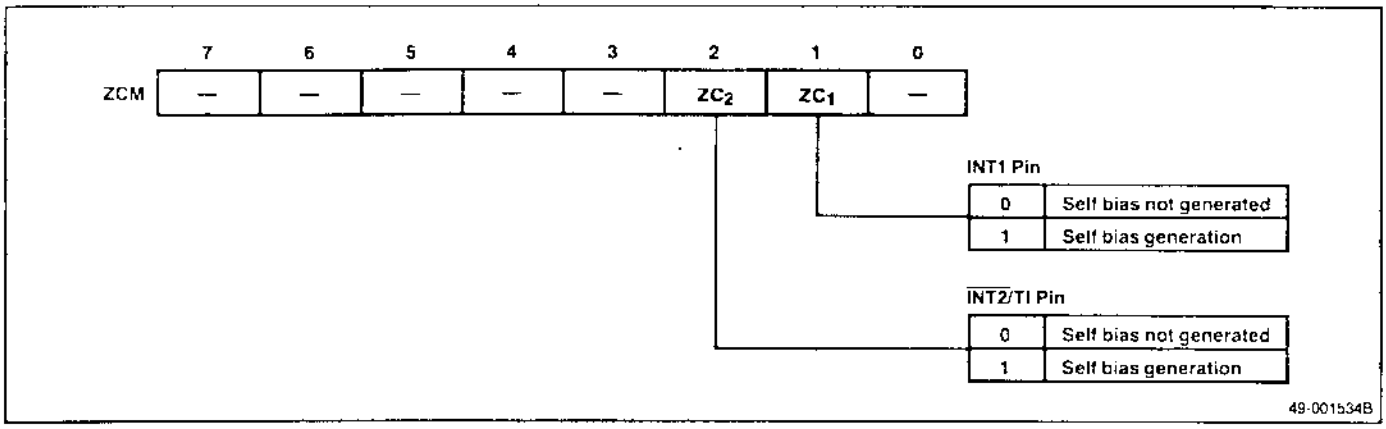
TIMER/EVENT COUNTER

A programmable 16-bit timer/event counter can provide the following functions:

- Interval timer
- Event counter
- Frequency measurement
- Pulse-width measurement
- Programmable frequency and duty cycle waveforms

Two programmable 8-bit interval timers may be used independently or together as a single 16-bit timer. The timer output TO (PC_4) is derived from the output of the TM0 or TM1 comparator or the internal clock (ϕ3).

Figure 3-5. Format of Zero-Cross Detection Register



PORT A

Port A is a programmable 8-bit I/O port. Each line (PA_7-PA_0)contains an I/O buffer and an output latch (see figure 4-1). The mode A register (MA) can designate each line independently as either input or output. A line's output buffer is high impedance when the line is designated as an input line.

Figure 4-2 shows how each bit in the mode A register corresponds to a port A line (MA_0 and PA_0, MA_1 and PA_1, and so on). When a bit in the mode A register is set, the output buffer of the corresponding port A line is disabled and the port line becomes an input line (see figure 4-4). A port A line functions as an output line when the corresponding bit in MA is cleared, enabling the output buffer (see figure 4-3). A reset initializes MA to all ones, establishing port A as an 8-bit input port.

After reset, the contents of the port A output latches are undefined.

Figure 4-1. Port A Block Diagram

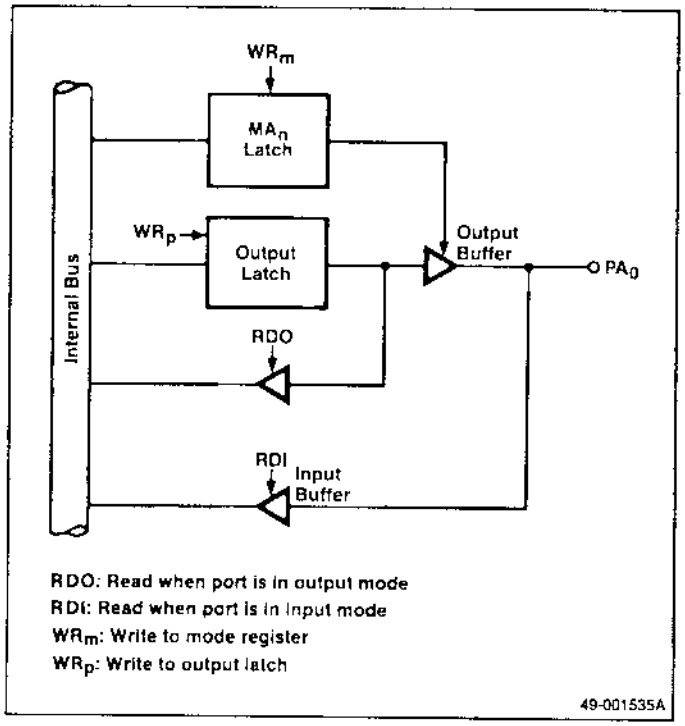


Figure 4-3. Block Diagram of a Port A Output Line

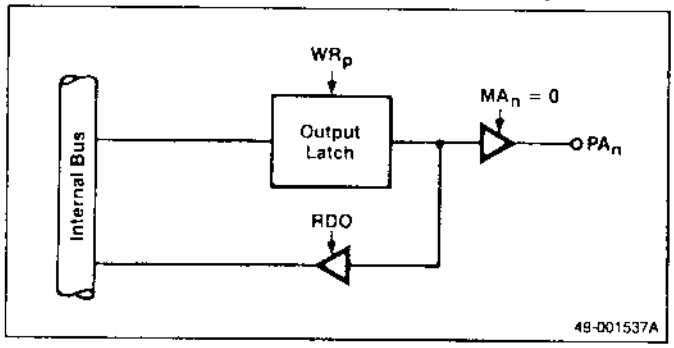


Figure 4-4. Block Diagram of a Port A Input Line

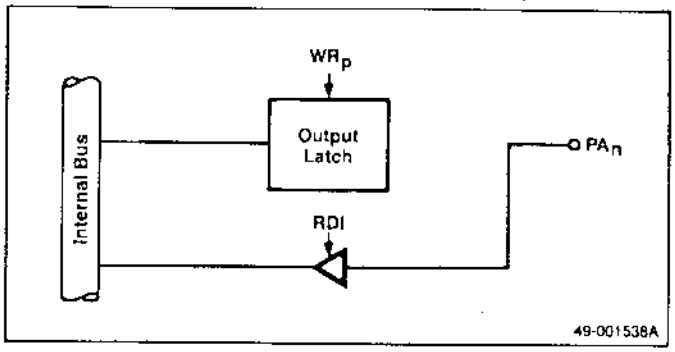
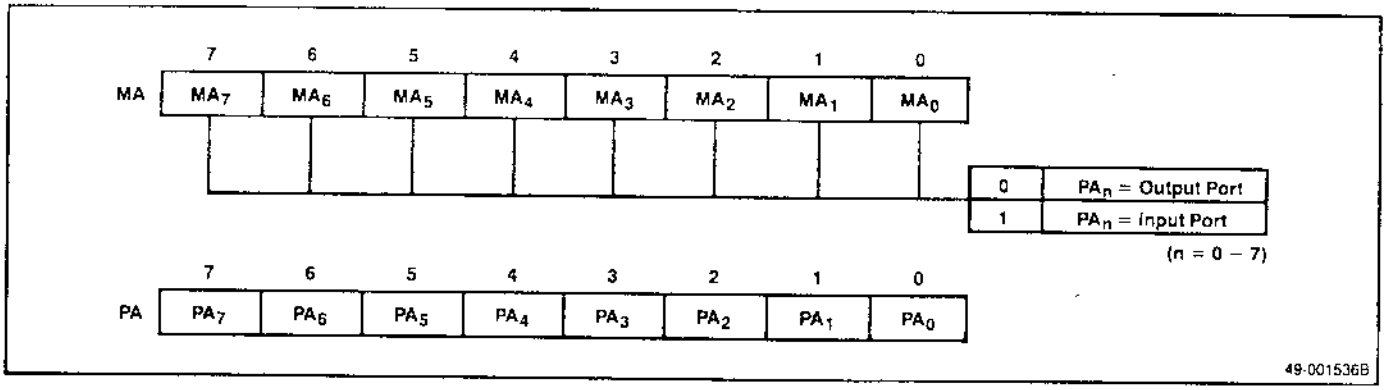


Figure 4-2. Mode A Register (MA)



Output Port Operation

When a line has been designated for output, the output buffer is enabled. Data can be exchanged between the output latches and the accumulator by using a data transfer instruction. When a read operation is executed on an output line, the contents of the output latch (not the level at the pin) is transferred to the accumulator. The bits of an output latch can also be set or cleared using an arithmetic or logic operation instruction without having to use the accumulator; for example, the instruction ORI PA,byte. Once data is written into an output latch, it remains unchanged until new data is written into the latch.

Input Port Operation

When a port A line has been designated for input, the contents of the line can be loaded into the accumulator by using a data transfer instruction. The contents of the line can also be tested directly by an arithmetic or logic operation instruction without having to use the accumulator.

Data transfer instructions address port A as a single 8-bit entity, operating on all port A lines simultaneously. In port A write operations, data is written into the port A output latches, regardless of whether or not a line has been designated for input or output. Even though a line designated for input has had data written into its output latch from the accumulator as a result of a write instruction, the output buffer is held at a high-impedance state, prohibiting the data in the output latch from being available at the pin. Since there is no input latch for any port line, the input data must be held stable when a data transfer or bit test instruction is executed.

Summary

Port A read and write instructions operate on port A as a single 8-bit entity. A read instruction (e.g., MOV A, PA) loads the logic level at the pin of all input lines or the contents of the output latches of all output lines into the accumulator. A write instruction (e.g., MOV PA,A) writes data from the accumulator to the output latches, regardless of whether a port line has been designated for input or output. However, since input lines are three-stated by RDO inactive (see figure 4-1), and since the output buffer is disabled, the data in the output latches is not available at the pins.

PORT B

This programmable 8-bit I/O port (PB_7-PB_0) functions exactly like port A. Each line can be independently placed in input mode or output mode by using the mode B register (MB). Each line contains an I/O buffer and an output latch. Setting a bit in the mode B register places the corresponding port B line in input mode (the output buffer goes to a high-impedance state), and clearing the bit places the line in output mode (the output buffer is active). A reset initializes the mode B register to all ones, establishing port B as an 8-bit input port. Figure 4-5 shows the mode B register format.

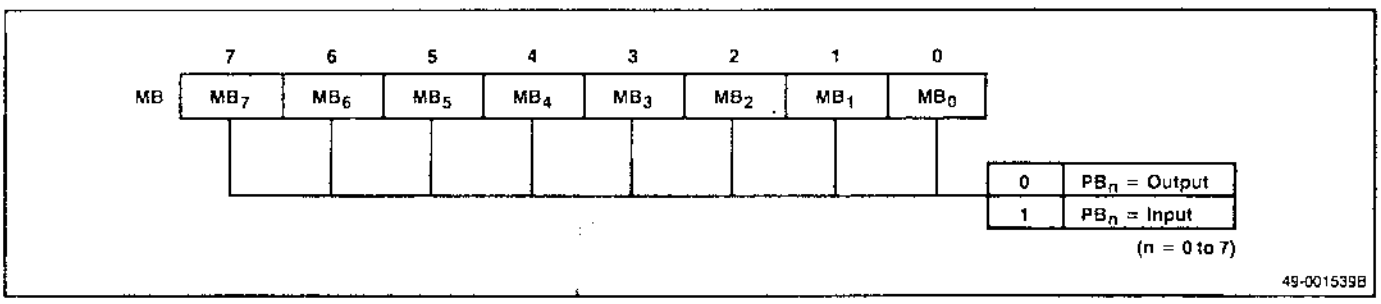
PORT C

The individual bits of port C (PC_7-PC_0) can be enabled in either of two modes: port mode and control mode. In port mode, port C has the same port characteristics as ports A and B. In control mode, port C provides special control signals. The mode control C register (MCC) designates which bits of port C are port lines and which are control lines (MCC_0 corresponding to PC_0, MCC_1 corresponding to PC_1, and so on). When a bit in the MCC register is set, the corresponding port C line is placed in control mode. When the bit is cleared, the port C line is placed in port mode (see table 4-1). A reset initializes the MCC register to all zeros, placing all port C lines in port mode.

Table 4-1. Port C Mode Enable

	$MCC_n = 0$	$MCC_n = 1$
PC_n (n = 0-7)	Port mode	Control mode

Figure 4-5. Mode B Register (MB)



Port Mode

In port mode, port C is a programmable I/O port with each line independently used for input or output as determined by the mode C register (MC). In the MC register, bits corresponding to the individual port C lines (MC_0 and PC_0, MC_1 and PC_1, and so on) are set to designate an input line or cleared to designate an output line (see figure 4-6). A reset initializes the MC register to all ones, establishing port C as an 8-bit input port. In port mode, the operational features of port C are identical to those of ports A and B.

Control Mode

In control mode, each port C line is used for a specific control signal. In the mode control C register (MCC), bits corresponding to each port C line independently place the respective line in port mode when cleared, or place the line in control mode when set. The format of the MCC register and the function of each line in control mode is shown in figure 4-7.

Figure 4-6. Mode C Register (MC)

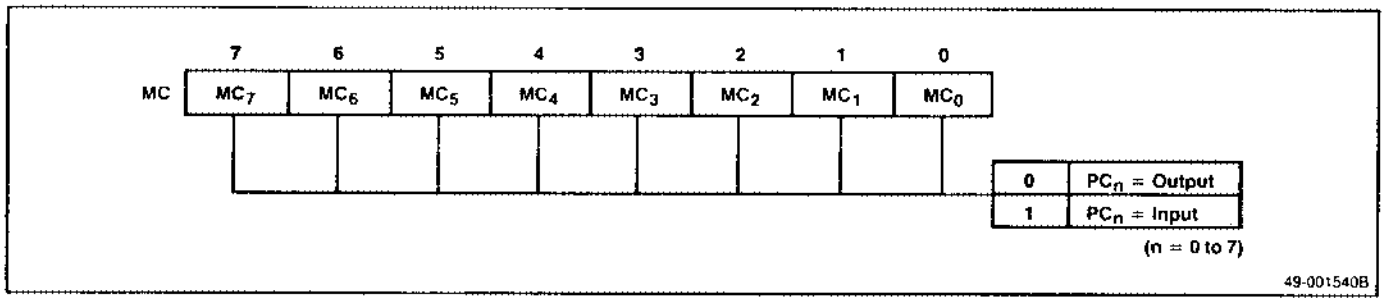
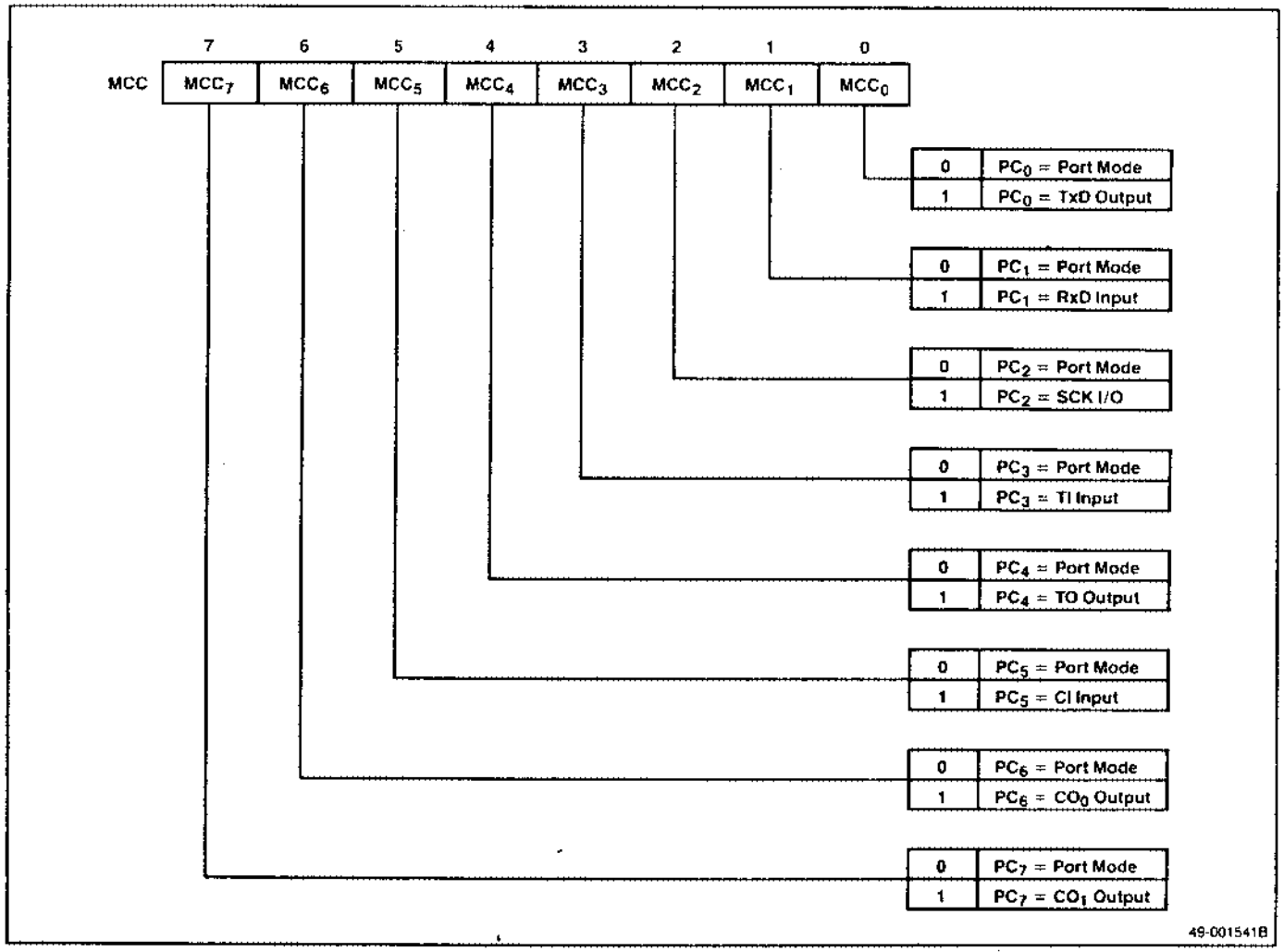


Figure 4-7. Mode Control C Register (MCC)



The status of the control signals on the individual port C lines can be read or tested using an 8-bit port C read or test instruction when the respective line (figure 4-8) is in control mode, as follows:

When PC_n is used to output control signals, the signal status is read into the accumulator or tested using the port C read or test instruction, respectively.

(1) If $MC_n = 1$, the control input buffer is enabled and the output signal at the PC_n pin is read or tested.

(2) If $MC_n = 0$, the internal control signal is read or tested.

When PC_n is used to input control signals, $MC_n = 1$. The control intput buffer is enabled and the status of the control signal at the PC_n pin can be read into the accumulator or tested using the port C read or test instruction, respectively.

PORT D

In the 7811, 7811H, 78C11, and 78C14, port D (PD_7-PD_0) may be enabled as either an 8-bit I/O port or as eight multiplexed address/data lines. Port D always provides eight multiplexed address/data lines in the 7810, 7810H, and 78C10.

Memory Mapping Register [MM]

This 8-bit register specifies the external memory capacity used with the ROM parts and configures ports D and F accordingly. This register also selectively enables/disables internal RAM access. The format of the memory mapping register (MM) is shown in figure 4-9. Bits 2-0 specify the configuration of ports D and F and designate port D as either input or output. When the RAE bit is set, internal RAM access is enabled; when cleared, access is disabled. Bits 7-4 are undefined. A reset clears bits 2-0, but the previous state of the RAE bit is maintained. For power-up or reset conditions, the RAE bit should be set according to user requirements.

Figure 4-8. Port C Control Signal Read Configuration

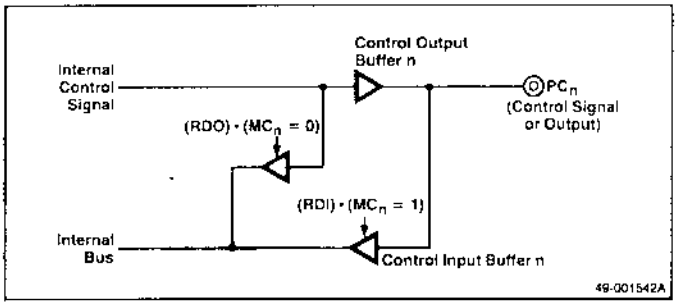
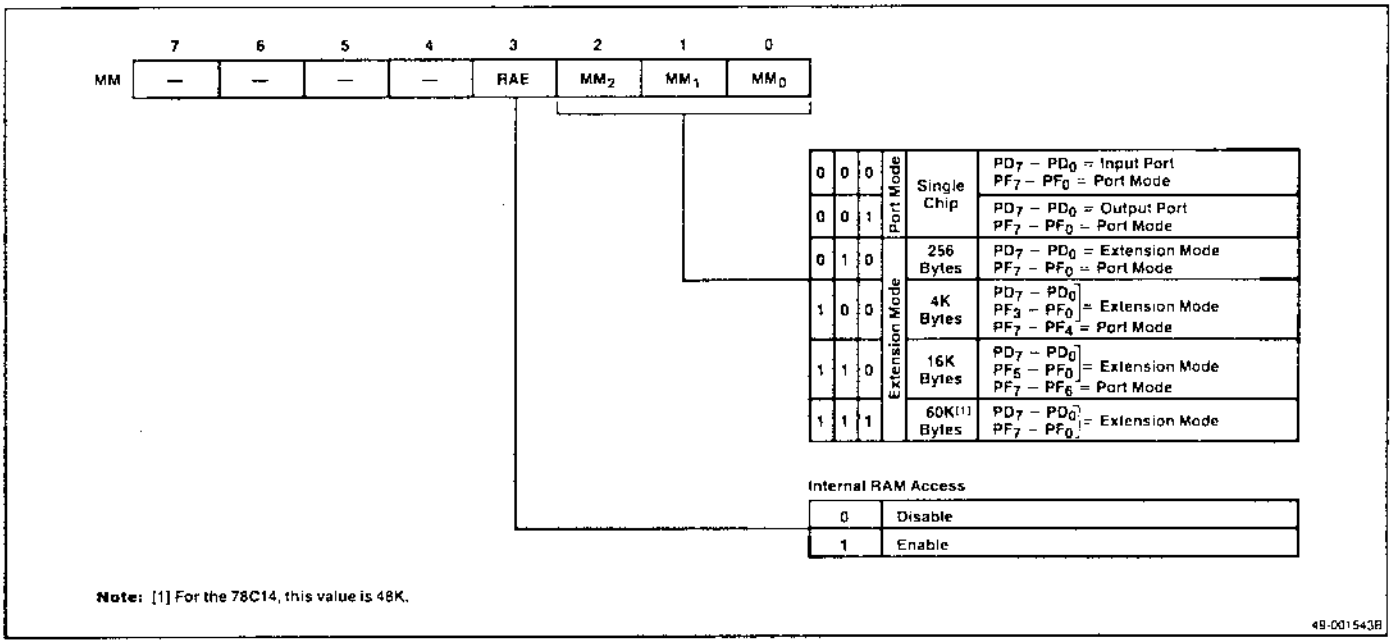


Figure 4-9. Memory Mapping Register (MM)



Port D Operation; 7811/7811H, 78C11, 78C14

In the 7811, 7811H, 78C11, and 78C14, the function of port D is based on the amount of external memory used, as determined by the MM register. When MM_2-$MM_1 = 00$ (no external memory), port D functions as a programmable 8-bit I/O port. MM_0 places port D in input mode when cleared or output mode when set.

Port D operates only as a byte-wide input or output port; the individual lines cannot be independently selected as input or output lines. Each port D line contains an I/O buffer and an output latch. Except for enabling of the lines as input or output on a byte-wide basis, port D operates identically to ports A and B.

When any external memory is used (MM_2-$MM_1 = \neq 00$), port D functions as eight multiplexed address/data lines. The 8 lower-order bits of the address are output in the first phase (T1) of an external memory access cycle, and 8-bit data is transferred in the second and third phases (T2 and T3).

Port D Operation; 7810/7810H, 78C10

In the 7810, 7810H, and 78C10, PD_7-PD_0 are enabled as multiplexed address/data lines and operate identically to when the 7811, 7811H, 78C11, and 78C14 parts are used with external memory.

PORT F

Lines PF_7-PF_0 can function as independent input and output port lines or as dedicated address lines. The external memory configuration is determined by the memory mapping register (MM) in the ROM part and by the MODE0 and MODE1 signals in the ROMless part.

Port F Operation, ROM Parts

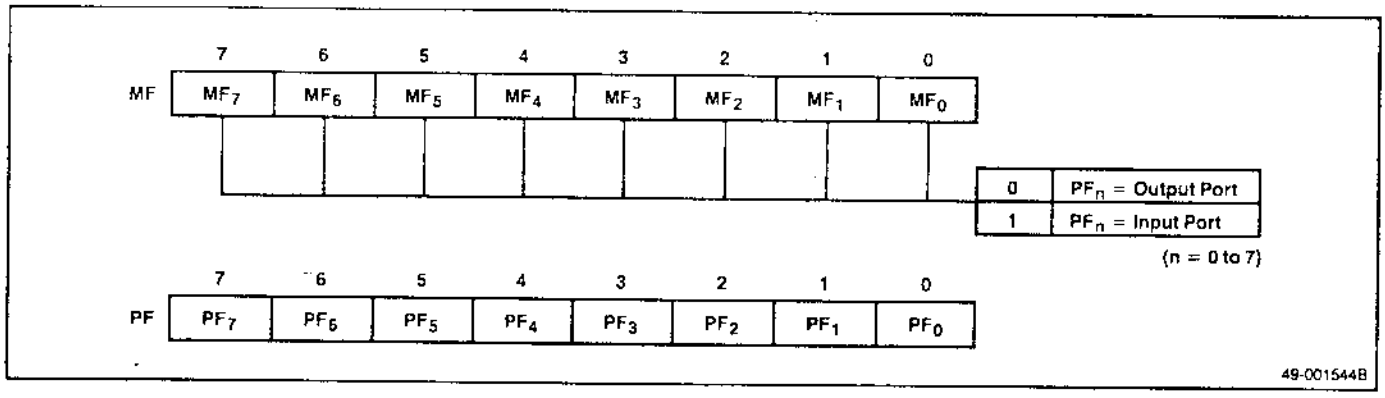
When 256 or fewer bytes of external memory are used, lines PF_7-PF_0 are enabled as a programmable 8-bit I/O port whose individual lines can be independently placed in input or output mode by the mode F register (MF). In the MF register, bits corresponding to each port F line place the respective line in input mode when set or output mode when cleared (see figure 4-10). A reset initializes the MF register to all ones, establishing port F as an 8-bit input port (all lines at high impedance). The port operation of PF_7-PF_0 is identical to that of ports A and B.

When the amount of external memory, as determined by the MM register, is between 257 bytes and 4K bytes, lines PF_3-PF_0 are enabled as dedicated address lines, and PF_7-PF_4 are enabled as port I/O lines. When the amount of external memory is between 4097 bytes and 16K bytes, PF_5-PF_0 are enabled as dedicated address lines and PF_7-PF_6 are enabled as port I/O lines. When the amount of external memory exceeds 16K bytes, PF_7-PF_0 are enabled as dedicated address lines. Table 4-2 shows the relationship between the MM register for the ROM parts and the function of the PF_7-PF_0 lines.

Table 4-2. Relationship Between 7811, 7811H, 78C11, and 78C14 Memory Mapping Register (MM) and Function of PF_7-PF_0 Lines (MODE1 = 1, MODE0 = 0)

MM_2-MM_0	Lines	Function	External Memory, Bytes
010	PF_7-PF_0	Port	0 to 256
100	PF_7-PF_4 PF_3-PF_0	Port AB_{11}-AB_8	257 to 4K
110	PF_7-PF_6 PF_5-PF_0	Port AB_{13}-AB_8	4097 to 16K
111	PF_7-PF_0	AB_{15}-AB_8	>16K to 60K max (48K max in 78C14)

Figure 4-10. Mode F Register (MF)



Port F Operation, ROMless Parts

In the ROMless parts, the MODE1 and MODE0 signals specify the external memory configuration (see table 4-3) and determine the function of the PF_7-PF_0 lines based on the amount of external memory used. When MODE1, MODE0 = 0, 0, a maximum of 4K bytes of external memory is specified; PF_3-PF_0 are enabled as dedicated address lines, and PF_7-PF_4 are enabled as port I/O lines. A designation of a maximum of 16K of external memory (MODE1, MODE0 = 0, 1) enables PF_5-PF_0 as dedicated address lines and PF_7-PF_6 as port I/O lines. When more than 16K bytes is specified (MODE1, MODE0 = 1, 1), PF_7-PF_0 are enabled as dedicated address lines. The port operation of PF_7-PF_0 for the ROMless parts is identical to that of the ROM parts.

Table 4-3. Values of MODE1 and MODE0 Signals and Function of PF_7-PF_0 Lines in 7810, 7810H, 78C10

MODE1	MODE0	Lines	Function	External Memory, Bytes
0	0	PF_7-PF_4 PF_3-PF_0	Port AB_{11}-AB_8	4K max
0	1	PF_7-PF_6 PF_5-PF_0	Port AB_{13}-AB_8	16K max
1	1	PF_7-PF_0	AB_{15}-AB_8	>16K

TIMING

Timing waveforms for opcode fetch, memory read, and memory write operations are shown in figures 4-11 to 4-13, respectively. One timing state is defined as three cycles of the crystal clock as shown in figure 4-11. Timing states are represented by T_n, where n is the timing state number within a given operating cycle; for example, T3 = the third timing state of a cycle.

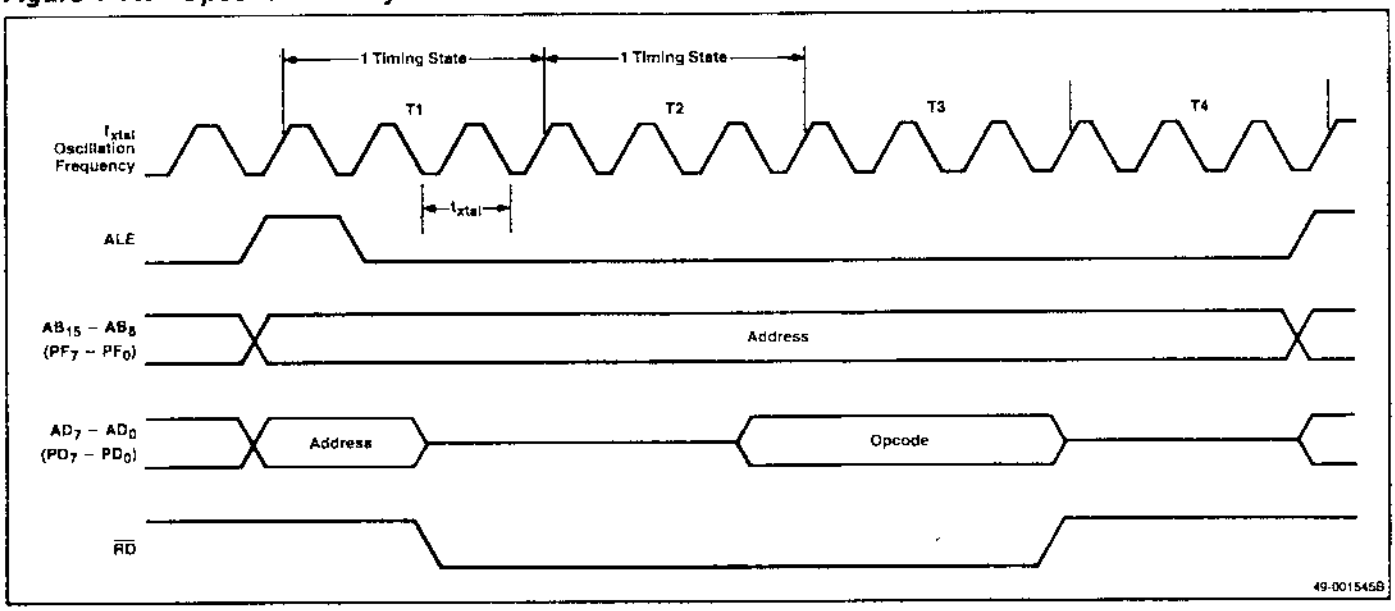
An ordinary read/write operation is executed in three timing states (9 clock cycles), whereas an opcode fetch operation requires four timing states (12 clock cycles). When external memory is not being accessed, the $\overline{RD}$ and $\overline{WR}$ signals remain high throughout the memory access.

Opcode Fetch Timing

An opcode fetch cycle (figure 4-11) consists of four timing states (T1-T4). The first two states (T1 and T2) are required for the program memory read operation and the last two (T3 and T4) are required for decoding. When accessing external memory, the higher-order 8 bits of the external memory address are continuously output on lines PF_7-PF_0 during T1-T4. Since lines PD_7-PD_0 are enabled as multiplexed address/data lines when external memory is used, the lower-order 8 bits of the external memory address are output on these lines during T1 and PD_7-PD_0 go to a high impedance state at the end of T1. Because the address bits driven out on PD_7-PD_0 are not output continuously, the lower-order address must be latched externally.

The ALE signal is used as a strobe for the external latching of A_7-A_0 and is output during T1 of every machine cycle. The $\overline{RD}$ signal is asserted midway through T1 and continuously until the start of T4.

Figure 4-11. Opcode Fetch Cycle



Memory Read Timing

A memory read operation (figure 4-12) consists of three timing states (T1-T3). Timing for a memory read operation with respect to port D and port F lines and the $\overline{RD}$ and ALE signals is the same as that of an opcode fetch operation except that there is no T4 in a memory read operation.

Memory Write Timing

A memory write operation (figure 4-13) consists of three timing states (T1-T3). Timing for memory addressing and address latching is the same as that for a memory read operation. The write data is output on lines PD_7-PD_0 from the beginning of T2 to the end of T3.

Figure 4-12. Memory Read Cycle

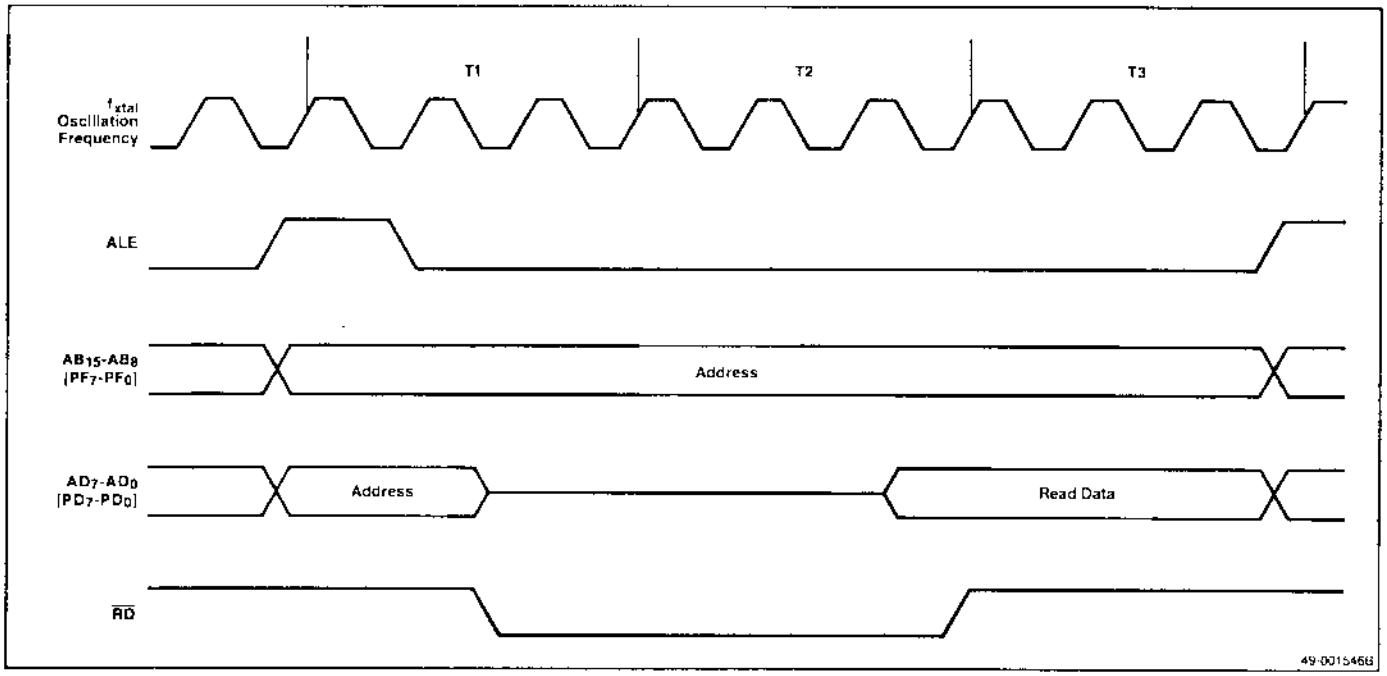
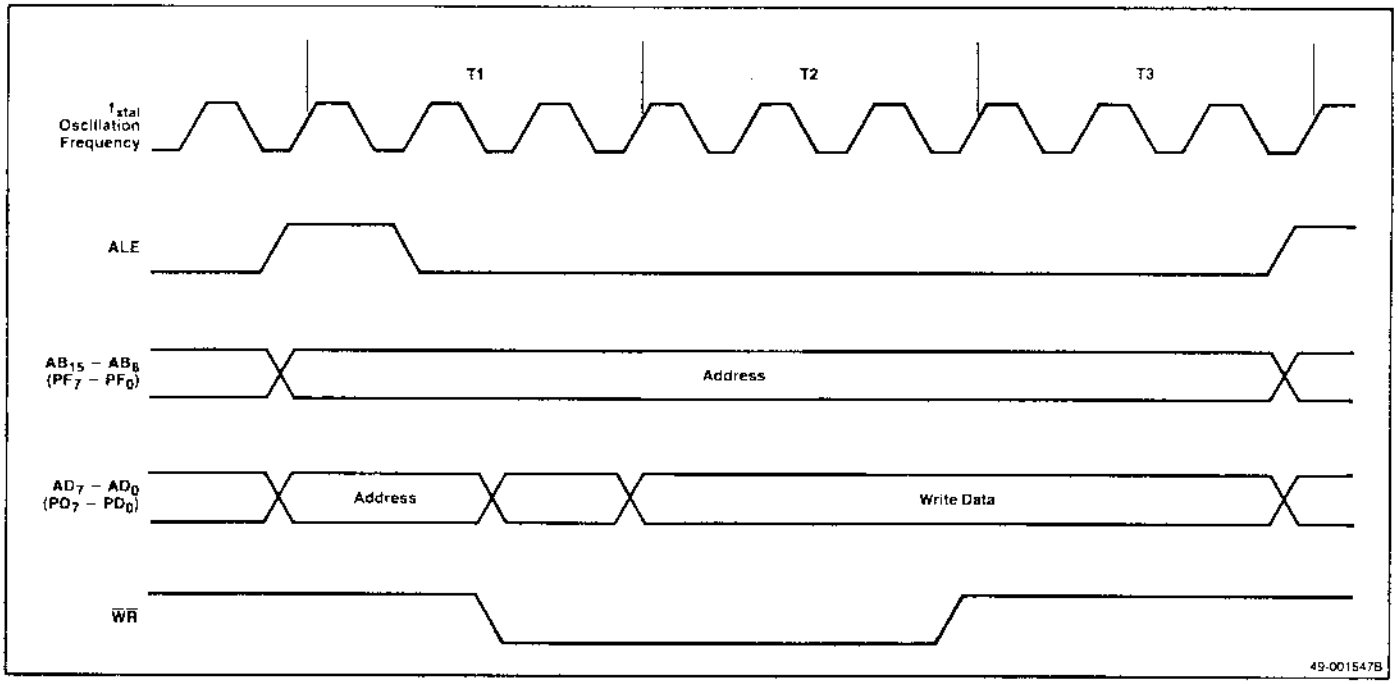


Figure 4-13. Memory Write Cycle



MEMORY EXPANSION

ROM Parts

Figure 4-14 shows the memory mapping of the various external memory configurations. Care must be used when implementing the 4K or 16K addressing mode in the ROM parts. In 4K mode, the upper 4 bits of the program counter (PC) are not output to port F (PF_7-PF_4). When the PC contains a value between 1000H and 1FFFH, the external address put on the address bus will be between 0 and FFFH. The address decoding hardware must reflect this.

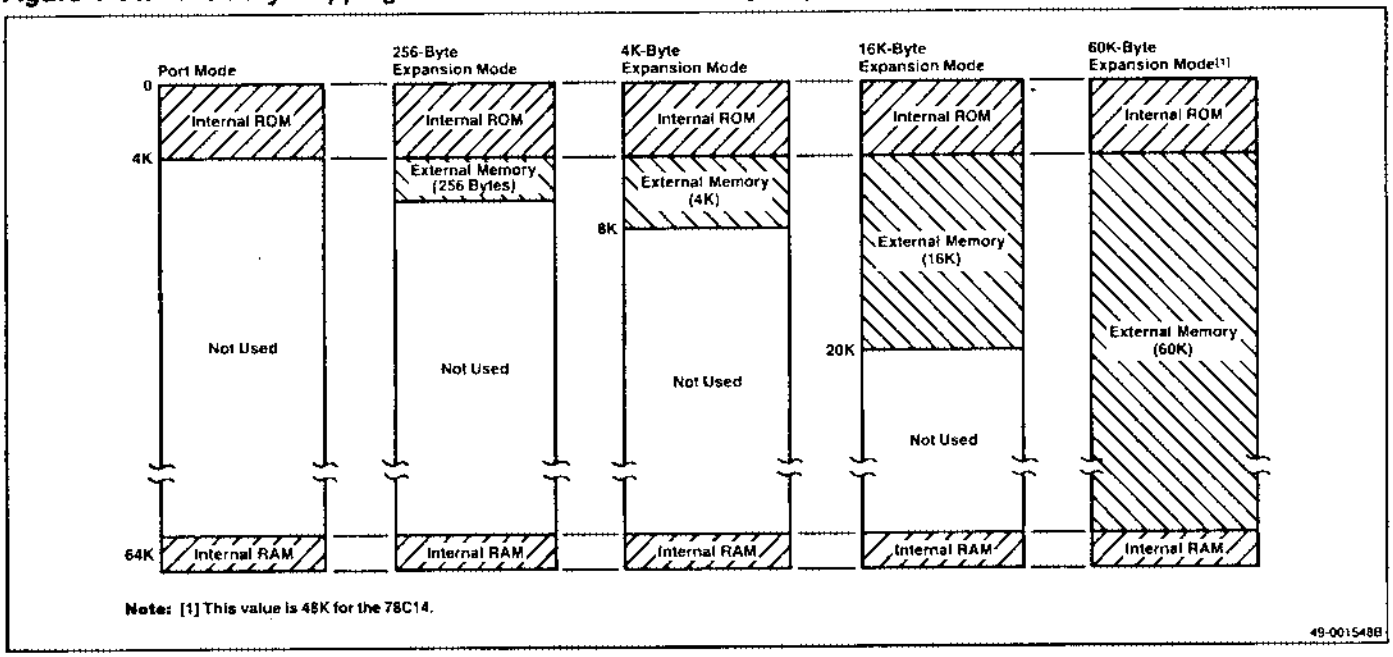
In 16K mode, the upper 2 bits of the PC are not output. Physical addressing of external memory must be made compatible with software addressing. Table 4-4 shows how the external address differs from the PC contents in the 16K mode. Note that when the contents of the PC are between 1000H and 4FFFH, the external memory addresses are between 0000H and 3FFFH and no two addresses are the same.

However, when the contents of the PC are between 5000H and 8FFFH, the external memory addresses are the same as when the PC was between 1000 and 4FFFH. Therefore, different PC values can generate external memory addresses that are the same. This must be taken into account to avoid incorrectly accessing the external memory. This situation also exists for the 4K mode. In the 256-byte and 60K-byte modes (or 48K-byte in the 78C11), this is not applicable.

Table 4-4. 16K Mode Addressing

PC		External Address
0-FFF	(0-4095)	Not output to address bus
1000-1FFF	(4096-8191)	1000-1FFF
2000-2FFF	(8192-12,287)	2000-2FFF
3000-3FFF	(12,288-16,363)	3000-3FFF
4000-4FFF	(16,364-20,479)	0000-0FFF
5000-5FFF	(20,480-24,575)	1000-1FFF
6000-6FFF		2000-2FFF
7000-7FFF		3000-3FFF
8000-8FFF		0000-0FFF
9000-9FFF		1000-1FFF
A000-AFFF		2000-2FFF
B000-BFFF		3000-3FFF
C000-CFFF		0000-0FFF
D000-DFFF		1000-1FFF
E000-EFFF		2000-2FFF
F000-FFFF		3000-3FFF

Figure 4-14. Memory Mapping for Various External Memory Capacities of ROM Parts



Note: [1] This value is 48K for the 78C14.

49-001548B

ROM Parts, Memory Expansion Example

Figure 4-15 shows an example of a 4K ROM expansion. The internal memory of the ROM part is expanded by the use of a 4K external ROM. Figure 4-16 shows the setup of the memory mapping register (MM) for that configuration.

ROMless Parts

Like the ROM parts, in addition to the internal RAM, the memory capacity of ROMless parts can be expanded using external memory. Since they do not have internal ROM, they can be used with up to 64K of external memory. The external memory capacity is determined by the MODE0 and MODE1 signals rather than by the memory mapping register as in the ROM parts. Table 4-5 shows the combinations of the MODE1 and MODE0 signals and the corresponding memory configurations. Port D is enabled as an 8-bit multiplexed address/data bus and PF_7-PF_0 can be enabled as higher-order address lines.

Table 4-5. Combinations of MODE0 and MODE1 Signals and Corresponding Memory Configuration for ROMless Parts

Mode	MODE1	MODE0	External Memory
4K bytes access	0	0	4K bytes
16K bytes access	0	1	16K bytes
64K bytes access	1	1	64K bytes

Figure 4-15. Example of 4K ROM Expansion of ROM Part Memory

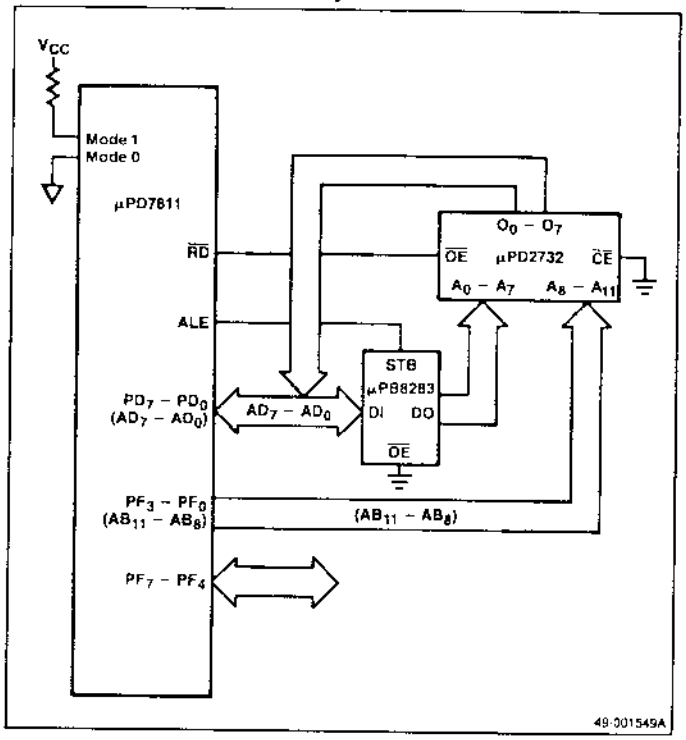
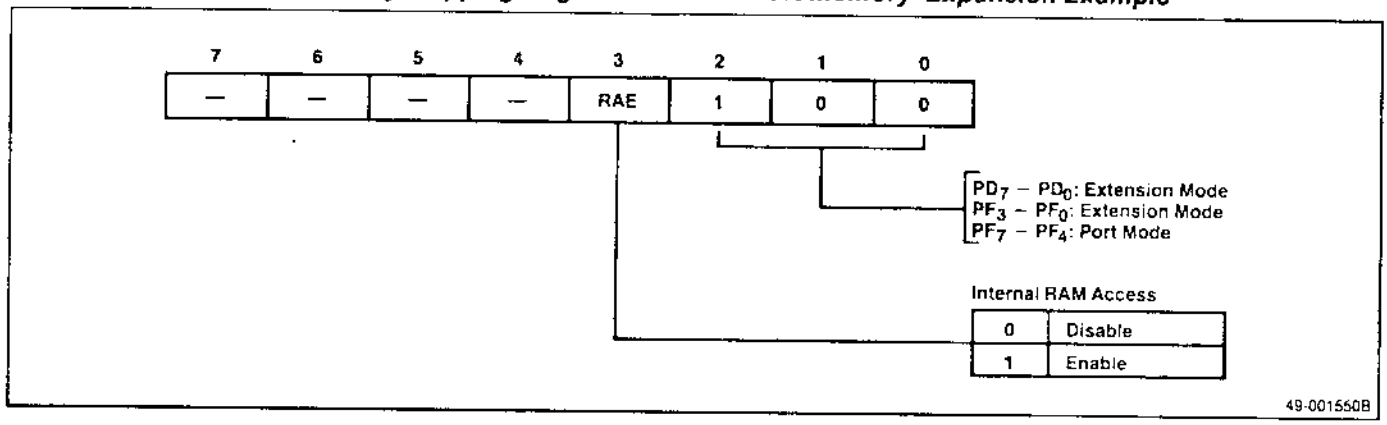


Figure 4-16. Setup of Memory Mapping Register for ROM Part Memory Expansion Example



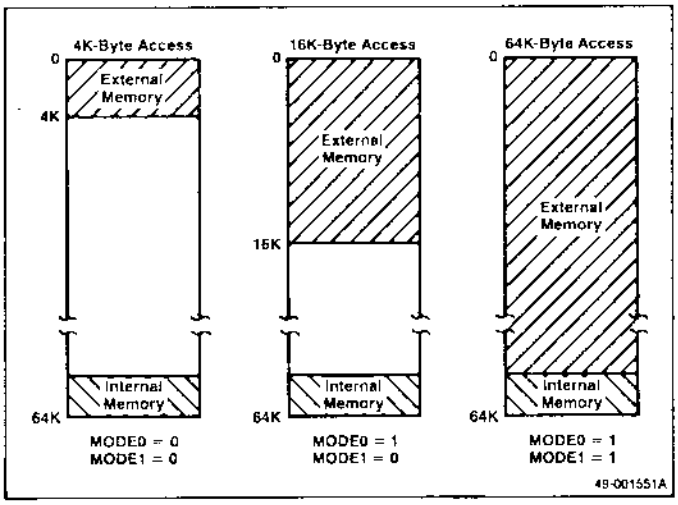
Since external memory capacity is determined by the MODE0 and MODE1 signals in the ROMless parts, bits 2-0 in the MM register should be cleared. The RAE bit and reset operations in the MM register for the ROMless parts are the same as for the ROM versions. Memory mapping for various memory configurations is shown in figure 4-17.

Port Emulation Mode

Port emulation mode (PEM) makes ports D and F available for I/O, even though the processor may be using all or parts of these ports to access external memory. PEM is implemented by adding extra external hardware and utilizes the MODE1 ($\overline{M1}$) and MODE0 ($\overline{IO}$) signals as timing signals.

Refer to the μPD7810/11, 7810H/11H, 78C10/C11/C14 Applications Manual for details.

Figure 4-17. Memory Mapping in ROMless Parts for Various Memory Configurations



Circuit Operation

The interval timer circuit consists of two 8-bit interval timers that can be used independently or cascaded in a 16-bit timer configuration (see figure 5-1). The 8-bit interval timers, TIMER0 and TIMER1, and a timer flip-flop are controlled by the timer mode register (TMM). Each interval timer consists of an 8-bit up-counter, an 8-bit comparator, and an 8-bit timer register (TM0 and TM1, respectively).

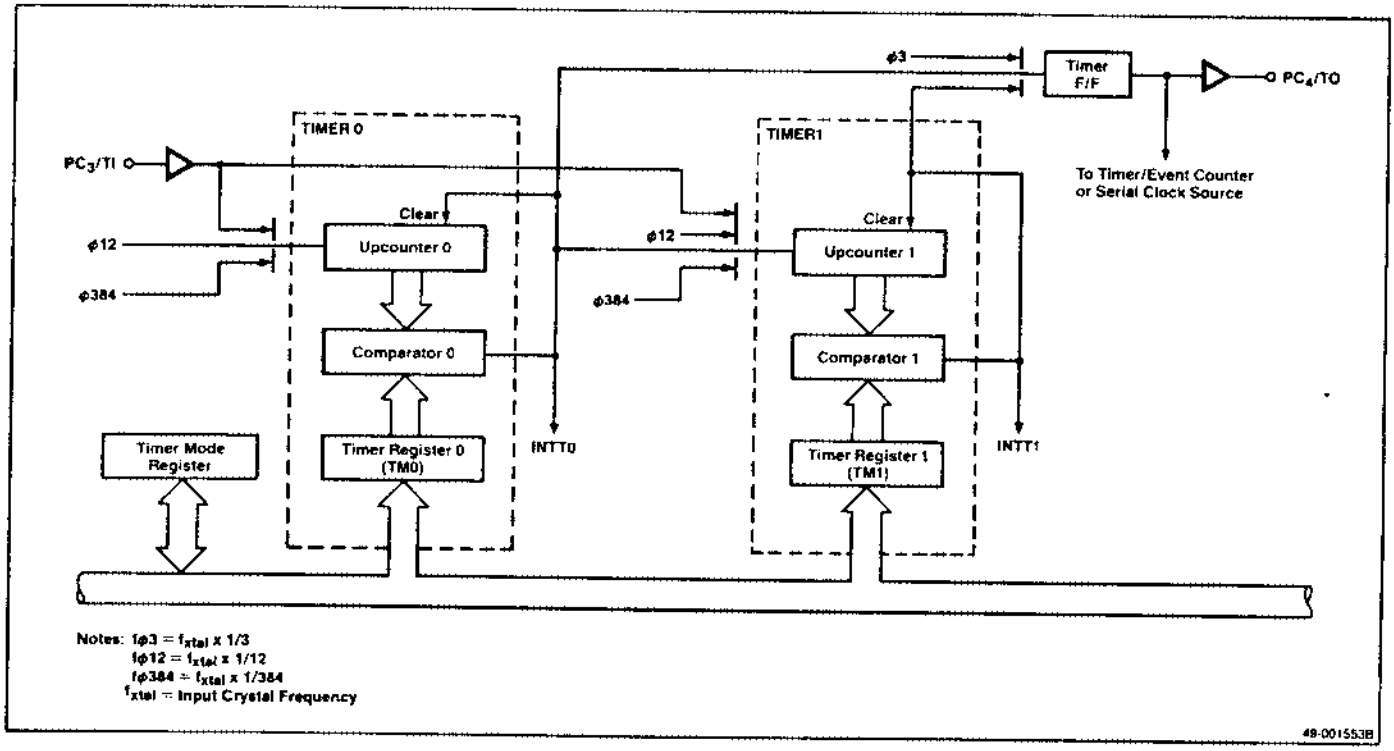
Upcounter. This counter is incremented by each input clock pulse as designated by the timer mode register.

Comparator. This circuit compares the contents of the upcounter and the timer register. The upcounter is cleared to 00H when the contents are equal, and corresponding interrupt requests (INTT0, INTT1) are generated. The interrupts can be masked and the interrupt flags can be tested by skip instructions.

Timer Register. These 8-bit registers, TM0 and TM1, contain the count for the respective interval timer.

Timer Flip-Flop. The output of this flip-flop is inverted by either a match signal from the timer registers (indicating that the contents of the respective register and the upcounter are identical) or by an internal clock pulse ($\phi 3$). The flip-flop output is a square wave one-half the frequency of the input signal. The signal can be sent to the TO (PC_4) pin. It can be used as a clock input to the timer/event counter if so designated by the timer/event counter mode register (ETMM), or it may be used as the serial clock timing source if so designated by the serial mode register (SMH).

Figure 5-1. Block Diagram of Interval Timer Circuit



Timer Mode Register

This 8-bit register (TMM) designates the operation mode of each of the interval timers and the timer flip-flop. The $\overline{\text{RESET}}$ input initializes the timer mode register to FFH, clears the upcounters to 00H, resets the timer flip-flop, and disables the timers. Referring to figure 5-2, which shows the format of the timer mode register, the function of each bit is described as follows.

TF_1-TF_0. These bits control the timer flip-flop. If TF_1-TF_0 = 11, the flip-flop is reset (no output). If TF_1-TF_0 ≠11, then the bits determine the input clock source of the flip-flop.

$CK0_1$-$CK0_0$. These bits select a clock source for TIMER0. They select an internal clock (ϕ12 or ϕ384) or an external clock (TI), or they disable TIMER0.

TS_0. This bit controls the upcounter for TIMER0. It determines whether the upcounter is to be incremented or cleared.

$CK1_1$-$CK1_0$. These bits select a clock source for TIMER1. They select an internal clock (ϕ12 or ϕ384) or an external clock (TI), or they increment TIMER1 when upcounter 0 = TM0.

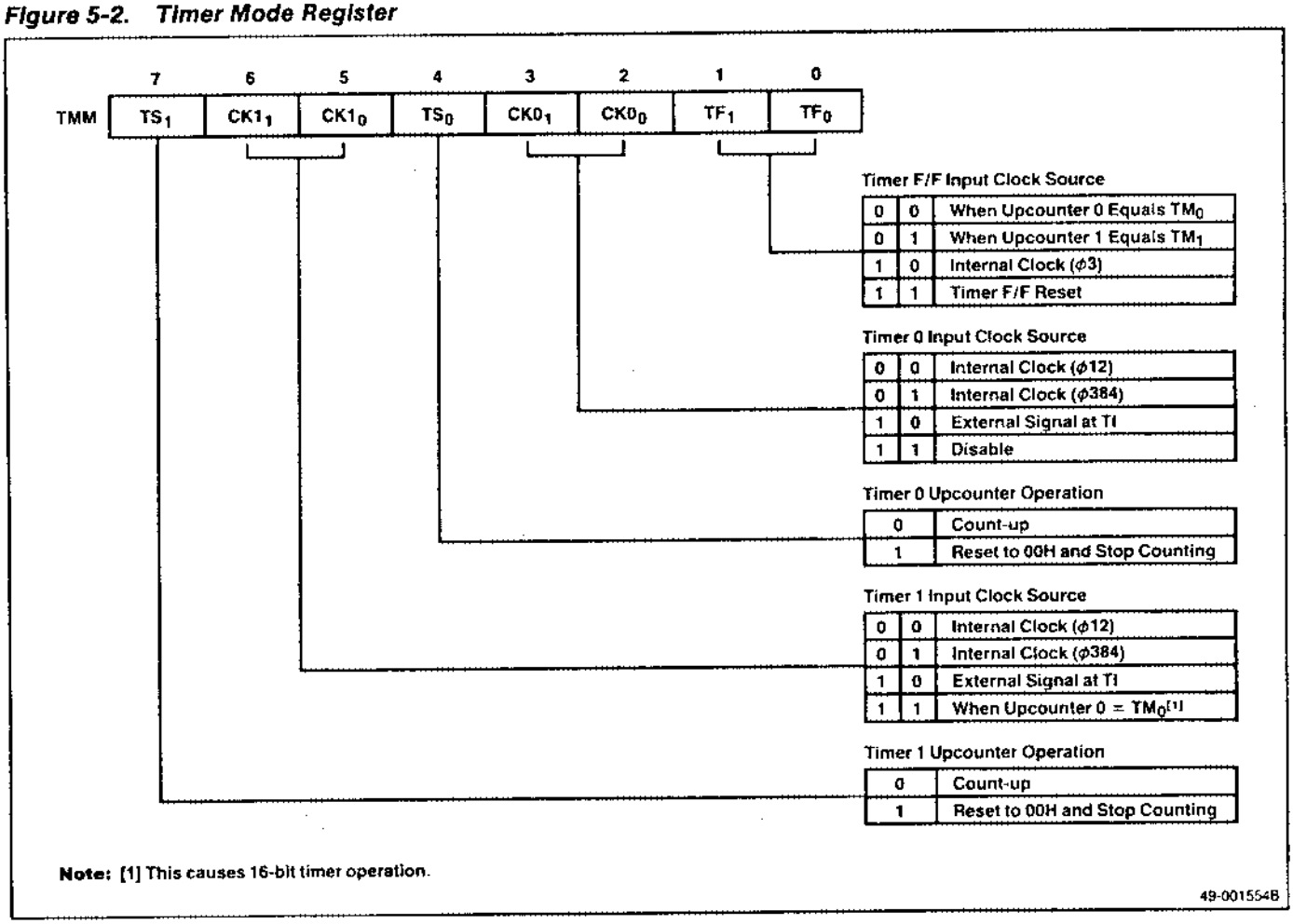
TS_1. This bit controls the upcounter for TIMER1. It determines whether the upcounter is to be incremented or cleared.

Clock Sources

The pair of 8-bit interval timers choose clock sources according to the contents of the timer mode register, as follows.

Internal Clock [ϕ12]. The designation of this clock source for the upcounter enables the timer to output intervals of from 1 to 256 times $12/f_{XTAL}$. With a 12-MHz crystal, intervals range from 1 to 256 μs with a resolution of 1 μs.

Figure 5-2. Timer Mode Register



Note: [1] This causes 16-bit timer operation.

49-001554B

Internal Clock [ϕ384]. The designation of this clock source for the upcounter enables intervals of 1 to 256 times $384/f_{XTAL}$. With a 12-MHz crystal, intervals range from 32 μs to 8.19 ms with a resolution of 32 μs.

External Clock. The designation of an external clock source (TI input) for the upcounter enables intervals based on the frequency of the external clock source. The maximum frequency at this input is 1.25 MHz for all parts except the 7810/7811. The 7810/7811 parts can accept an input frequency of 1.0 MHz.

Timer Operation

The interval timer begins operation when the timer register (TM0 or TM1) is loaded with an interval time value and the timer mode register (TMM) is loaded with operating data.

The upcounter is incremented by each clock pulse. The comparator continuously compares the contents of the upcounter with that of the timer register, and when the values are equal, an interrupt request (INTT0 or INTT1) is generated. The upcounter is then cleared to 00H and resumes counting. The count value contained in the timer register controls the interrupt rates.

If the timer flip-flop uses the TIMER0 or TIMER1 output for its clock source, the contents of the timer flip-flop are inverted when the contents of the upcounter match the contents of the timer register. This outputs a square wave on the TO line. The pulse width of the square wave is equal to the interval contained in the timer register. If ϕ3 is selected as the timer flip-flop clock source, the output at TO is a square wave of one-half the frequency of ϕ3. Timer interrupt requests can be unmasked by clearing bits MKT0 and MKT1 in the interrupt mask register.

16-Bit Interval Timer

TIMER0 and TIMER1 can be cascaded creating a 16-bit timer using the match signal from TIMER0 to increment the TIMER1 upcounter. With a 12-MHz crystal, TIMER0 and TIMER1 together can generate an interval of up to 65 ms using the ϕ12 clock or up to 2.1 seconds using the ϕ384 clock. With a 15-MHz crystal, the intervals are 52.4 ms to 1.677 seconds.

ORGANIZATION

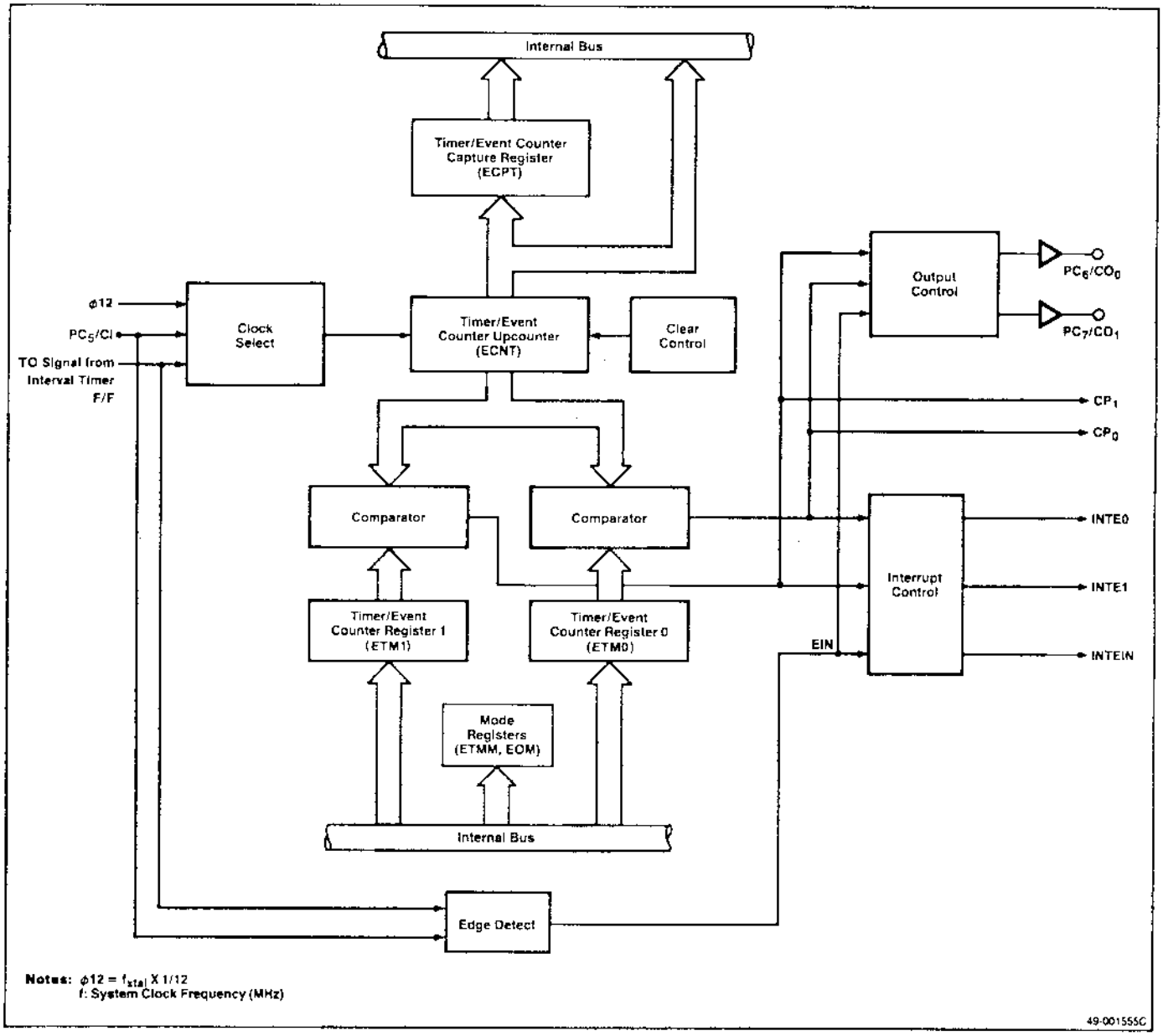
The 16-bit timer/event counter can provide the following functions:

- Interval timing
- External event counting
- Frequency measurement
- Pulse-width measurement
- Programmable frequency and duty cycle waveforms
- Single-pulse output

Figure 6-1 shows that the main functional blocks of the timer/event counter include the following:

- Upcounter (ECNT)
- Capture register (ECPT)
- Two counter registers (ETM0 and ETM1)
- Two comparators
- Input clock select circuit
- Clear control circuit
- Interrupt control circuit
- Output control circuit
- Two mode registers (ETMM and EOM)

Figure 6-1. Block Diagram of Multifunction Timer/Event Counter



Upcounter [ECNT]

This is a 16-bit counter that is incremented by one each time it receives a pulse from the input clock select circuit and is cleared by the clear control circuit. When the upcounter counts up to FFFFH and overflows to 0000H, the OV (overflow) flag is set. The OV flag can be tested by the SKIT and SKNIT skip instructions. The OV flag will not cause an interrupt.

Capture Register [ECPT]

This 16-bit buffer register holds the captured contents of the upcounter. The contents of the upcounter can be latched into the capture register at the falling edge of the CI input (PC_5) if the ϕ12 clock is the ECNT clock source or by the falling edge of the TO input when CI is the input clock source.

Counter Registers [ETM0 and ETM1]

These 16-bit, user-programmable registers contain the number of clock inputs to be counted.

Comparators

These circuits compare the contents of ETM0 or ETM1 with the contents of the upcounter and issue a match signal (CP0 or CP1) when the contents are identical.

Input Clock Select Circuit

This circuit selects the input clock to the upcounter as specified by ETMM. When CI or TO is selected, it is sampled by the ϕ3 clock (250 ns at 12 MHz) to provide noise immunity. The input (either high or low level) must be present long enough to be sampled by two clock edges. Therefore, the input must be present for a minimum time of 500 ns.

Clear Control Circuit

This circuit clears the upcounter to 0000H as programmed by ETMM. When cleared, the upcounter is either stopped or starts counting up from 0000H.

Interrupt Control Circuit

This circuit generates interrupt requests by setting the interrupt request flag (INTFE0 or INTFE1) when one of the following conditions occurs.

- Contents of ECNT and ETM0 are equal.
- Contents of ECNT and ETM1 are equal.

An external interrupt request flag (INTFEIN) is set when one of the following conditions occurs. See table 6-1.

- Fall of CI.
- Fall of TO.

Table 6-1. Setting the Interrupt Request Flag

ETMM			Interrupt Request
ET_1	ET_0	ECNT Input	Flag Set
0	0	Internal clock (ϕ12)	Fall of CI input
0	1	ϕ12 while CI input is high level	
1	0	CI input	Fall of TO
1	1	CI input while TO is high level	

Output Control Circuit

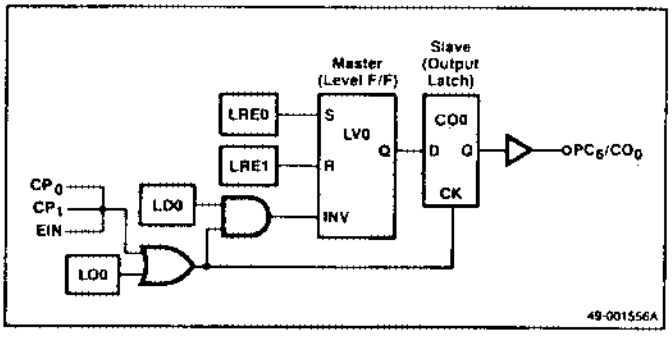
This circuit is used to output various frequencies and pulse widths to the CO_0 and CO_1 outputs. The outputs vary depending upon the programming of mode registers ETMM and EOM.

Figure 6-2 is a diagram of the CO_0 output circuitry at pin PC_6. There is identical circuitry for the CO_1 output connected to pin PC_7

The level flip-flop (LV0) is the master portion of the CO_0 output and it contains the next level that will go to the output latch (the slave portion). The CO_0 output can be controlled using the ETMM register in combination with the EOM register. The output can also be controlled by EOM only.

The CP_0, CP_1, and EIN inputs are generated by the timer circuitry. CP_0 is generated (figure 6-1) when ECNT = ETM0, CP_1 is generated when ECNT = ETM1, and EIN is generated if a falling edge of CI or TO is detected. The remaining inputs—LD_0, LO_0, LRE_0, and LRE_1—are from the EOM register.

Figure 6-2. Block Diagram of Output Control Circuit (Showing CO_0 Output)



CP_0, CP_1, and LO_0 will all clock the slave portion (output latch) of CO_0. When clocked, the output latch, and hence CO_0, will be loaded with the current level in the LV0 flip-flop. Bits LRE_0, LRE_1, and LD_0, respectively, control the set, reset, and invert inputs of LV0. Both LV0 and LV1 are reset to 0 after a $\overline{\text{RESET}}$ input.

Mode Registers

There are two registers (in addition to the counter registers ETM0 and ETM1) that, when programmed, control the operation of the timer/event counter. These registers are the timer/event counter mode register (ETMM) and the timer/event counter output mode register (EOM). Figures 6-3 and 6-4 show the format of the ETMM and EOM registers, respectively.

The ETMM register is used to specify the ECNT clock source, when ECNT will be cleared, and when outputs CO_0 and CO_1 will change. The EOM register is mainly used to determine how the LV0 flip-flop will change. Bits LO_0 and LO_1 of the EOM register can be used to control when CO_0 and CO_1 will change.

ETMM Register

The contents of ETMM are reset to 00H when the $\overline{\text{RESET}}$ input is valid. Bits ET_0 and ET_1 specify the input clock source to the upcounter. Bits EM_0 and EM_1 specify how and when the ECNT is cleared. Table 6-2 shows in detail the clearing of ECNT when $EM_0 = 0$ and $EM_1 = 1$. Bits CO_{00} and CO_{01} specify the conditions when ECNT will cause a CO_0 output change and bits CO_{10} and CO_{11} specify the conditions when ECNT will cause a CO_1 output change.

Table 6-2. Clearing the Upcounter Based on CI and TO Signals

ETMM			
ET_1	ET_0	ECNT Input	ECNT Clear
0	0	Internal clock (ϕ12)	Fall of CI input
0	1	ϕ12 while CI input is high level	
1	0	CI input	Fall of TO
1	1	CI input while TO is high level	

Figure 6-3. Timer/Event Counter Mode Register (ETMM)

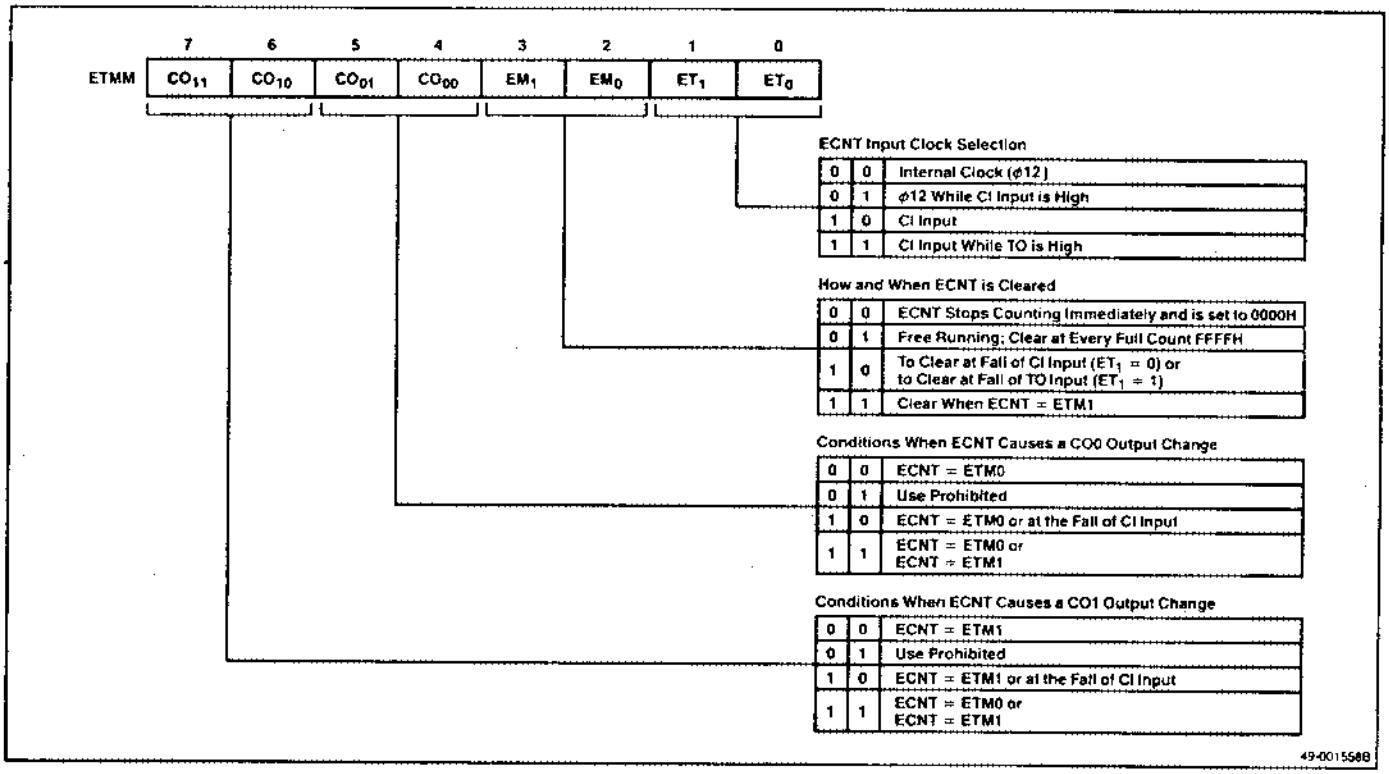
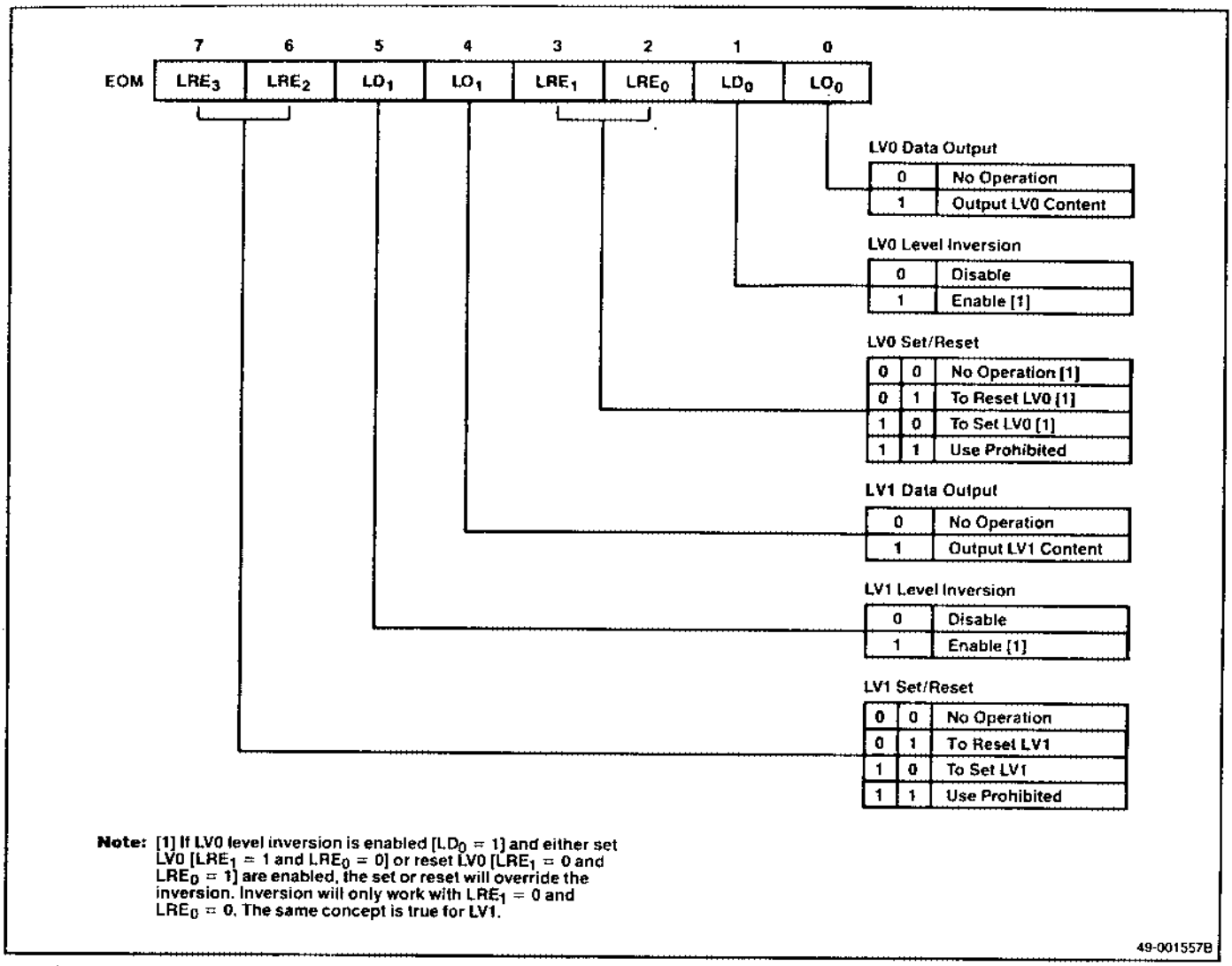


Figure 6-4. Timer/Event Counter Output Mode Register



EOM Register

This 8-bit register is used to control the LV0 and LV1 flip-flops and may also be used to control the CO_0 and CO_1 outputs. All 8 bits may be written to, but only bits 1 and 5 may be read. The remaining bits will read 0. Figure 6-4 shows the format of the EOM register.

Bits 0-3 of the EOM register control CO_0 and bits 4-7 control CO_1. Bits 1-3 specify whether LV0 is to be set, reset, or inverted. Bit 0 of EOM controls when the LV0 output latch will change. For example, if EOM is programmed:

```
MVI    EOM,09H
```

LV0 will be set since LRE_1 and LRE_0 = 10. LV0 will be output to CO_0 immediately (high level) since bit LO_0 = 1. CO0 can be set by just one EOM = 09H instruction. However, if the counter were programmed:

```
SET F/F:   MVI    EOM,08H
OUTPUT:    MVI    EOM,01H
```

LV0 will be set when the SET F/F instruction is executed, but the output CO_0 will not become a high until the OUTPUT instruction is executed. After the instruction is executed, bit LO_0 will become a 0. Therefore, when using EOM only to control CO_0, EOM must be programmed with bit 0 = 1 each time the CO_0 output is to be changed. Usually, bit LO_0 is used to set an initial state at CO_0 and ETMM bits CO_{01} and CO_{00} (bits 5 and 4) control when CO_0 will change for the remainder of the time.

Bits 1-3 of EOM determine how output CO_0 will change. Bits 4-7 of EOM and bits 6 and 7 of ETMM perform the corresponding functions on LV1 and output CO_1.

The inversion bit LD_0 works differently from the set or reset of LV0 using LRE_0 and LRE_1 in that it does not invert CO_0 the first time the invert bit is set. Consider the following example:

```
INV1:  MVI   EOM,03H   ;Output LV0, then
                       ;invert LV0.
INV2:  MVI   EOM,03H   ;Output LV0 and invert
                       ;LV0.
```

When INV1 is executed, LV0 will be output and then inverted. Therefore, output CO_0 does not change. When INV2 is executed, LV0 will be output to CO_0 and then inverted. Therefore, CO_0 will not be inverted the first time it gets EOM = 03H. After this, LV0 will invert each time EOM is loaded with 03H.

OPERATION

The timer/event counter begins operation when the timer/event counter registers are loaded with the appropriate count values and EOM and ETMM are loaded with the appropriate operating data in the sequence outlined in figure 6-5.

Operation as an Interval Timer

As an interval timer, the circuit generates interrupts at programmed time intervals. These intervals may range from 1 μs to 65.535 ms with a resolution of 1 μs (at 12 MHz). To operate the timer/event counter as an interval timer, the upcounter (ECNT) must be cleared; then the desired count value must be loaded into the counter registers (ETM0, ETM1). Bits 0-1 of the mode register (ETMM) must then be cleared and bits 2-3 set, as shown in figure 6-6. The timer/event counter now functions as an interval timer using the internal clock (ϕ12) as the clock source to ECNT.

Figure 6-5. Setup Sequence for Timer/Event Counter

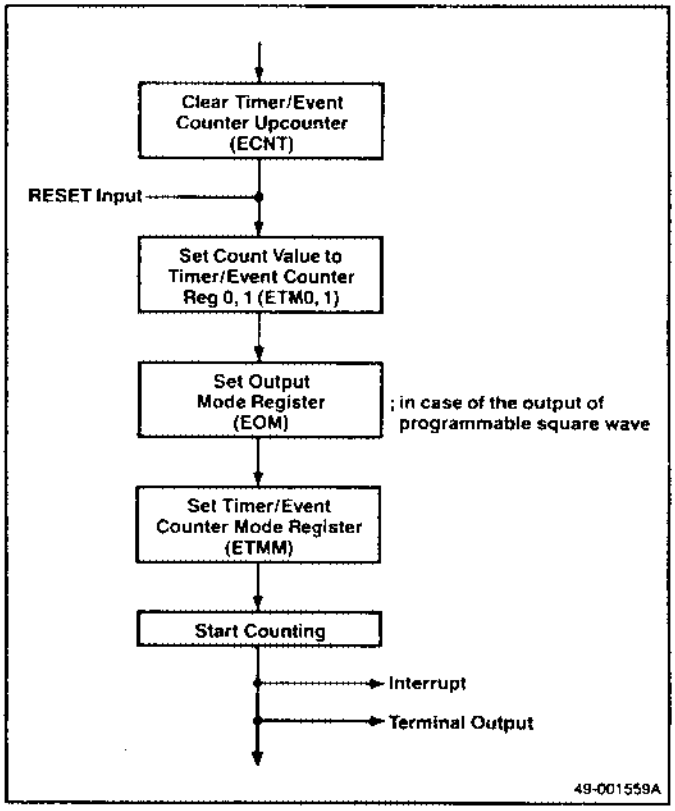
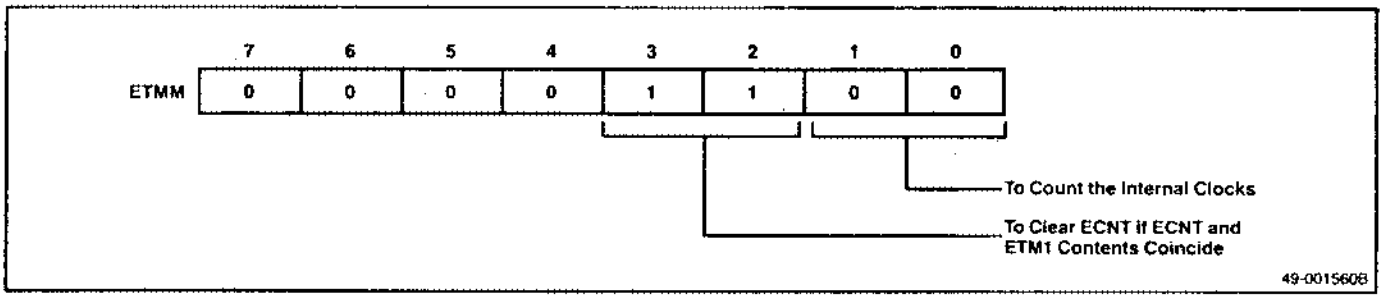
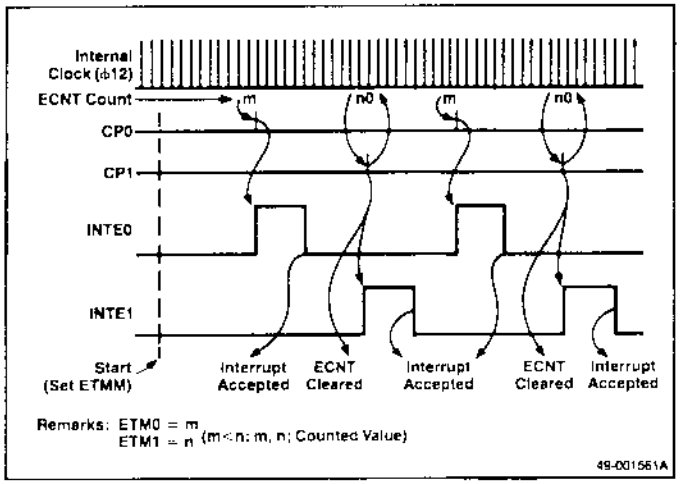


Figure 6-6. Setting Up ETMM for Interval Timer Operation



The upcounter (ECNT) is incremented every 12 cycles of the external oscillator by using the ϕ12 clock (1 μs when operating at 12 MHz). The comparator continuously compares the contents of the upcounter with ETM0 and ETM1. When the upcounter equals either ETM0 or ETM1, the timer/event counter generates an internal interrupt (INTE0 or INTE1). When the contents of the upcounter and ETM1 are equal, the upcounter is cleared and counting resumes from 0000H. In this way, the timer/event counter marks intervals based on the count value contained in ETM1. This operational timing sequence is diagramed in figure 6-7. The INTE0 and INTE1 interrupts can be masked by setting interrupt mask register bits MKE0 and MKE1.

Figure 6-7. Timing Sequence of Interval Timer Operation of Timer/Event Counter

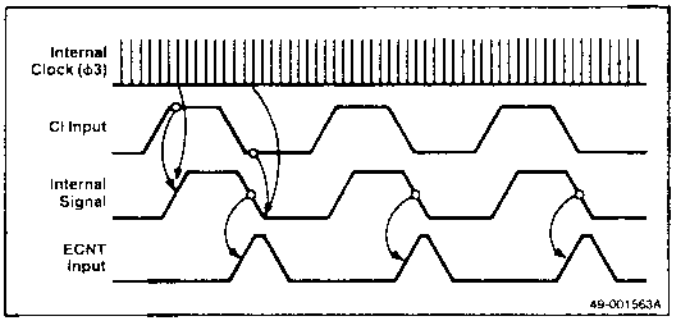


Operation as an Event Counter

The number of pulses input from the CI (PC_5) input can be counted by clearing the upcounter, then loading the appropriate operating data in the lower-order 4 bits of ETMM as shown in figure 6-8. The external pulses from the CI input are synchronized by the internal clock, and the upcounter is incremented at the falling edge of each pulse. The count value can be read at any time by the program.

When the lower-order 4 bits of the mode register are loaded with the values shown in figure 6-8, the overflow flag (OV) is set when the upcounter increments from FFFFH to 0000H. The overflow flag can be tested by skip instructions SKIT and SKNIT. When the value of the upcounter equals the count value contained in either ETM0 or ETM1, an interrupt (INTE0 or INTE1) is generated. Figure 6-9 shows the timing sequence for the event counter operation.

Figure 6-9. Timing Sequence for Event Counter Operation of Timer/Event Counter



Frequency Measurement

The number of CI input pulses per time interval can be counted by setting the timer/event counter to count while the interval timer output (TO) is high. The upcounter must first be cleared by programming ETMM with 00H and then loading the appropriate operating data in the lower-order bits of ETMM as shown in figure 6-10.

The upcounter is incremented by each clock input on the CI input during the time TO is held high. At the falling edge of TO, an internal interrupt (INTEIN) is generated, the contents of the upcounter are transferred to the timer/event counter capture register (ECPT), and the upcounter is cleared. Since the CI pulses are input to the upcounter only when TO is high, the upcounter counts the number of CI input pulses per time interval measured by TO and puts the result in the capture register. Dividing the value in ECPT by the time interval generated on TO gives the frequency of the CI input. The timing sequence for this operation is shown in figure 6-11.

Figure 6-8. Setting Up ETMM for Event Counter Operation

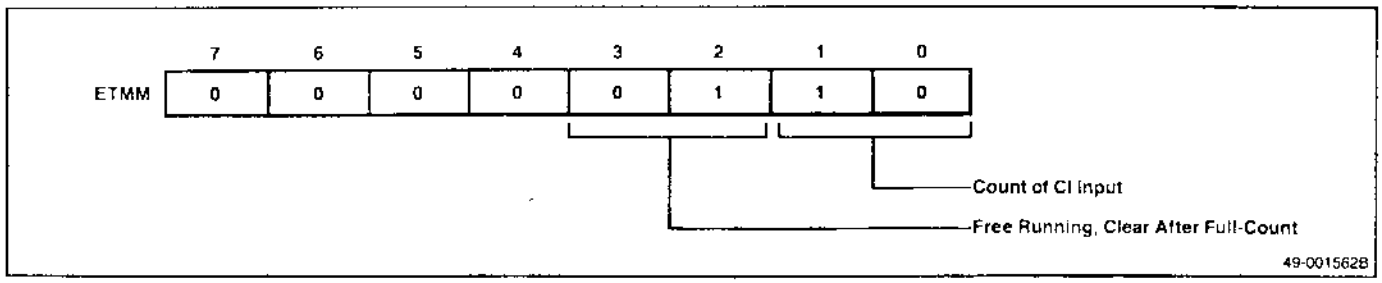


Figure 6-10. Setting Up ETMM for Frequency Measurement Operation

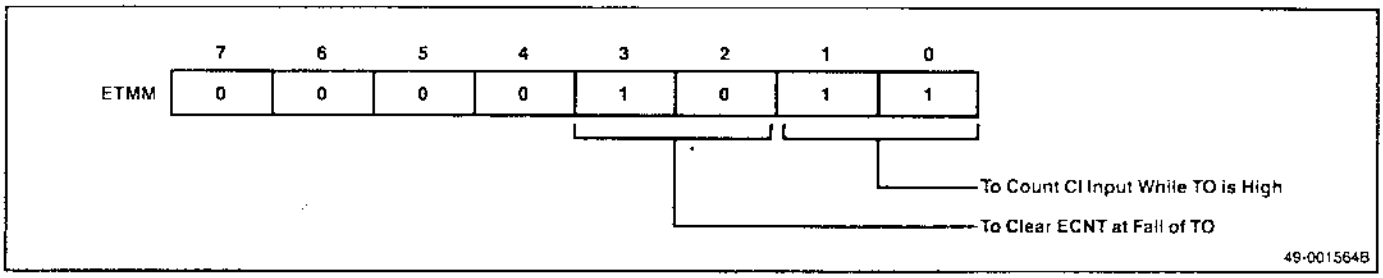
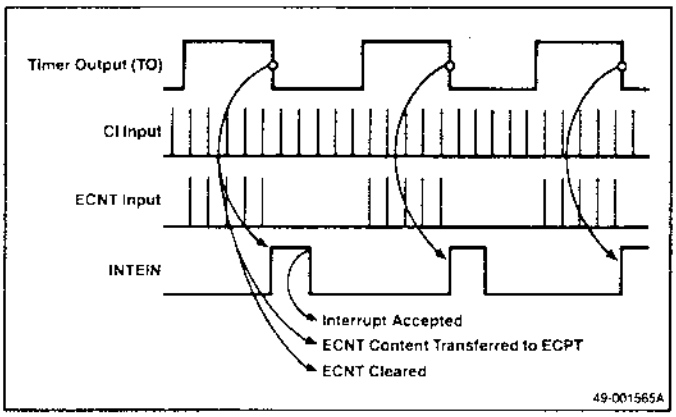


Figure 6-11. Timing Sequence for Frequency Measurement Operation of Timer/Event Counter



Pulse-Width Measurement

The width of the CI input pulses can be measured by counting the number of clock pulses that occur in the time the CI line is held high. The upcounter must be cleared, and then ETMM must be programmed as shown in figure 6-12.

When the CI input goes high, the internal clock (ϕ12) becomes the clock source for the upcounter and the upcounter is incremented at each clock pulse. When the CI input goes low, internal interrupt INTEIN is generated, the contents of the upcounter are transferred to the capture register (ECPT), and the upcounter is then cleared.

This operation requires that the CI input has a high or low pulse width of no less than 16 timing states (4 μs at 12 MHz). If this requirement is not met, this function cannot be performed. The pulse width can be calculated by multiplying the value in ECPT by 1 μs (the period of the ϕ12 clock). The timing sequence for this operation is shown in figure 6-13.

Figure 6-13. Timing Sequence for Pulse-Width Measurement Operation of Timer/Event Counter

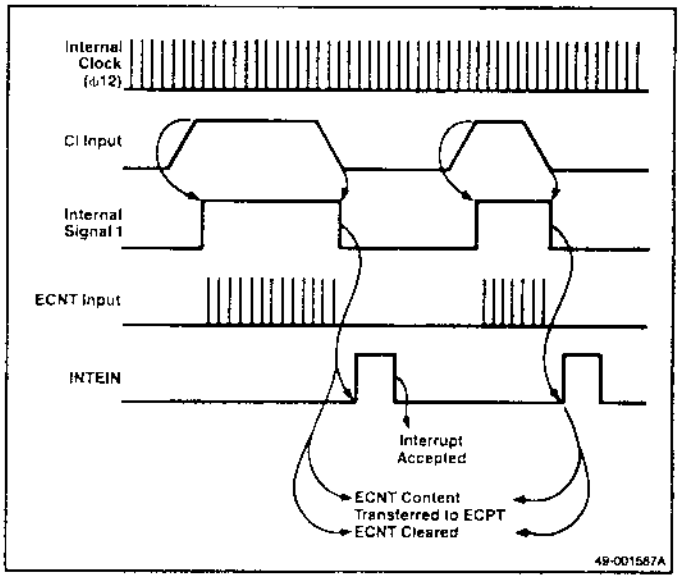
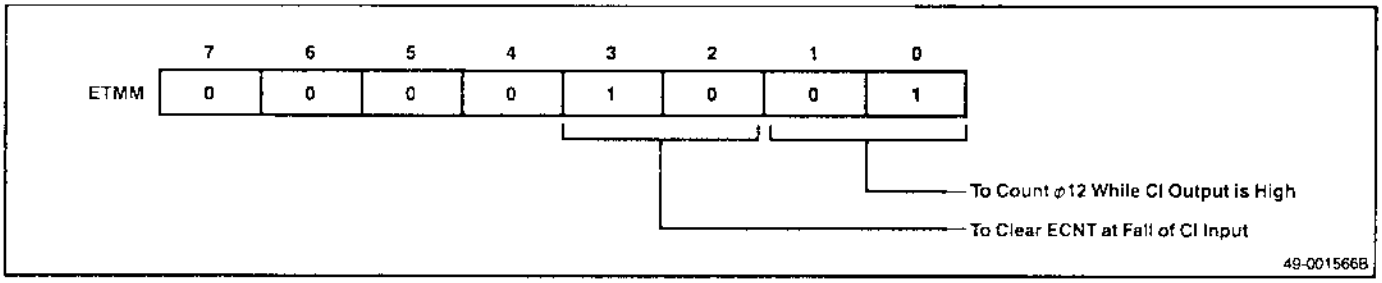


Figure 6-12. Setting Up ETMM for Pulse-Width Measurement Operation



Programmable Waveform Output

The timer/event counter can output a programmable frequency and a programmable duty-cycle waveform at two outputs. These outputs are designated CO_0 and CO_1, and operate identically. For a programmable waveform output, the upcounter must be cleared and the desired count value loaded into ETM0 and ETM1. The appropriate operational data should then be loaded into the output mode register (EOM) and the counter mode register (ETMM), as shown in figures 6-14 and 6-15, respectively. Setting EOM to 05H resets CO0 to a low and setting EOM to 02H enables the inversion of LV0.

The upcounter is incremented by each pulse of the internal clock (ϕ12). The comparator compares the contents of the upcounter to both ETM0 and ETM1. If the upcounter is equal to either, then a match signal (CP0 or CP1) and internal interrupt (INTE0 or INTE1) are generated. Each time a match signal is generated, LV0 is output to CO_0 and then LV0 is inverted. If the contents of the upcounter and ETM1 are equal, the upcounter is cleared and counting resumes from 0000H. Interrupts INTE0 and INTE1 can be masked by setting bits MKE0 and MKE1 in the interrupt mask register. Figure 6-16 shows the timing sequence for this operation.

Figure 6-14. Example of Setting Up EOM for Programmable Waveform Output on CO_0

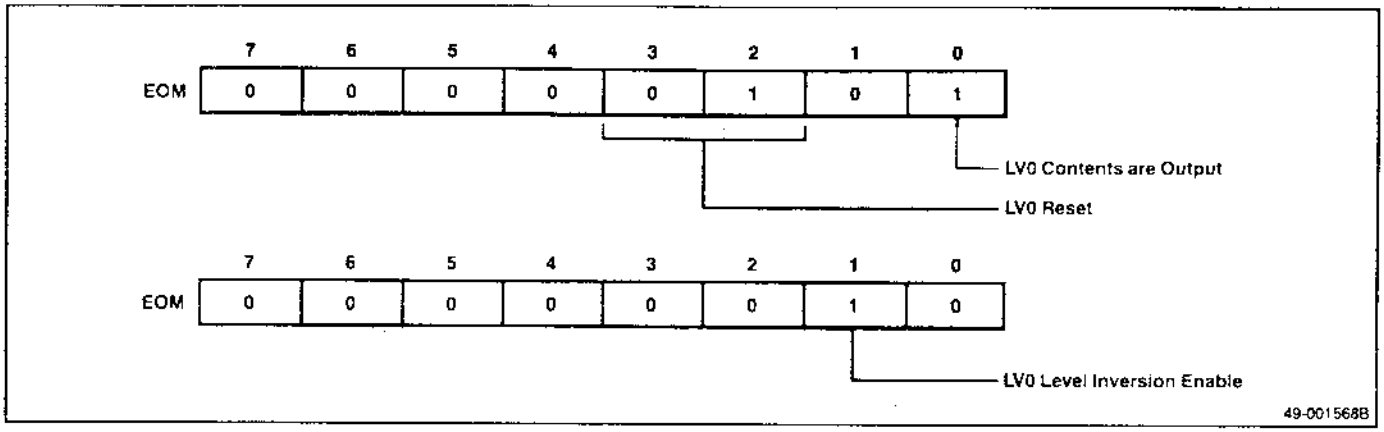


Figure 6-15. Example of Setting Up ETMM for Programmable Waveform Output on CO_0

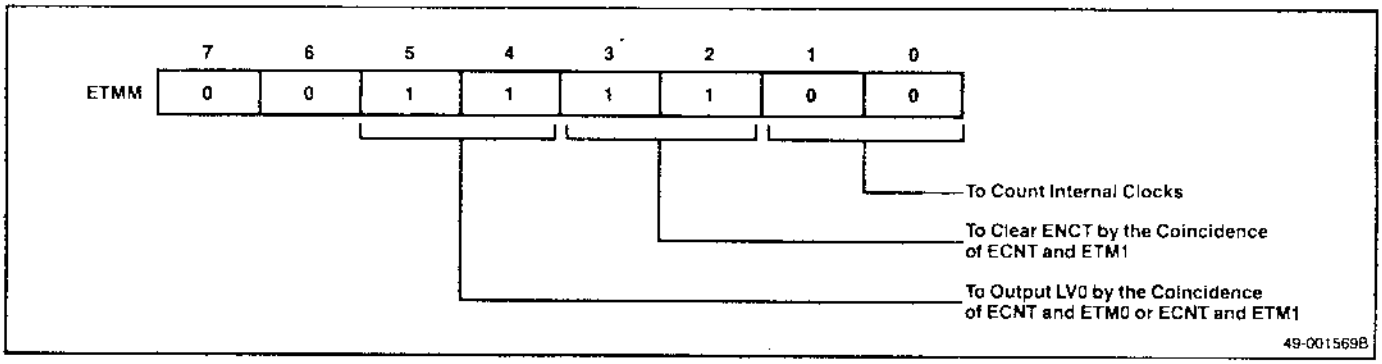
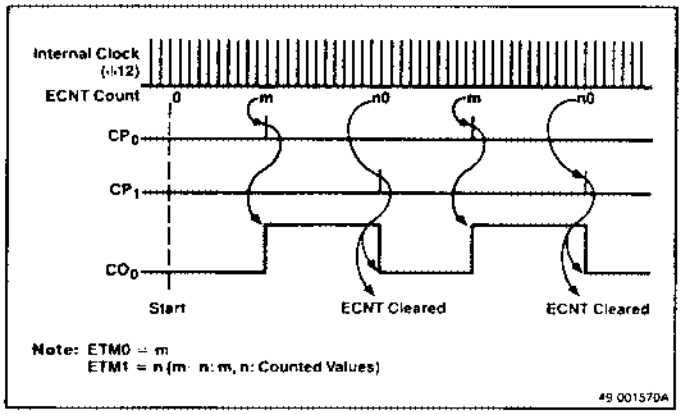


Figure 6-16. Timing Sequences for Programmable Waveform Output Example



PROGRAMMING EXAMPLES

Programmable Waveform Output

In this example, the programmable waveform is output from CO_0; the low-level value is 200 μs and the high-level value is 300 μs. The sequence of events is shown in figure 6-17 and described below in steps (a) through (d). Table 6-3 gives examples of program codes for each step.

Figure 6-17. Flowchart of Step Sequences in Programmable Waveform Output Example

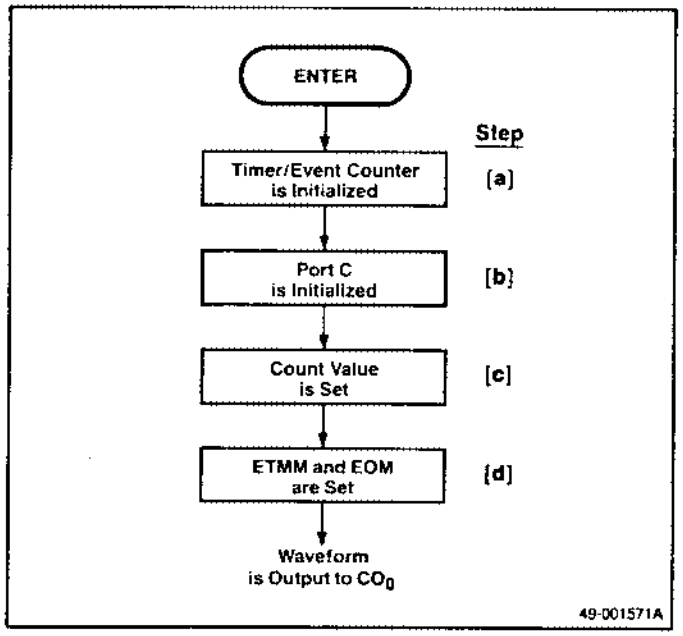


Table 6-3. Example of Program Code for Programmable Waveform Output

```
;*****TIMER/EVENT COUNTER INITIALIZATION*********
INIT:   MVI   A,00H                                    Step (a)
        MOV   ETMM,A    ;Clear timer/event counter.
        MVI   EOM,05H   ;Initialize CO0 output.
        MVI   EOM,02H   ;Enable LV0 inversion.
        MVI   A,40H                                    Step (b)
        MOV   MCC,A     ;Set PC6 to mode control.
        LXI   EA,00C8H  ;Low level: 200 μs at 12 MHz.  Step (c)
        DMOV  ETM0,EA   ;Count value to ETM0.
        LXI   EA,01F4H  ;High level: 300 μs at 12 MHz.
        DMOV  ETM1,EA   ;Count value to ETM1.
START:  MVI   A,3CH                                    Step (d)
        MOV   ETMM,A    ;Set timer/event counter mode
                        ; and start.
```

(a) The upcounter is cleared. The LV0 flip-flop is cleared and sent to the output latch, driving the CO_0 output low and enabling LV0 inversion. Figure 6-18 shows how these events are set up in ETMM and EOM.

(b) In the mode control C register (MCC), the PC_6 line is specified for CO_0 output as shown in figure 6-19.

(c) Register ETM0 is set to 00C8H (low level = 200 μs at 12 MHz) and register ETM1 is set to 01F4H (interval = 500 μs at 12 MHz). ETM0 defines the low-level pulse width of the signal, and ETM1 defines the period of the waveform output at CO_0.

(d) Various operating conditions are specified in the timer/event counter mode register (ETMM) as shown in figure 6-20. The internal clock (ϕ12) is specified as the clock source to the upcounter by bits 0 and 1. The condition to clear the upcounter when the contents of the upcounter and ETM1 are equal is specified by bits 2 and 3. The condition to output the CO_0 signal when the contents of the upcounter and either ETM0 or ETM1 are equal is specified by bits 4 and 5 (figure 6-20).

Figure 6-18. Setting Up ETMM and EOM for Programmable Waveform Output Example

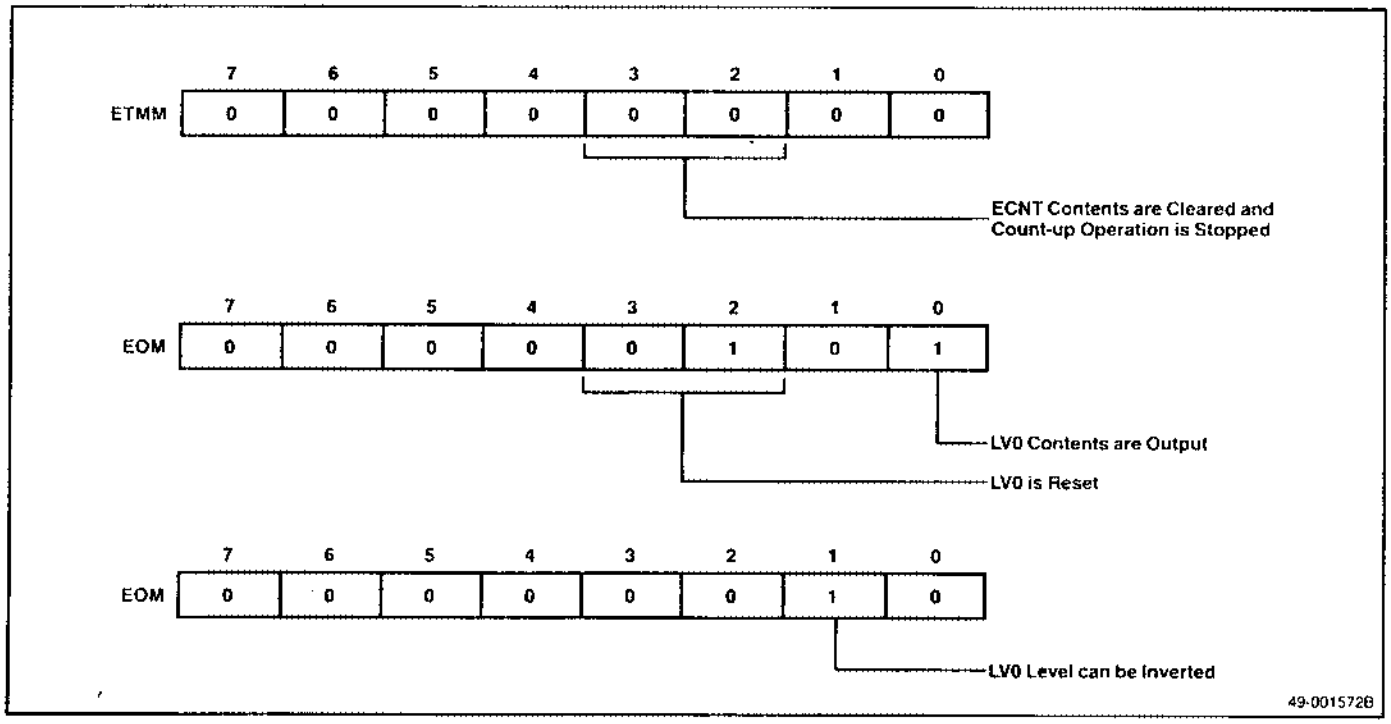


Figure 6-19. Specifying PC_6 for CO_0 Output in Programmable Waveform Output Example

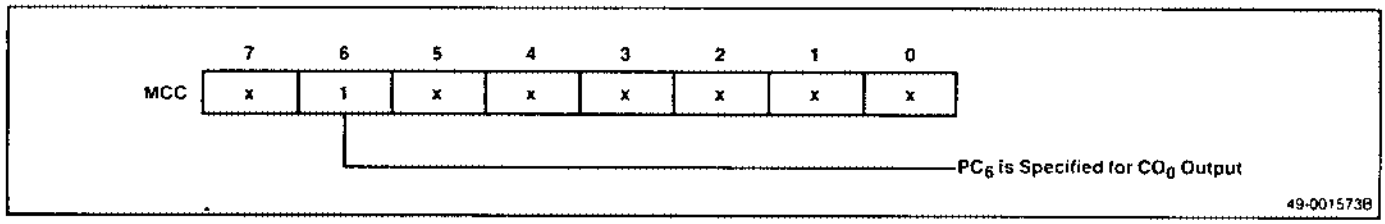
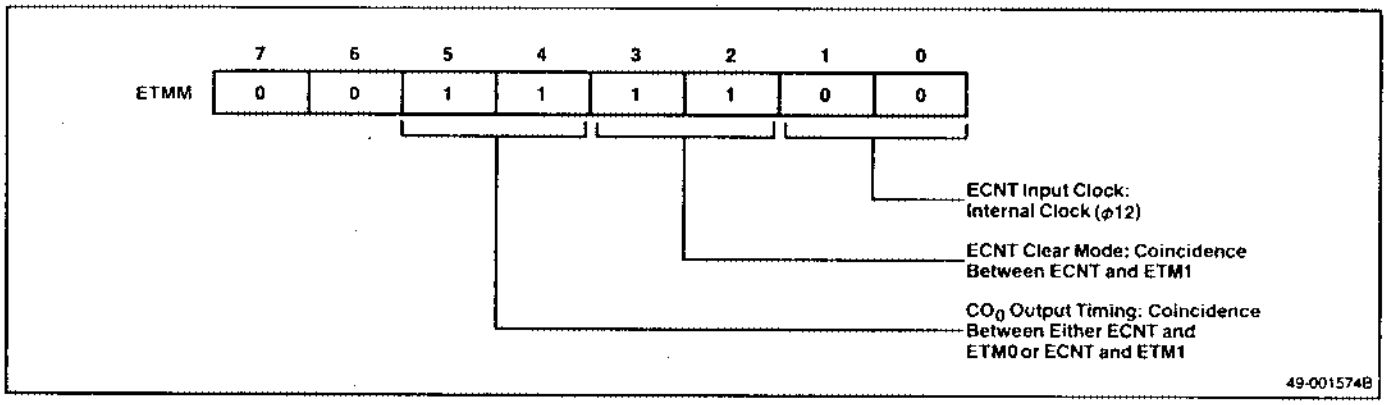


Figure 6-20. Setup of ETMM for Programmable Waveform Output Example

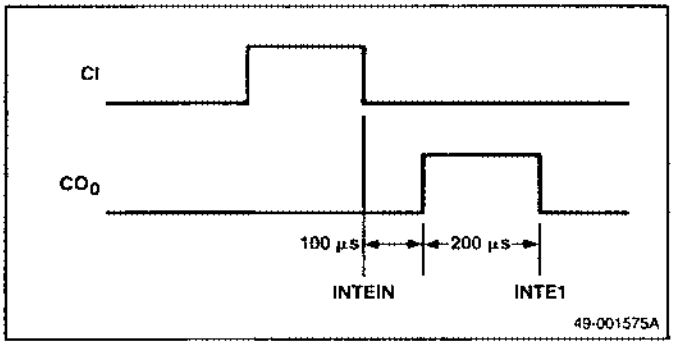


Single-Pulse Output

In this example, a single pulse is output from CO_0 after a specified time has elapsed from the fall of the CI input. In the figure 6-21 timing diagram, a 200-μs pulse is output 100 μs following the fall of CI.

The phases in this operation include initialization, processing of the INTEIN interrupt generated at the falling edge of CI input, and processing of the INTE1 interrupt generated when the contents of the upcounter and ETM1 are equal.

Figure 6-21. Signal Timing for Single-Pulse Output Example



Initialization Phase. Initialization is flowcharted in figure 6-22 and described in steps (a) through (d) below.

(a) The upcounter is cleared (figure 6-18). The contents of the LV0 and the CO_0 output are driven low by resetting the LV0 level flip-flop (figure 6-19).

(b) The PC_5 line is specified for CI input and the PC_6 line is specified for CO_0 output, as shown in figure 6-23.

(c) The INTEIN interrupt is enabled and the INTAD interrupt (which has the same vector location) is masked, as shown in figure 6-24.

(d) Basic operating conditions of the timer/event counter are established by the counter mode register (ETMM), as shown in figure 6-25. Bits 0 and 1 specify the internal clock (ϕ12) as the clock source to the upcounter. Bits 2 and 3 specify that the upcounter is free-running (it is cleared at full count and continues counting from 0000H).

Figure 6-22. Flowchart of Initialization Phase of Single-Pulse Output Example

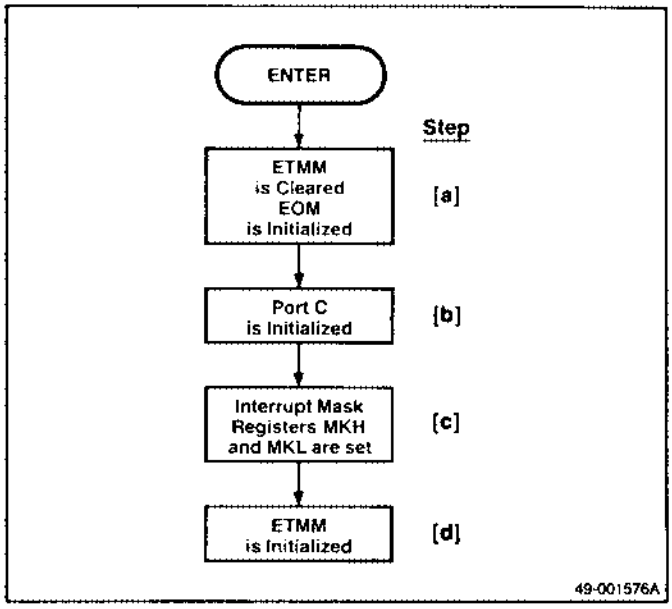


Table 6-4 gives examples of program codes for each step in the initialization phase.

Table 6-4. Example of Program Code for Initialization Phase of Single-Pulse Output Example

```
;*****TIMER/EVENT COUNTER INITIALIZATION*********
INIT:   MVI   A,00H                                    Step (a)
        MOV   ETMM,A      ;Clear timer/event counter.
        MVI   EOM,05H     ;Initialize CO0 = 0.
        MVI   A,60H       ;PC5 = CI, PC6 = CO0.        Step (b)
        MOV   MCC,A       ;Set port mode control.
        ANI   MKL,7FH     ;INTEIN enable.              Step (c)
        ORI   MKH,0FH     ;Mask INTAD.                 Step (d)
START:  MVI   A,04H       ;ECNT free running.
        MOV   ETMM,A      ;Set timer/event counter
                          ;mode and start.
```

Figure 6-23. Specifying PC_5 and PC_6 for Single-Pulse Output Example

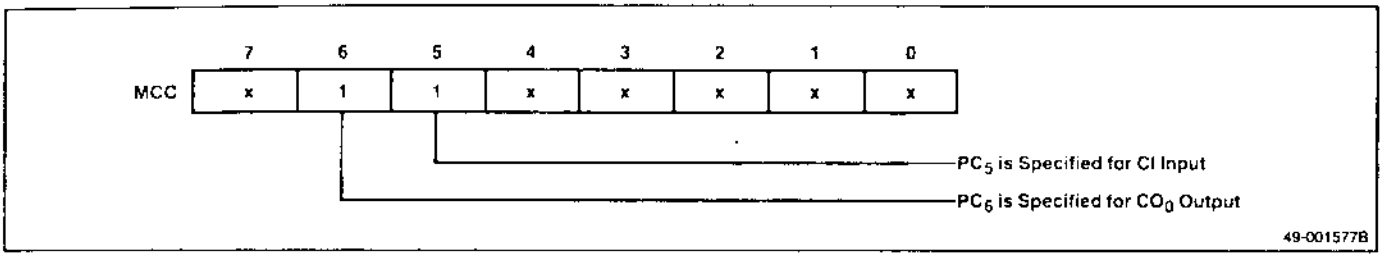


Figure 6-24. Setup of Interrupt Mask Register for Initialization Phase of Single-Pulse Output Example

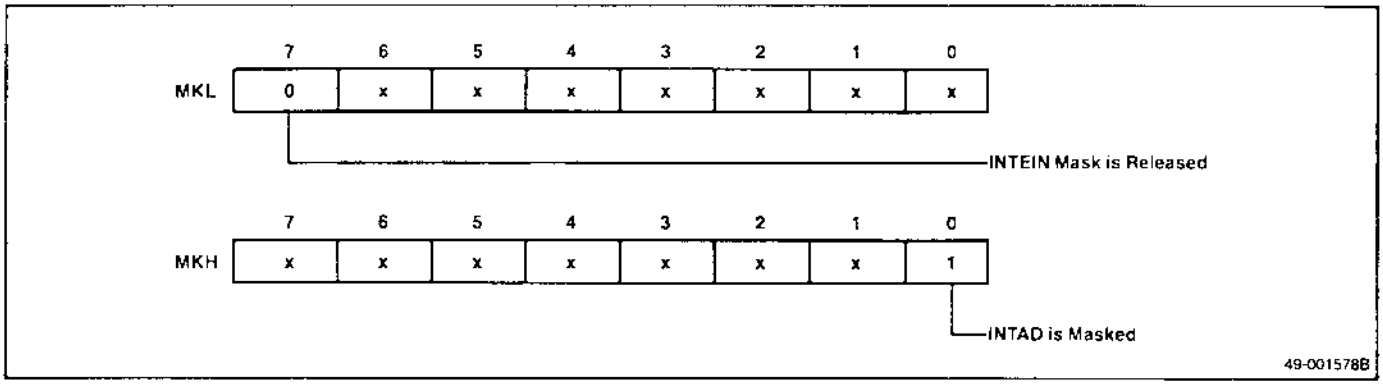
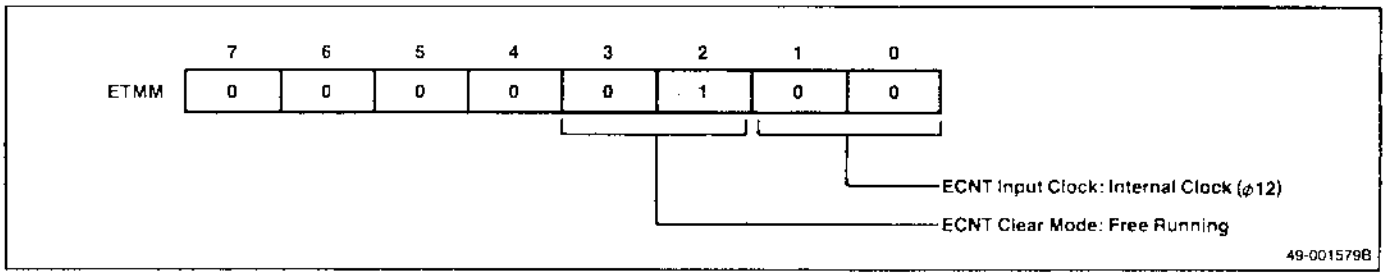


Figure 6-25. Setup of ETMM for Initialization Phase of Single-Pulse Output Example



INTEIN Interrupt Phase. After initialization, when the CI input goes low, the value in the upcounter is latched into the timer/event counter capture register (ECPT), and internal interrupt INTEIN is generated. The INTEIN interrupt phase, flowcharted in figure 6-26, is described below in steps (a), (b), and (c).

(a) ETM0 is loaded with the value in the capture register plus 0064H (100 μs) to establish the time interval of 100 μs after the falling edge of CI input. ETM1 is loaded with the value in the capture register plus 012CH (i.e., 300 μs after the falling edge of CI) to establish a pulse width of 200 μs (at 12 MHz).

(b) The timer/event counter mode register (EOM) will be used to control CO_0 when the contents of the upcounter and either ETM0 or ETM1 are equal. The LV0 level inversion bit (LD_0) is set. Figure 6-27 shows the operating parameters set up in ETMM and EOM.

(c) The INTE1 interrupt is enabled and the INTE0 interrupt is masked by the interrupt mask register, as shown in figure 6-28.

Figure 6-26. Flowchart of INTEIN Interrupt Service Routine in Single-Pulse Output Example

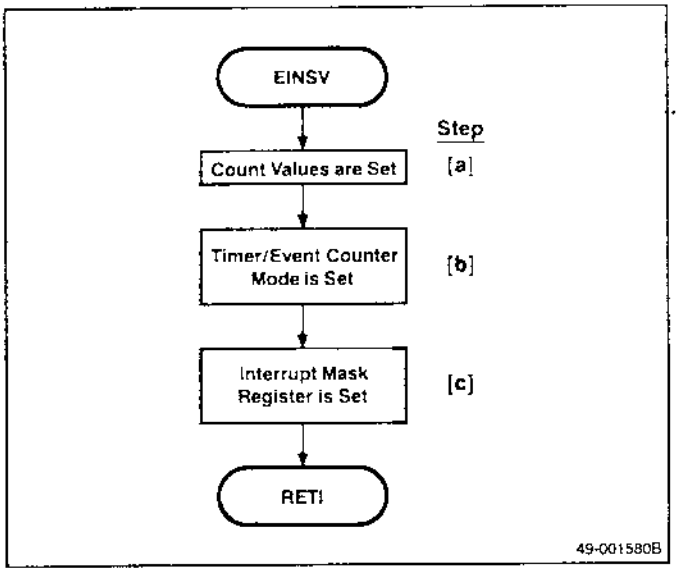


Figure 6-27. Setup of ETMM and EOM for INTEIN Interrupt Phase of Single-Pulse Output Example

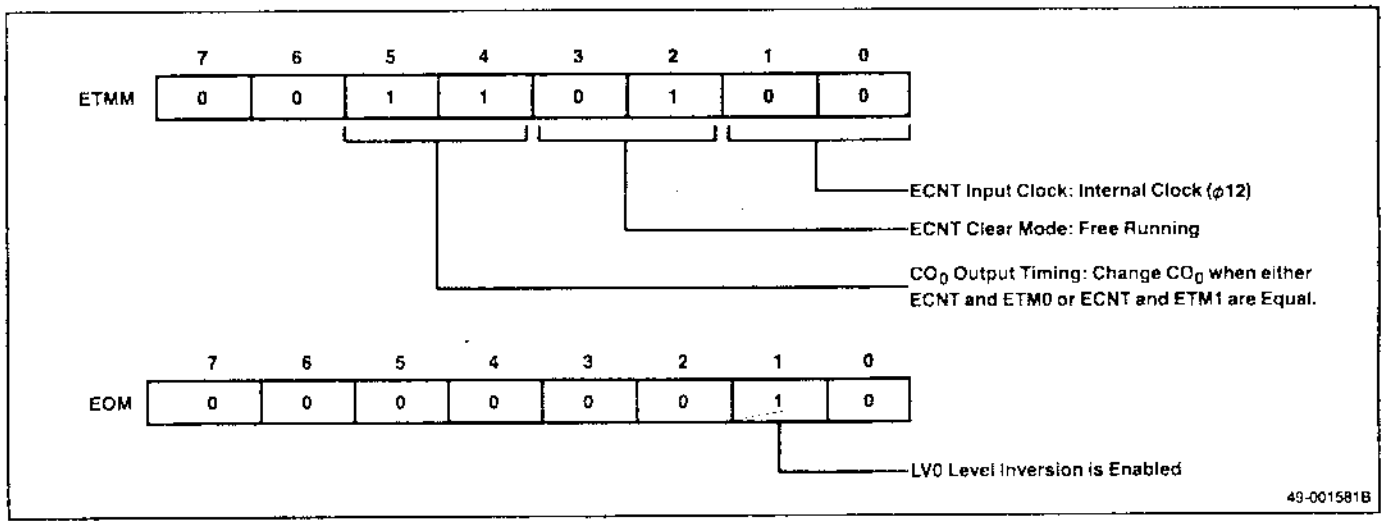


Figure 6-28. Setup of Interrupt Mask Register for INTEIN Interrupt Phase of Single-Pulse Output Example

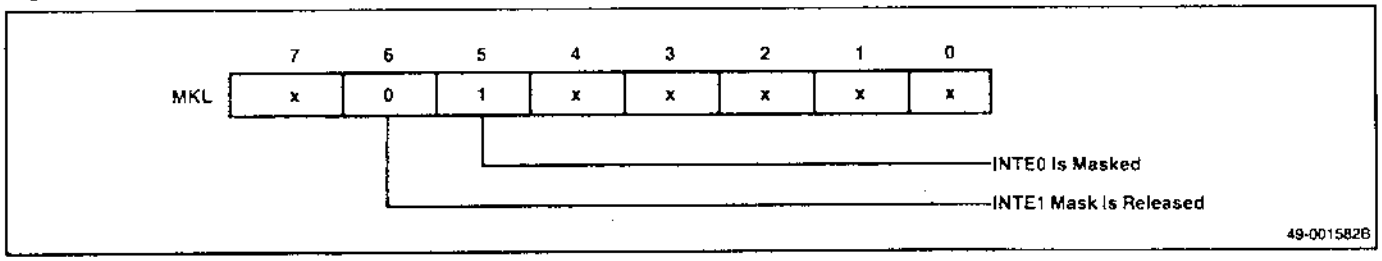


Table 6-5 gives examples of program codes for each step in the INTEIN interrupt phase. The JMP EINSV instruction should be stored in the INTEIN interrupt start address 0020H.

Table 6-5. Example of Program Code for INTEIN Interrupt Phase of Single-Pulse Output Example

```
;*****TIMER/EVENT COUNTER INTERRUPT WHEN CI GOES LOW*********
EINSV:  EXA                     ;Save accumulator.
        EXX                     ;Save registers.
        DMOV    EA,ECPT         ;Get ECPT.                      Step (a)
        LXI     B,0064H
        DADD    EA,B            ;Low level: 100 μs at 12 MHz
        DMOV    ETM0,EA         ;Set count value in ETM0.
        LXI     B,00C8H         ;Load BC with 200.
        DADD    EA,B            ;High level: 200 μs at 12 MHz.
        DMOV    ETM1,EA         ;Set count value in ETM1.
        MVI     A,34H                                           Step (b)
        MOV     ETMM,A          ;Set ETMM
        MVI     EOM,02H         ;Set LV0 inversion enable.
        ANI     MKL,08FH        ;INTE1 unmasked and             Step (c)
                                ;INTE0 masked.
        EXX                     ;Recover register.
        EXA                     ;Recover accumulator.
        EI
        RETI
```

INTE1 Interrupt Phase. After the program in table 6-5 is executed, ECNT starts counting up. When ECNT = ETM0, CO_0 is set high and ECNT continues counting up. When ECNT = ETM1, interrupt INTE1 is generated. The INTE1 interrupt phase is described in steps (a) and (b) below. Figure 6-29 is the flowchart.

(a) The timer/event counter output mode register (EOM) designates that CO_0 output operations be stopped.

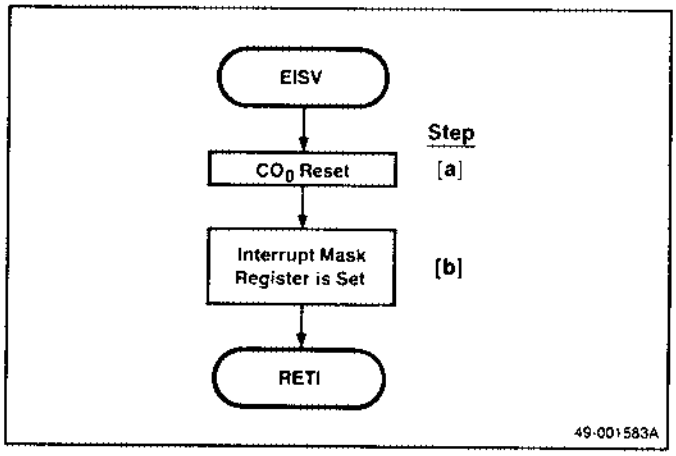
(b) Internal interrupt INTE1 is masked by the setup in the interrupt mask register.

Table 6-6 gives examples of program codes for each step in the INTE1 interrupt phase.

Table 6-6. Example of Program Code for INTE1 Interrupt Phase in Single-Pulse Output Operation

```
;*****TIMER/EVENT COUNTER INTERRUPT WHEN ECNT = ETM1*********
EISV:   MVI     EOM,00H                                 Step (a)
        ORI     MKL,40H         ;INTE1 disable.         Step (b)
        EI
        RETI
```

Figure 6-29. Flowchart of INTE1 Interrupt Service Routine in Single-Pulse Output Example



ORGANIZATION

The microcomputer contains a serial interface that permits communications in either asynchronous, synchronous, or I/O interface mode.

The serial interface consists of three lines: the RxD serial data input line, the TxD serial data output line, and the $\overline{SCK}$ serial I/O clock line. Operation is controlled by the serial mode register. The receiver and transmitter each contain a control section, a conversion register (parallel-to-serial or serial-to-parallel), and a buffer register. This allows full-duplex operation in the asynchronous mode. Since the same clock is used for transmit and receive, half-duplex operation is available in synchronous and I/O mode. A block diagram of the serial interface is shown in figure 7-1.

Transmitter

Parallel-to-Serial Conversion Register. This register converts the parallel data loaded from the transmit buffer register into serial data that is sent out on the TxD line.

Transmit Buffer. Parallel data intended for transmission is temporarily stored in this register. The new contents of this register are transferred to the parallel-to-serial conversion register after data in the conversion register has been transmitted. When the transmit buffer register is empty, the INTST interrupt request is generated.

Transmit Control Circuit. This circuit controls the entire transmitter and generates all required signals.

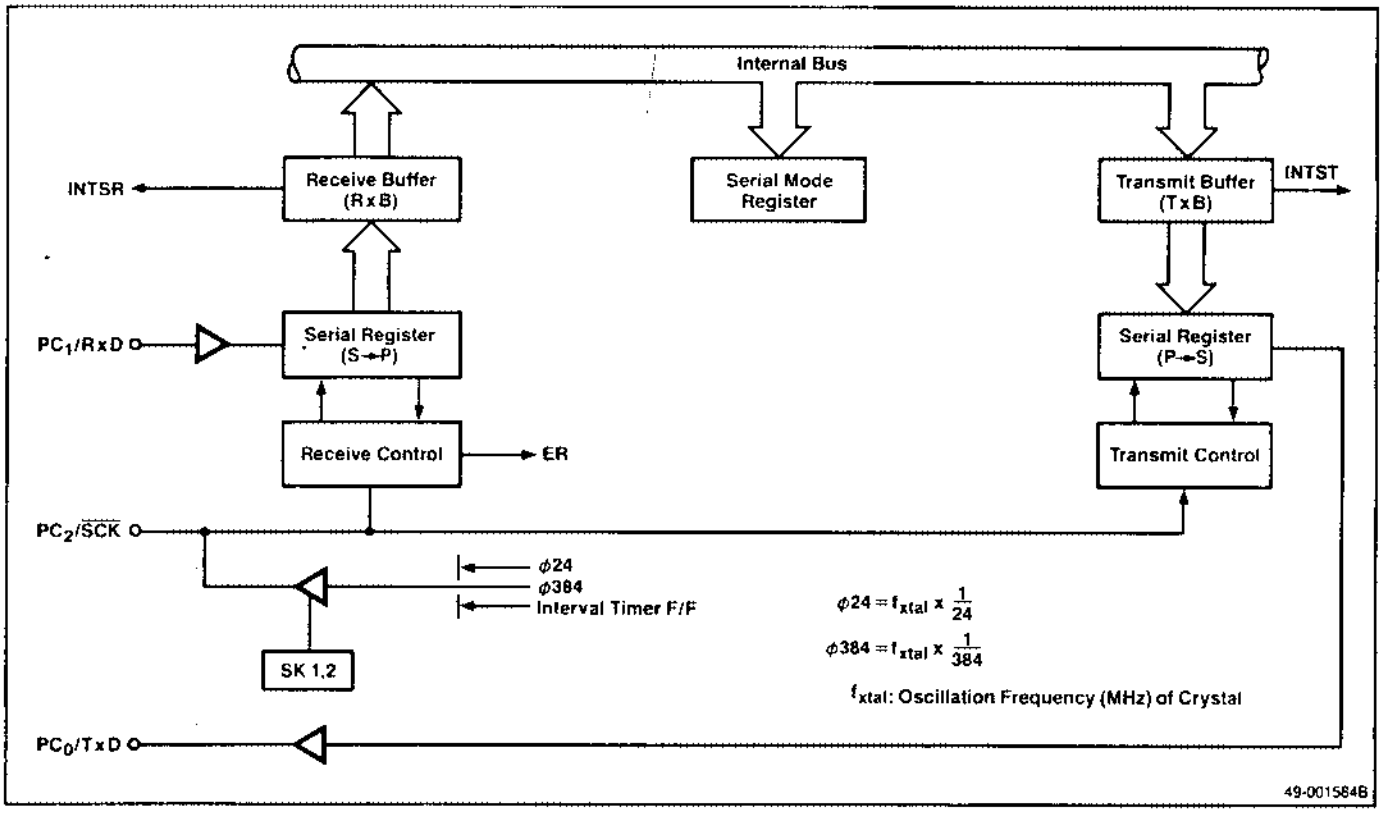
Receiver

Serial-to-Parallel Conversion Register. This register converts the serial data input from the RxD line into parallel data and transfers it to the receive buffer register.

Receive Buffer. This register temporarily holds the parallel data sent from the serial-to-parallel conversion register. When the receive buffer register is full, the INTSR interrupt request is generated.

Receive Control Circuit. This circuit controls the receiver and generates all the required control signals.

Figure 7-1. Block Diagram of Serial Interface Transmitter



SERIAL MODE REGISTERS

There are two serial mode registers: serial mode high-byte and serial mode low-byte. These specify the operating conditions (such as data length) of the serial interface.

Serial Mode High-Byte

Figure 7-2 shows the format of the serial mode high-byte register. The function of each bit is described as follows.

SK_2-SK_1. These bits determine the clock source by selecting either an internal clock (ϕ24, ϕ384, or interval timer F/F) or an external clock. When ϕ24 or ϕ384 internal clock is specified, the serial clock rate f_{SCK} is based on the following equations in which f_{XTAL} is the crystal oscillation frequency.

(1) Internal clock ϕ24:
$f_{SCK} = f_{XTAL} \div 24$

(2) Internal clock ϕ384:
$f_{SCK} = f_{XTAL} \div 384$

When the clock source is the interval timer F/F, the serial clock rate f_{SCK} is based on the following equations. (C represents the interval timer count value.)

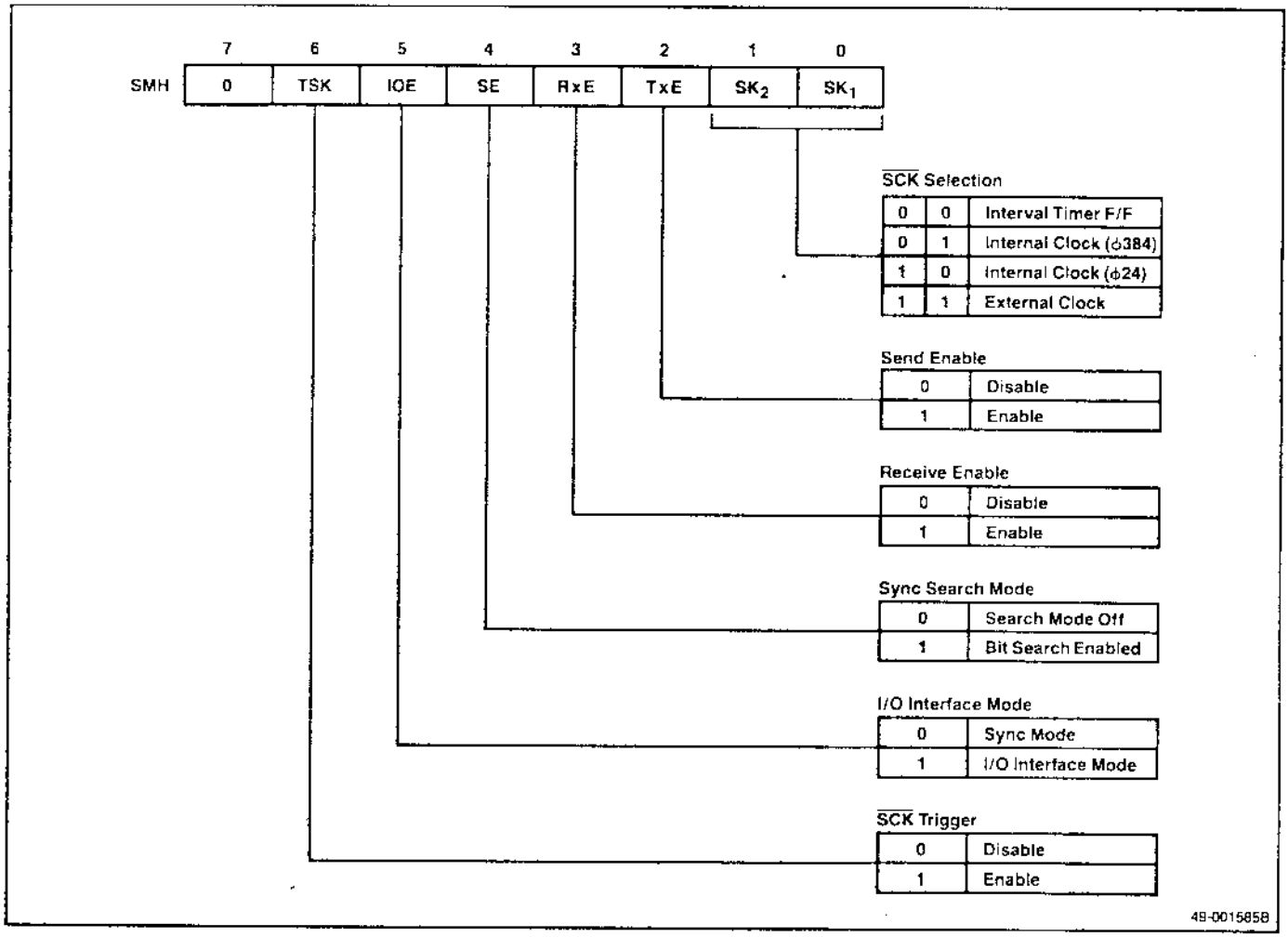
(1) Timer upcounter input clock = ϕ12:
$f_{SCK} = f_{XTAL} \div 24C$

(2) Timer upcounter input clock = ϕ384:
$f_{SCK} = f_{XTAL} \div 768C$

(3) Interval timer F/F clock = ϕ3 (timer not counting):
$f_{SCK} = f_{XTAL} \div 6$

TxE. The transmitter enable bit controls whether or not data is to be sent. If this bit is cleared, the TxD line is driven high, preventing data from being sent out on the line. When this bit is set, either newly written or previously stored serial data in the transmit buffer register can be sent out on the TxD line.

Figure 7-2. Serial Mode High-Byte Register



If the logic state of the TxE bit changes from a one to a zero, data transmission is inhibited after the serial data currently held in the parallel-to-serial conversion register is transmitted, preventing the transmission of any serial data held in the transmit buffer register. This data will not be lost, but will remain in the transmit buffer register until transmission is enabled (i.e., when the logic state of TxE changes to a one).

RxE. The receive enable bit controls whether or not serial data is to be received. When set, reception is enabled; when cleared, reception is disabled.

SE. This bit enables the search mode during synchronous data transmissions. When SE is set, the contents of the receive serial-to-parallel conversion register are transferred to the receive buffer register each time a data bit is received (search mode). This generates the INTSR serial receive interrupt. When SE is cleared (search mode off), the contents of the serial-to-parallel conversion register are transferred to the receive buffer register after a byte of serial data has been received (byte mode). This generates the INTSR serial receive interrupt.

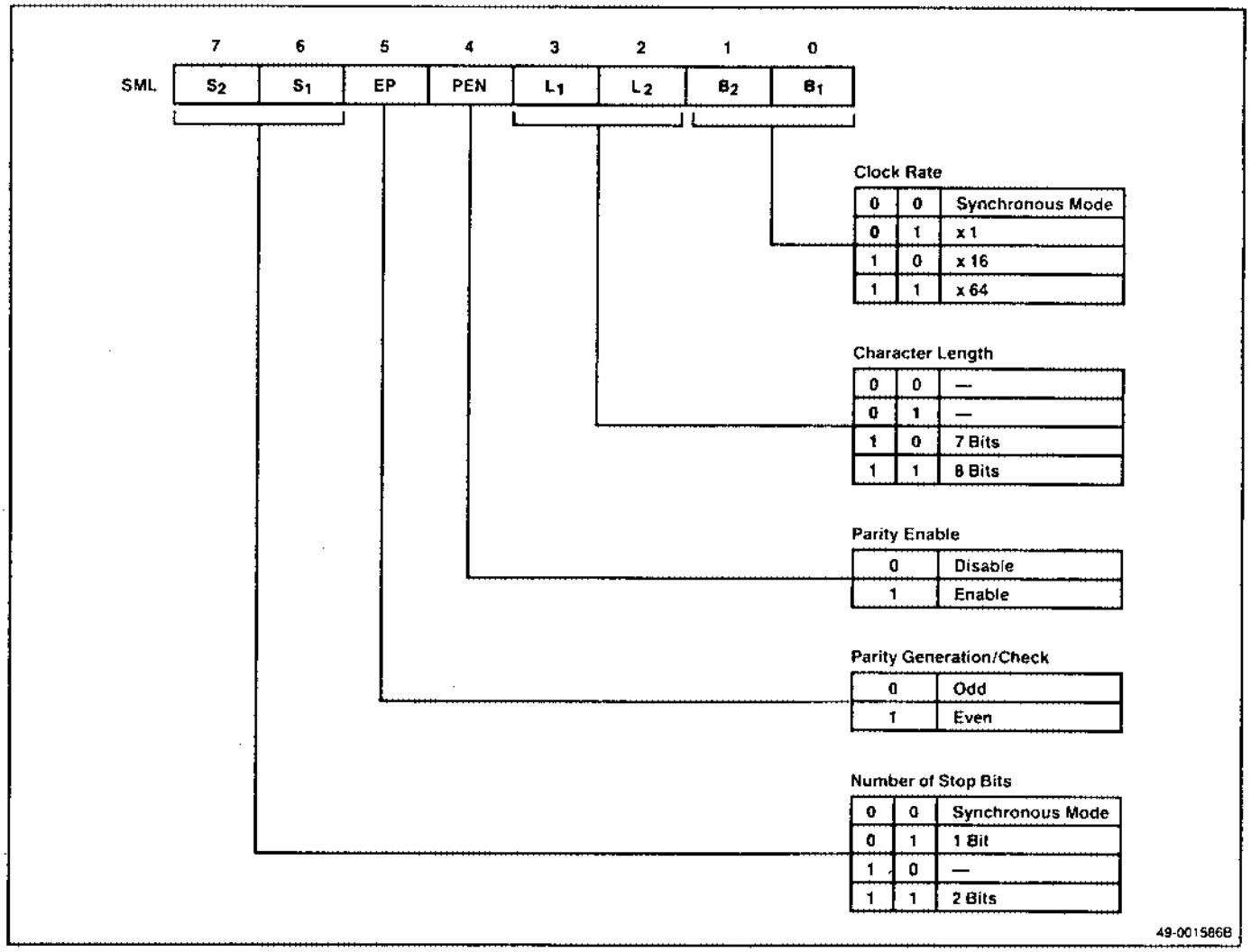
IOE. Two bits in the serial mode low-byte register (SML) select between synchronous or asynchronous operation. When synchronous operation is selected in the SML, the IOE (input/output enable) bit in the SMH selects between synchronous mode or I/O interface mode. When this bit is cleared, synchronous mode is enabled; when set, the I/O interface mode is selected.

TSK. This bit starts the internal serial clock when data is to be received in I/O interface mode. When the TSK bit is set, eight clock pulses are generated and a data bit is read on the rising edge of each clock pulse. After eight clock pulses, reception is completed and the TSK bit is automatically cleared.

Serial Mode Low-Byte

Figure 7-3 shows the format of the serial mode low-byte register. The function of each bit is described as follows.

Figure 7-3. Serial Mode Low-Byte Register



B_2-B_1. These bits select either asynchronous or synchronous mode. B_2 and B_1 = 00 selects the synchronous mode.

L_2-L_1. These bits select the character length.

PEN. This bit enables or disables parity. When PEN is set, a parity bit is added to each character transmitted and a parity check is conducted on all data received. An error flag is set if any errors are encountered. When this bit is cleared, the parity operation is disabled.

EP. This bit selects between even (EP = 1) or odd (EP = 0) parity.

S_2-S_1. These bits select the number of stop bits transmitted in asynchronous mode. In the synchronous mode, the bits should be set to 00.

OPERATION

The serial interface supports three methods of data transmission: asynchronous mode, synchronous mode, and I/O interface mode.

Asynchronous Mode

This mode of operation uses start and stop bits to synchronize the transmission and reception of data. To enable this mode, several operating parameters such as length of transmission characters, clock rate, number of stop bits, parity selection, etc., must be established using the serial mode registers as shown in figure 7-4.

When an internal clock source is specified, the data transfer rate B in bits/second is determined by the clock rate using the following formulas in which:

f_{XTAL} = crystal oscillation frequency in Hz
n = clock rate division (16 or 64)
C = timer count value

(1) Internal clock ϕ24:
$B = f_{XTAL} \div 24n$

(2) Internal clock ϕ384:
$B = f_{XTAL} \div 384n$

(3) Interval timer F/F when the timer upcounter clock source is internal clock ϕ12:
$B = f_{XTAL} \div 24nC$

(4) Interval timer F/F when the timer upcounter clock source is internal clock ϕ384:
$B = f_{XTAL} \div 768nC$

(5) When the interval timer F/F clock source is internal clock ϕ3:
$B = f_{XTAL} \div 6n$

Table 7-1 shows the timer count values (C) required to set up data transfers at baud rates ranging from 110 to 9600 b/s. The clock source to the timer upcounter is ϕ12 ($B = f_{XTAL} \div 24nC$).

Table 7-1. Using Timer Count Values To Set Up Baud Rates

Data Transfer Rate, b/s	Timer Count Value (C)			
	f_{XTAL} = 7.3728 MHz		f_{XTAL} = 11.0592 MHz	
	n = 16	n = 64	n = 16	n = 64
9600	2		3	
4800	4	1	6	
2400	8	2	12	3
1200	16	4	24	6
600	32	8	48	12
300	64	16	96	24
150	128	32	192	48
110	175	44	262	65

Data Transmission. Data transmission is enabled in asynchronous mode by setting the TxE bit in the serial mode high-byte register (SMH). The MOV TXB,A instruction writes data from the accumulator into the transmit buffer. When the contents of the parallel-to-serial register have been transmitted, the new contents are automatically transferred to the parallel-to-serial conversion register. One or two stop bits and a start bit are added to the serial data. A parity bit (odd or even) may be added if desired. The completed character is then sent out on the TxD line, beginning with the least-significant bit (LSB). See figure 7-5.

A serial transmission interrupt request (INTST) is generated when the transmit buffer register is empty. The interrupt may be disabled by setting the MKST bit of the MKH interrupt mask register. If additional data is written into the transmit buffer register before the register has become empty, the original contents are overwritten and destroyed. Therefore, when writing to the transmit buffer register, check that the serial transmission interrupt request flag (INTFST) is set, indicating that the transmit buffer is empty.

Figure 7-5. Data Format In Asynchronous Mode

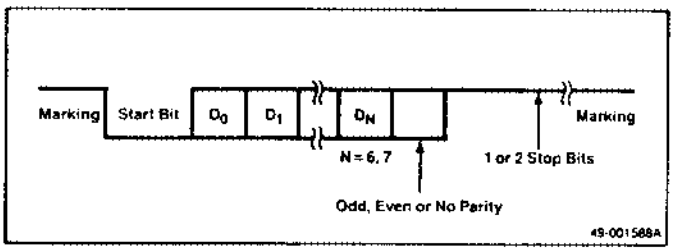
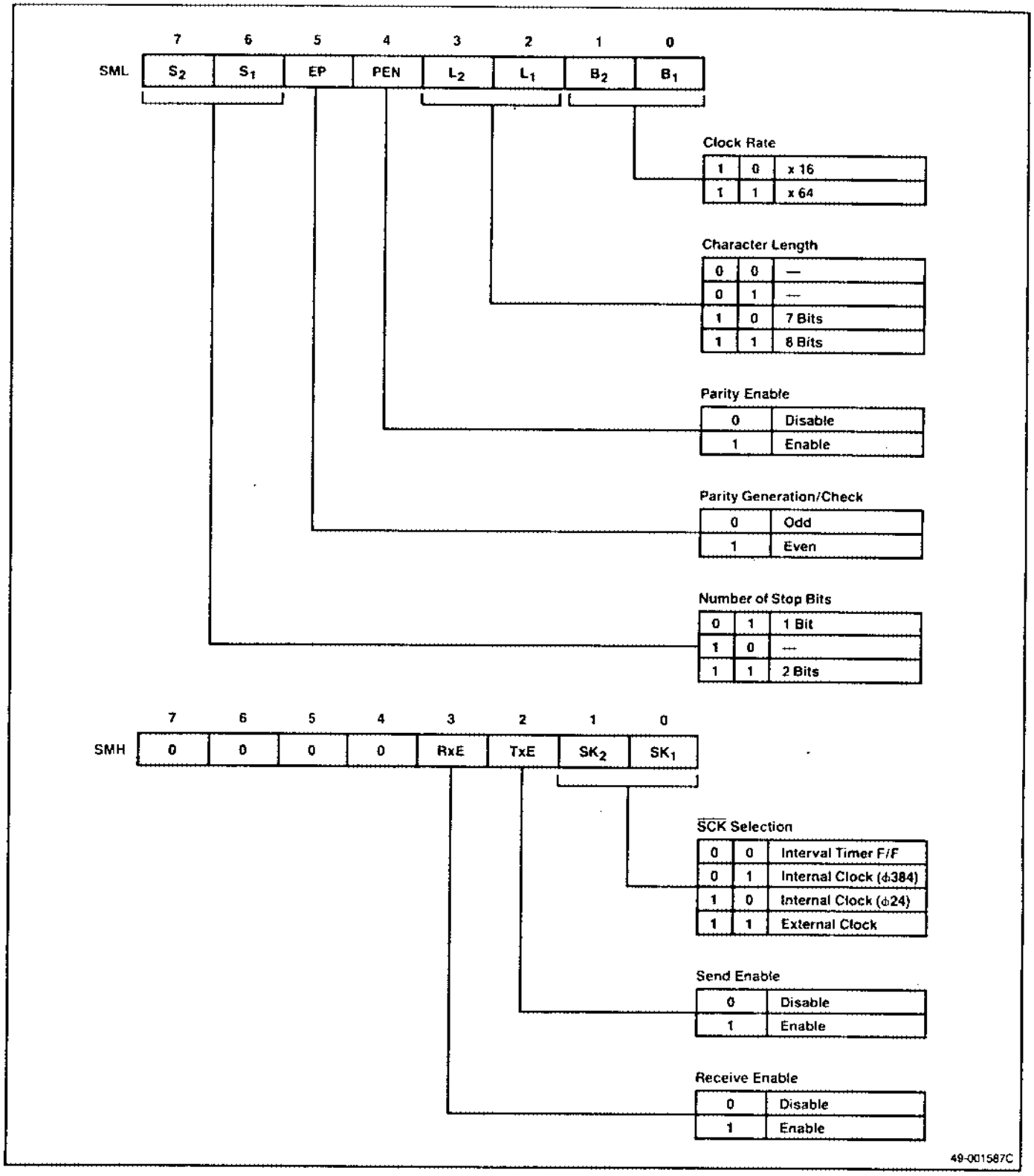


Figure 7-4. Format of Serial Mode Registers in Asynchronous Mode



When the TxE bit in SMH is cleared and the transmit buffer is empty, data transmission on the TxD line is disabled. When transmission is enabled, data is sent out on the TxD line at the fall of $\overline{SCK}$ at a rate of 1/16 or 1/64 of the serial clock frequency. The equivalent data transfer rates based on a master clock of 12 MHz are shown in table 7-2.

Data Reception. Data reception is enabled in asynchronous mode by setting the RxE bit in the serial mode high-byte register (SMH). There is a valid start bit when a low level on the RxD input is detected 1/2 bit time after the initial detection of a low level on the RxD input line. This start bit procedure prevents data reception from being incorrectly initiated by noise on the RxD line. After a valid start bit, character, parity, and stop bits are sampled every bit time starting from the center of the start bit.

Data received from the RxD line is written into the serial-to-parallel conversion register and transferred to the receive buffer. When the receive buffer is full, an INTSR interrupt request is generated. This interrupt can be masked by setting the MKSR bit in the interrupt mask register. The error flag (ER) in the interrupt flag register is set if a parity, framing, or overrun error occurs. Because no interrupts are generated by an error condition, the error condition must be tested using the SKIT or SKNIT skip instruction.

The equivalent data transfer rates based on a master clock of 12 MHz are shown in table 7-2.

Table 7-2. Clock Rates and Equivalent Data Transfer Rates

	Internal Clock		External Clock	
Clock Rate	$\overline{SCK}$	Data Transfer Rate	$\overline{SCK}$	Data Transfer Rate
x1 (1)	500 kHz (φ24)	500 kb/s	660 kHz 1 MHz	660 kb/s 1Mb/s (2)
x16 x64	2 MHz (φ3)/2	125 kb/s 31.25 kb/s	2 MHz	125 kb/s 31.25 kb/s

Notes:

(1) At a clock rate of x1 serial clock frequency, RxD and $\overline{SCK}$ must be synchronized externally.

(2) Two stop bits are required when data is being received at data transfer rates ranging from 660 kb/s to 1 Mb/s.

Synchronous Mode

In the synchronous mode, data transfers are synchronized with the serial clock. Character lengths are fixed at 8 bits, and there is no parity. Figure 7-6 shows the format of the serial mode registers in synchronous mode.

As shown in figure 7-7, data is transmitted in 8-bit quantities beginning with the least-significant bit (LSB) at the falling edge of $\overline{SCK}$. In synchronous mode, data is received at the rising edge of $\overline{SCK}$.

Data Transmission. Data transmission in synchronous mode is enabled by setting the TxE bit in the SMH register. The data to be transmitted is written into the transmit buffer by the MOV TXB,A instruction. When the previous data transmission has been completed, the new contents of the transmit buffer are automatically transferred to the parallel-to-serial conversion register, converted into serial data, and sent out on the TxD line at the falling edge of $\overline{SCK}$. The transmission data rate is equal to the $\overline{SCK}$ clock rate.

After all data has been transferred from the transmit buffer, an INTST interrupt request is generated. The interrupt can be masked by setting the MKST bit in the interrupt mask register. Data transmission is disabled on the TxD line when the TxE bit in SMH is cleared. The TxD line will be marking when the transit buffer is empty or when TxE is 0.

Data Reception. Data reception in synchronous mode is enabled by setting the RxE bit in the SMH register. Synchronously transmitted serial data is received on the rising edge of $\overline{SCK}$. There are two different methods by which synchronous data may be received: bit search mode and byte mode. The SE bit in the SMH register determines the method of data reception.

When the SE bit is set, the contents of the serial-to-parallel conversion register are transferred to the receive buffer each time a bit is received at the RxD pin, generating the INTSR serial receive interrupt. However, when the SE bit is cleared, the contents of the serial-to-parallel conversion register are not transferred to the receive buffer register until 8 bits have been received, at which time the INTSR serial receive interrupt is generated. In either case, the INTSR interrupt can be masked by setting the MKSR bit in the interrupt mask register.

For bit-oriented protocols, SE is set and SYNC detection is done on a bit-basis by a search program. Once SYNC is detected, the SE bit is reset and byte-long characters are received. For byte-oriented protocols, SE is always reset.

Figure 7-6. Format of Serial Mode Registers in Synchronous Mode

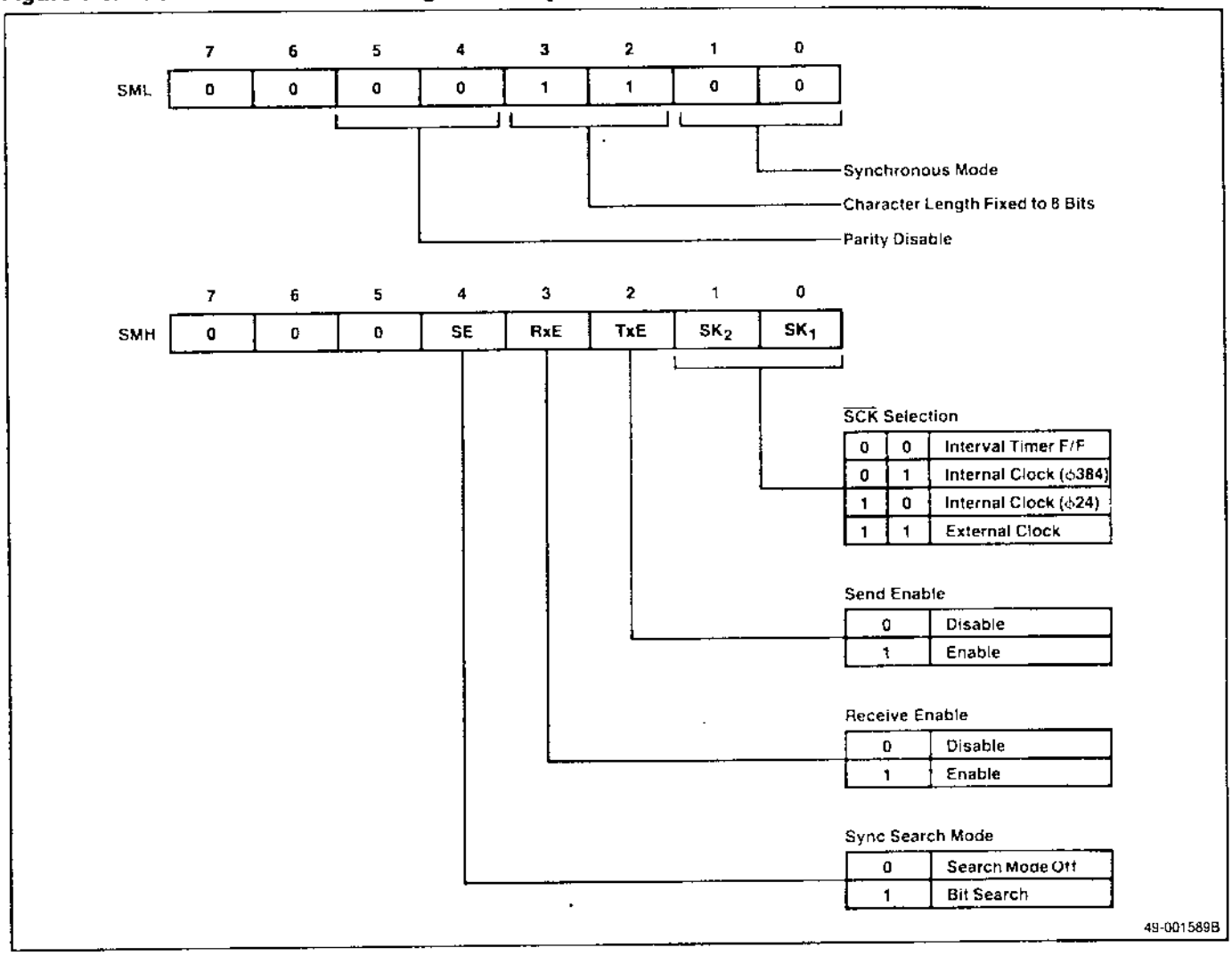
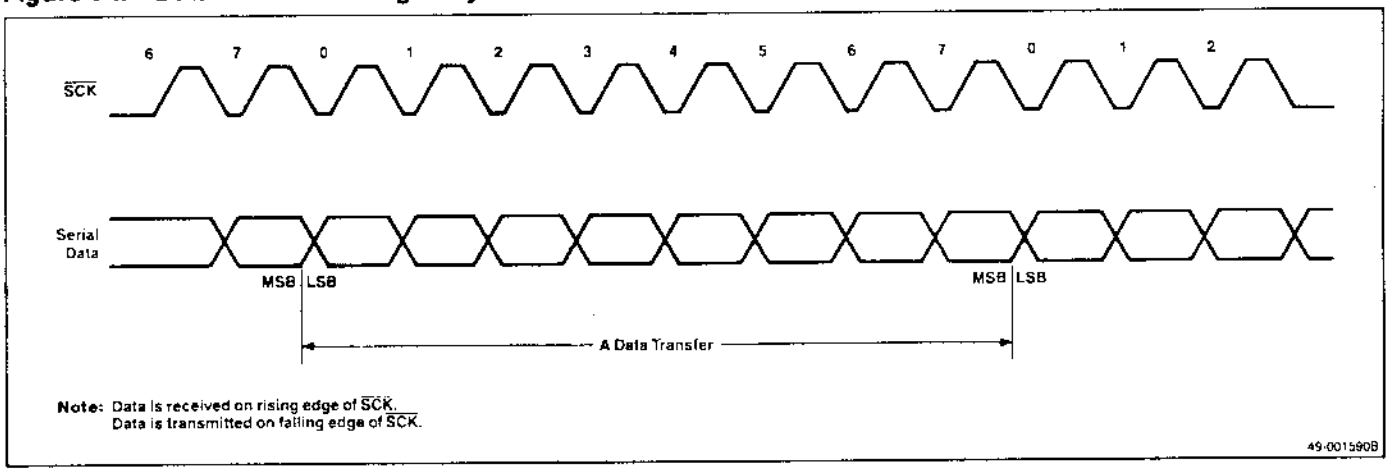


Figure 7-7. Data Transfer Timing in Synchronous Mode



I/O Interface Mode

In I/O interface mode, data transfers are controlled by the serial clock, which may be external or internal. Character lengths are fixed at 8 bits and no parity bits are added. Figure 7-8 shows how the serial mode register is formatted in I/O interface mode.

Data is transmitted at the falling edge of $\overline{SCK}$, beginning with the most-significant bit (MSB), and data is received in I/O interface mode at the rising edge of $\overline{SCK}$. Characters are synchronized by the serial clock, which may use either an internal or external clock source, as determined by the SMH register. When an internal clock source is used, eight clock pulses are output on the $\overline{SCK}$ line for each character transfer. When using an external clock source, a clock must be supplied that generates eight clock pulses and stops.

Figure 7-8. Format of Serial Mode Registers in I/O Interface Mode

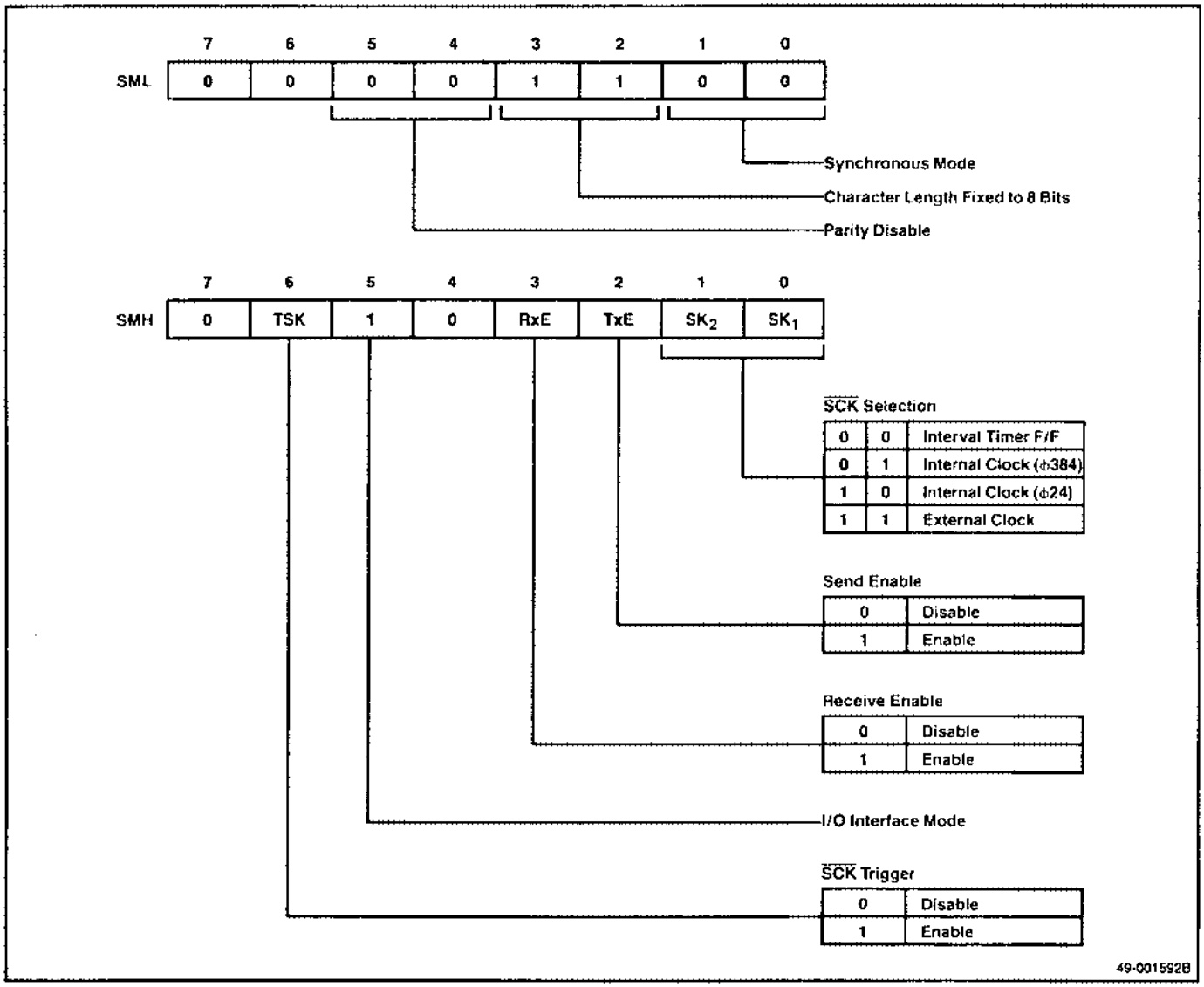


Figure 7-9 shows data transfer timing in I/O interface mode.

Data Transmission. Data transmission in I/O interface mode is enabled by setting the TxE bit in the SMH register. Data is written into the transmit buffer by a MOV TXB,A instruction, and when the data transmission of the previous byte has been completed, the transmit buffer contents are automatically transferred to the parallel-to-serial conversion register. Eight $\overline{SCK}$ pulses are automatically generated and the data is transmitted on the TxD line at the falling edge of $\overline{SCK}$, beginning with the most-significant bit.

When the contents of the transmit buffer have been transmitted, the INTST serial transmit interrupt request is generated. The interrupt may be masked by setting the MKST bit in the interrupt mask register.

Data Reception. Data reception in I/O interface mode is enabled by setting the RxE bit in the SMH register. If an internal clock source is used for the serial clock, the serial clock must be started by setting the TSK bit in the SMH register. The serial data is then received on the RxD line at the rising edge of $\overline{SCK}$. After 8 bits of data have been received into the serial-to-parallel conversion register, the data is transferred to the receive buffer and the INTSR serial receive interrupt request is generated. If an external clock source is used for the serial clock, all data sent synchronously with $\overline{SCK}$ is loaded into the serial-to-parallel conversion register at the rising edge of $\overline{SCK}$. The INTSR interrupt can be masked by setting the MKSR bit in the interrupt mask register. The high level of the eighth $\overline{SCK}$ must be at least six clock periods long.

PROGRAMMING EXAMPLE

To illustrate operation and programming concepts of the serial interface, an example of a program that transfers data asynchronously between the serial interface of the 7811 and a μPD8251A USART follows. Although references are made to the 7811, this example applies to all the parts in this manual.

In this example, the 7811 is set up as follows:

Clock oscillator	11.0592 MHz
Serial clock	Interval timer F/F
Data transfer rate	110 b/s
Clock rate division	16
Character length	8 bits
Number of stop bits	2
Parity	Even

As shown by the interconnection diagram in figure 7-10, three lines are needed to execute the transfer: a serial data input (RxD), a serial data output (TxD), and a control line (CTS). The PC_7 line is used as the CTS control line on the 7811 side.

Figure 7-10. Interconnection Diagram for Serial Data Transfer Example

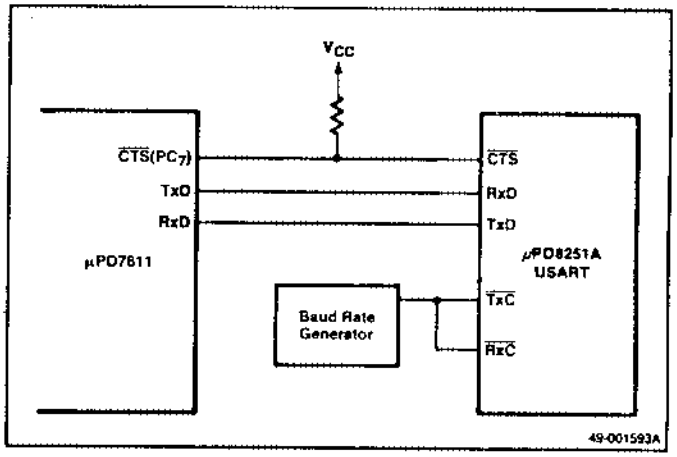
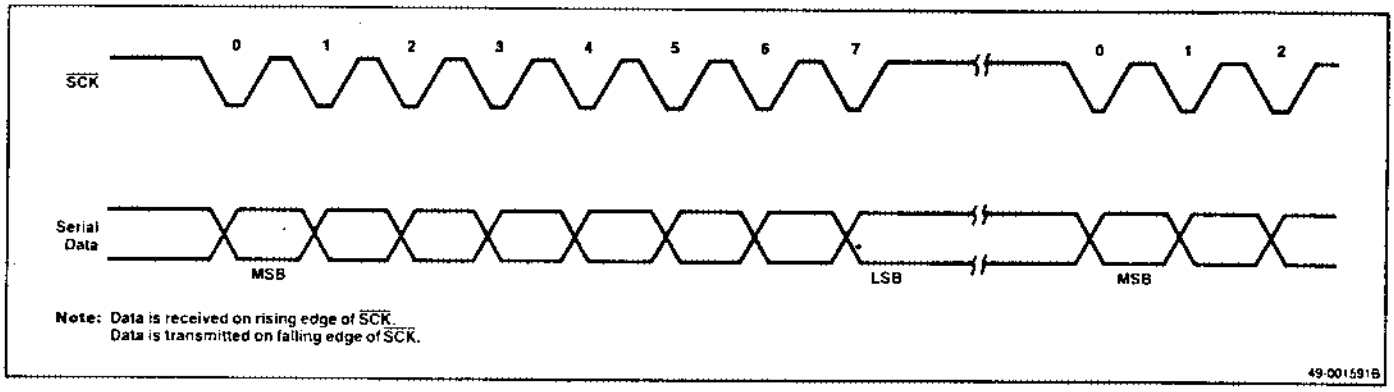


Figure 7-9. Data Transfer Timing In I/O Interface Mode.



Initialization

The first phase of the serial data transfer is initialization, which is outlined in figure 7-11 and described below in steps (a) through (d).

(a) Various operating parameters such as character length, clock rate, number of stop bits, serial clock source, parity-checking characteristics, etc., are specified in the serial mode registers as shown in figure 7-12.

(b) Because the serial clock uses the interval timer F/F as the clock source, counter operation parameters and the counter value are specified by the timer mode register as shown in figure 7-13. For this programming example, the count value is determined to be 262 using the following equation.

f = clock oscillation frequency (Hz)
n = clock rate division
B = data transfer rate (b/s)
C = count value

$$C = \frac{f}{24nB} = \frac{11.0592 \times 10^6}{24 \times 16 \times 110} = 262$$

Because the count value exceeds 255 (FFH), TIMER0 and TIMER1 are cascaded into a 16-bit timer configuration with TM0 = 131 (83H) and TM1 = 2.

Figure 7-11. Flowchart of Initialization Phase of Serial Data Transfer Example

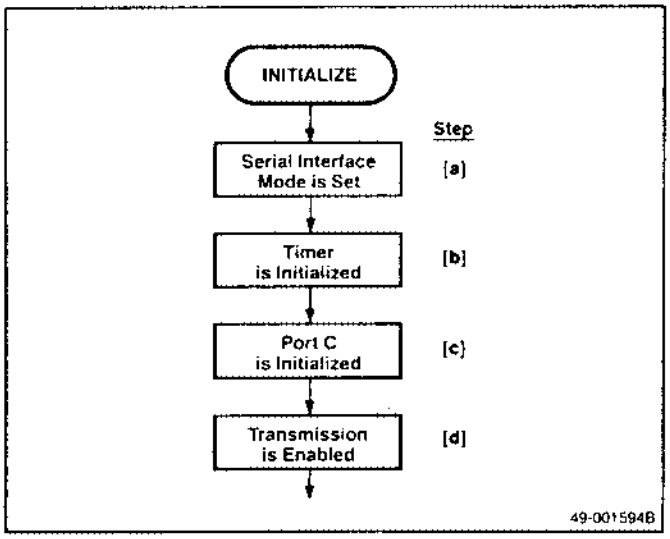
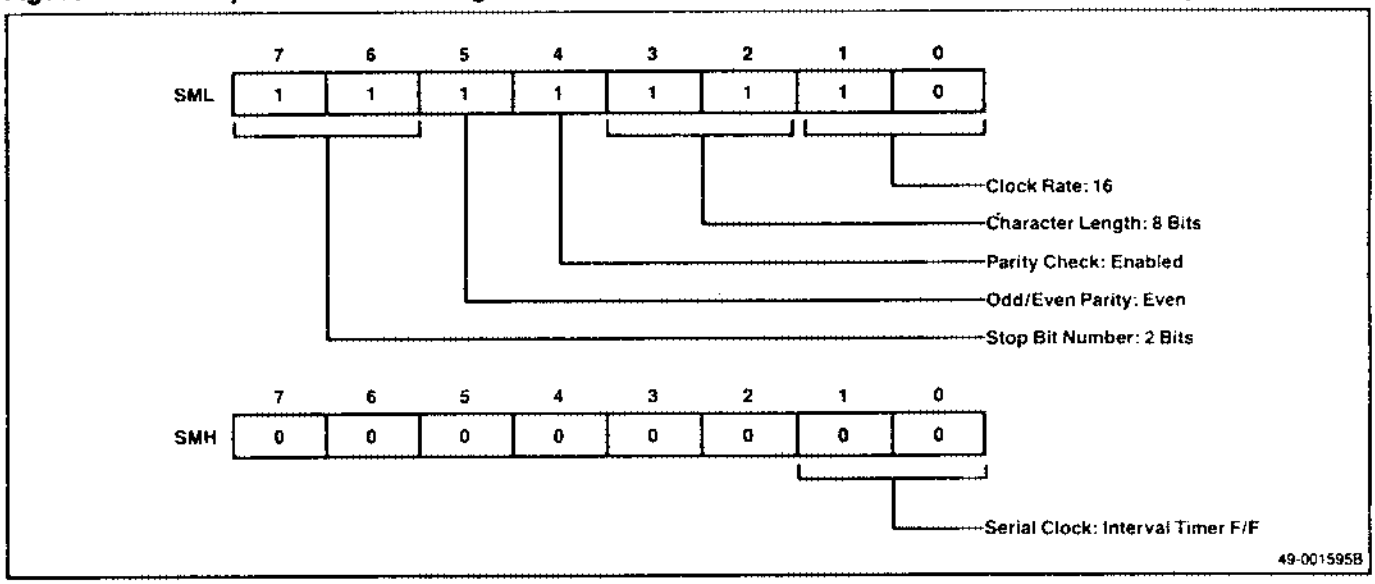


Figure 7-12. Setup of Serial Mode Registers for Initialization Phase of Serial Data Transfer Example



(c) As shown in figure 7-14, port C is initialized by establishing PC_0 as the TxD line, PC_1 as the RxD line, and PC_7 as an output port line outputting a high-level signal. PC_2 can be designated for serial clock input or output by the MCC register.

(d) Transmission is enabled by setting the TxE bit in the SMH register as shown in figure 7-15.

Figure 7-13. Setup of TMM for Initialization Phase of Serial Data Transfer Example

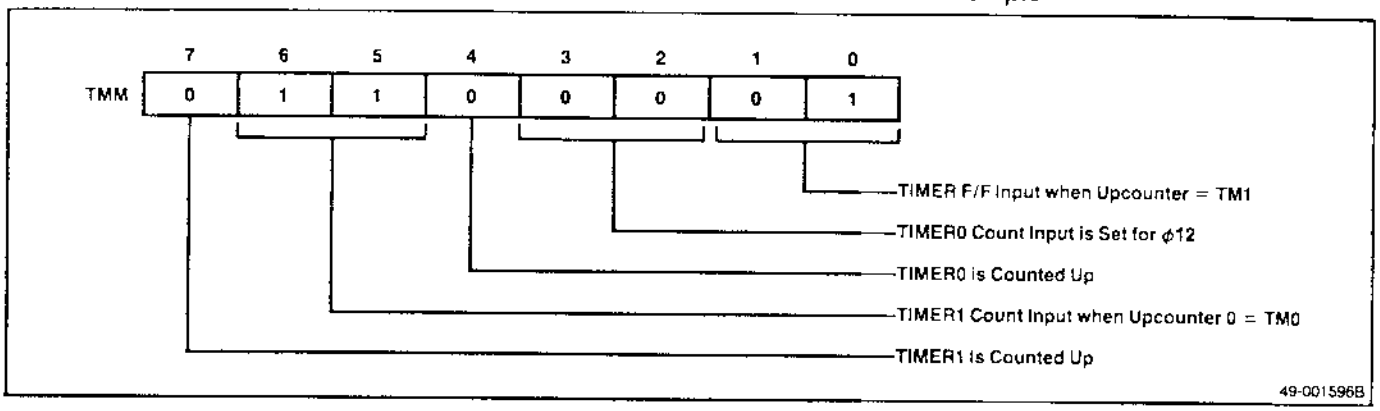


Figure 7-14. Port C Specification for Initialization Phase of Serial Data Transfer Example

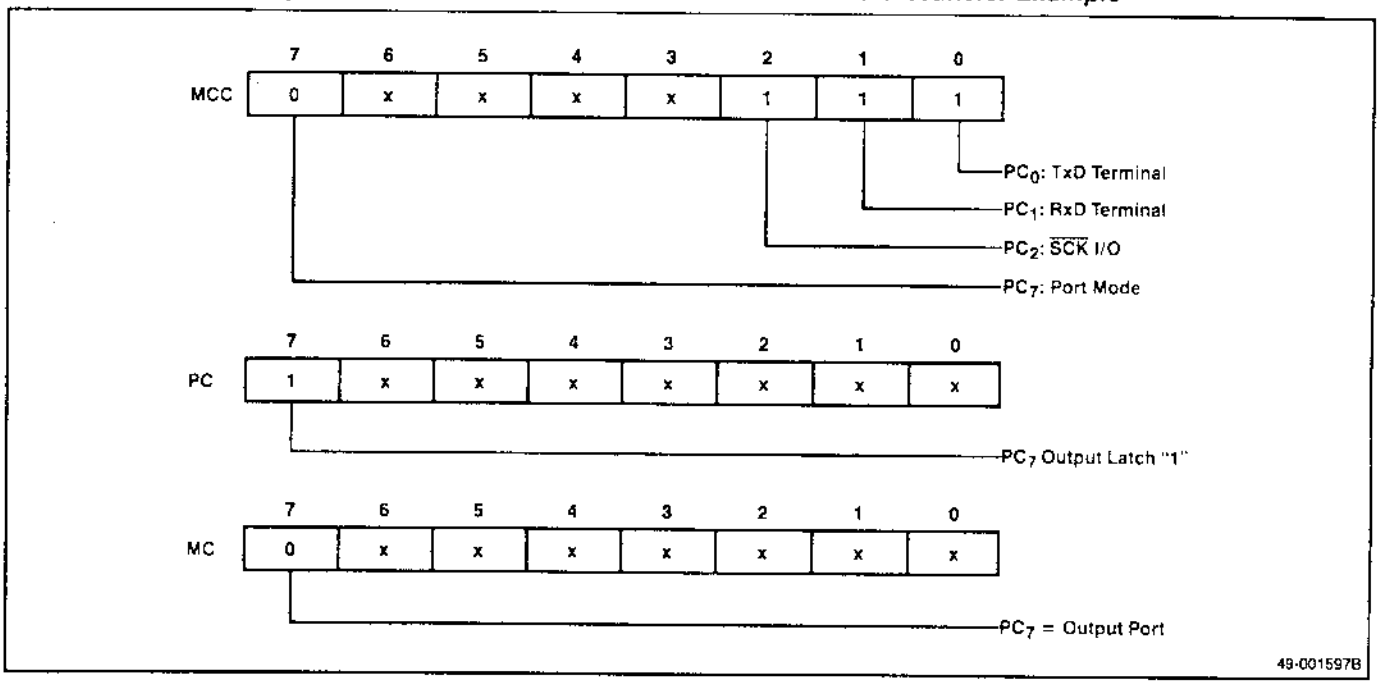


Figure 7-15. Enabling Transmission in Serial Data Transfer Example

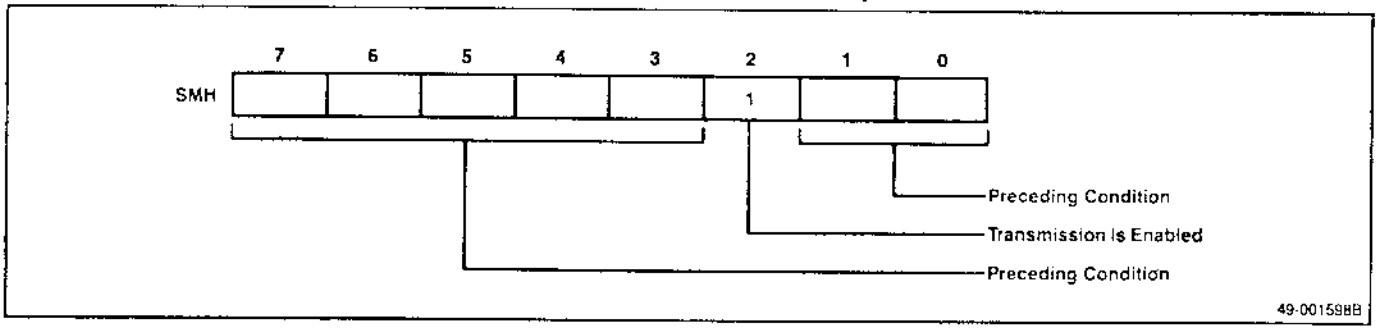


Table 7-3 is an example of program code for each step in the initialization phase of the serial data transfer.

Table 7-3. Example of Program Code for the Initialization Phase of Serial Data Transfer

```
;**********SERIAL INTERFACE INITIALIZATION**********
SINIT: MVI   SMH,00H    ;Internal serial clock (timer F/F)   Step (a)
       MVI   A,0FEH     ;x16, even parity, 8-bit
                        ;character, 2 stop bits.
       MOV   SML,A      ;Set serial mode.
       ------------------------------------------------------
       MVI   A,83H      ;                                    Step (b)
       MOV   TM0,A      ;Set timer register.
       MVI   A,02H      ;Baud rate = 100 b/s.
       MOV   TM1,A      ;
       MVI   TMM,61H    ;Set timer mode and start.
       ------------------------------------------------------
       MVI   A,07H      ;Set port C mode control.            Step (c)
       MOV   MCC,A      ;TxD, RxD, SCK available.
       ORI   PC,80H     ;PC7 output latch = 1.
       MVI   A,00H      ;Initialize port C.
       MOV   MC,A       ;Port C output mode.
       ------------------------------------------------------
       ORI   SMH,04H    ;Transmit enable.                    Step (d)
```

Data Transmission

In this phase of the serial data transfer example, the contents of the accumulator (one byte) are serially transmitted and the interrupt request flag (INTFST) is tested rather than using the INTST serial transmit interrupt. The sequence is outlined in figure 7-16 and described below in steps (a) and (b).

(a) To determine if the transmit buffer is empty, the INTFST interrupt request flag is tested.

(b) The contents of the accumulator are transferred to the transmit buffer register.

Table 7-4 is an example of program code for each byte transmitted. Interrupt masks are set in the data reception subroutine. This operation in the 7811 assumes that the μPD8251A USART has been enabled for the reception of serial data.

Table 7-4. Example of Program Code for Transmission Phase of Serial Data Transfer

```
;**********TRANSMIT SERVICE**********
TRNS:  SKIT   FST      ;Test FST, skip if FST = 1.          Step (a)
       JR     TRNS     ;Wait until FST = 1.
       ------------------------------------------------------
       MOV    TXB,A    ;Output transmit data.               Step (b)
```

Data Reception

Receive-Enable Phase. The INTSR interrupt is used for serial data reception, requiring the allocation of memory space for storage of received data, determination of the number of bytes to be received, setup of the interrupt mask register, etc., before serial data can be received.

The receive-enable phase of the serial data transfer example is outlined in figure 7-17 and described below in steps (a) through (e).

(a) Memory address where the received data is to be stored is specified by register pair HL. The address 2000H is specified in this example.

(b) Number of bytes to be received is specified by general-purpose register C. For this example, 16 (10H) bytes are specified.

Figure 7-16. Flowchart of Transmission Phase of Serial Data Transfer Example

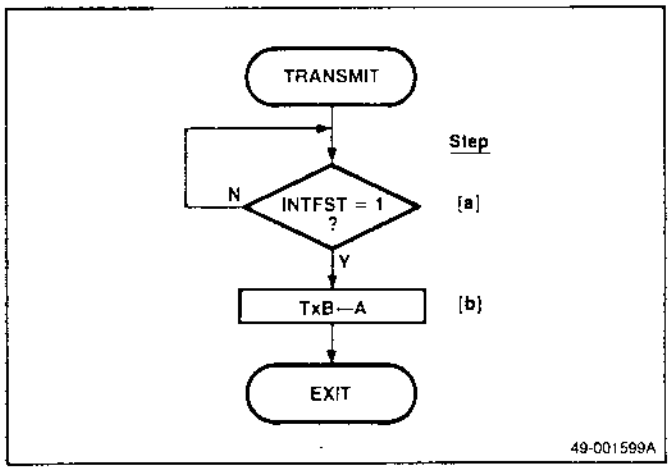
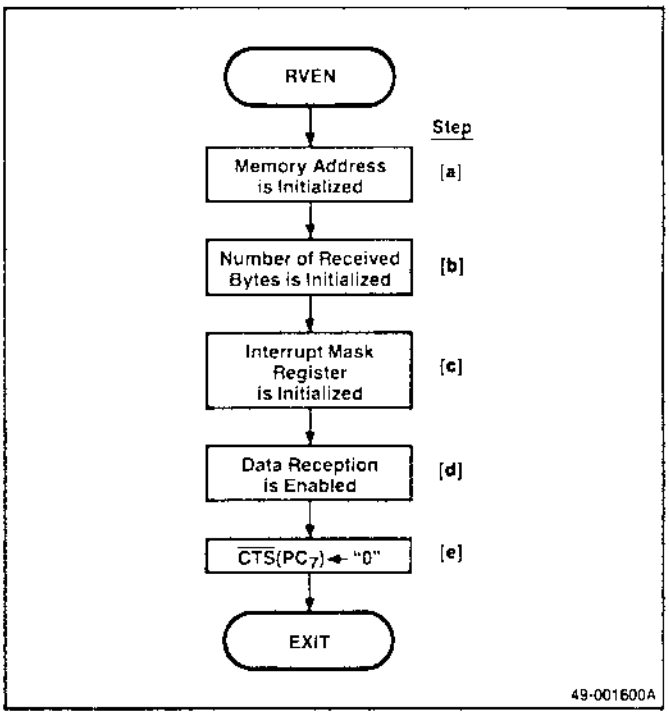


Figure 7-17. Flowchart of Receive-Enable Phase of Serial Data Transfer Example



(c) As shown in figure 7-18, the MKSR bit in the interrupt mask register is cleared, enabling the INTSR interrupt. The INTST interrupt mask is set, masking the INTST interrupt. (INTST and INTSR have the same priority.)

(d) RxE bit in the SMH register is set, enabling the reception of serial data.

(e) PC_7 line is set low, enabling the $\overline{CTS}$ line.

Table 7-5 is an example of program code for each step in the receive-enable phase of serial data transfer. After all required parameter settings for the receive enable phase have been made, the INTSR interrupt is generated each time the specified data string is received.

Table 7-5. Example of Program Code for Receive-Enable Phase of Serial Data Transfer

```
;**********RECEIVE ENABLE**********
RVEN: LXI  H,2000H    ;Set data pointer (HL = 2000H).      Step (a)
      MVI  C,0FH      ;Set data counter (C = 0FH).         Step (b)
      EXX             ;
      MVI  MKH,05H    ;INTST masked and INTSR unmasked.    Step (c)
      ORI  SMH,08H    ;Receive enabled.                    Step (d)
      ANI  PC,7FH     ;CTS = 0.                            Step (e)
```

Interrupt Phase. The INTSR interrupt steps listed below are outlined in figure 7-19.

(a) Received data is checked for errors. If an error is found, a jump is made to the error processing routine.

(b) Received data is stored in memory.

(c) A check is made to determine if the receive buffer is full. If not, processing continues through the main stream of the program.

(d) If the receive buffer is full, the PC_7 line is driven high, disabling the $\overline{CTS}$ line. As a result, transmission from the μPD8251A is stopped.

(e) RxE bit in the SMH register is cleared, disabling serial data reception.

(f) MKSR bit in the interrupt mask register is set, disabling INTSR interrupts.

Figure 7-19. Flowchart of INTSR Interrupt Processing Phase of Serial Data Transfer Example

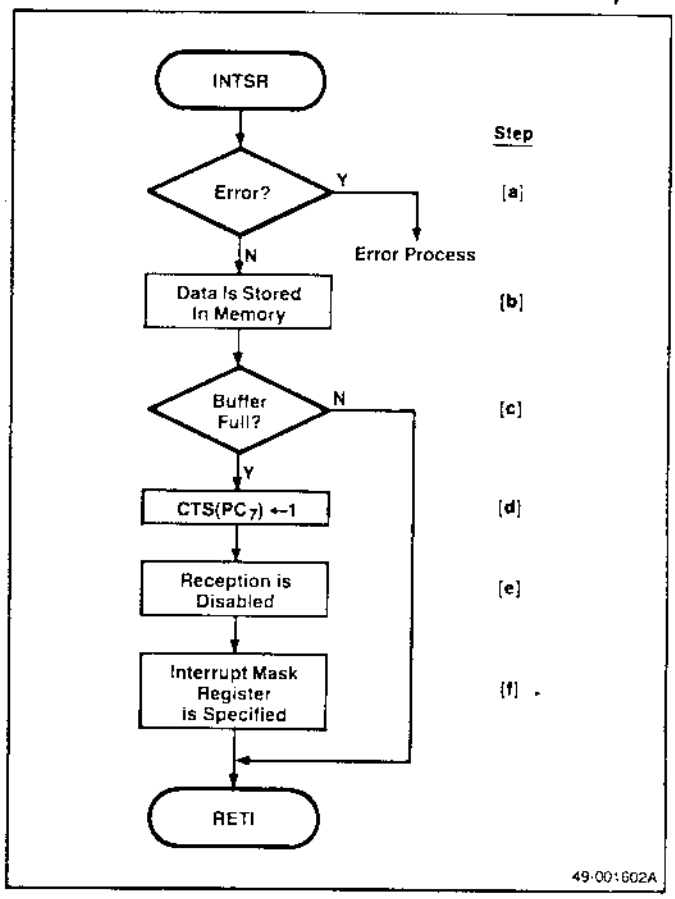


Figure 7-18. Setup of Interrupt Mask Register in Receive-Enable Phase of Serial Data Transfer Example

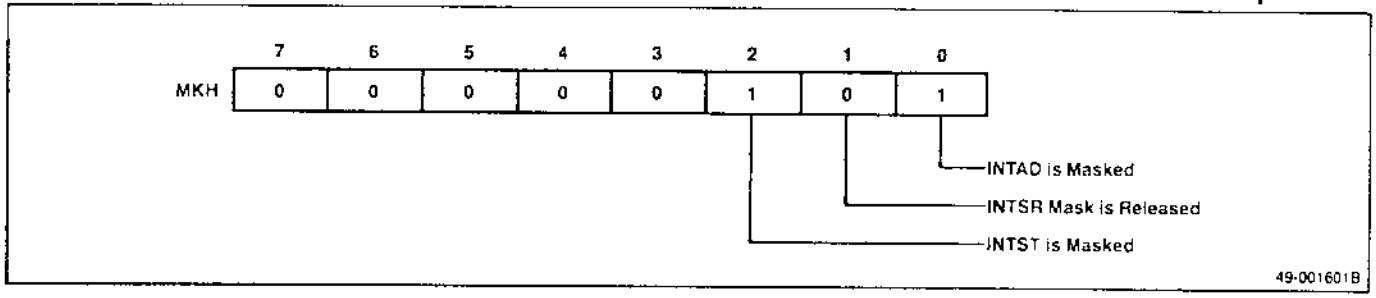


Table 7-6 is an example of program code for each step in the INTSR interrupt service routine. The JMP RECV instruction should be located at vector address 0028H.

Table 7-6. Example of Program Code for INTSR Interrupt Service Routine of the Serial Data Transfer Operation

```
;*****RECEIVE SERVICE**********
RECV: EXA                    ;Save accumulator.                    Step (a)
      EXX                    ;Save register.
      SKNIT   ER             ;Test ER flag; skip if ER = 0.
      JMP     ERROR          ;Jump to ERROR routine.
      MOV     A,RXB          ;Input received data.                 Step (b)
      STAX    H+             ;Store received data to memory.
      DCR     C              ;Skip next instruction if             Step (c)
                             ;buffer full.
      JR      RECO           ;
      ORI     PC,80H         ;CTS ← 1.                             Step (d)
      ANI     SMH,0F7H       ;Receive disable.                     Step (e)
      ORI     MKH,02H        ;INTSR disabled.                      Step (f)
RECO: EXX                    ;Recover register.
      EXA                    ;Recover accumulator.
      EI                     ;Enable interrupt.
      RETI                   ;Return
```

CONFIGURATION

An eight-channel, 8-bit, successive-approximation, analog-to-digital converter provides on-board signal processing capability. As shown in figure 8-1, the A/D converter (ADC) consists of an input circuit, a 256-resistor ladder, a voltage comparator, a successive-approximation register (SAR), and four conversion result registers (CR0-CR3). Two modes of analog input selection are provided: scan mode and select mode.

Conversion Process

The eight analog inputs are multiplexed internally, according to the contents of the A/D channel mode register (ANM). The selected analog input is sampled by the ADC, and sent to the voltage comparator where the difference between the instantaneous analog input voltage and the voltage tap in the resistor ladder is amplified.

The resistor ladder is connected between the ADC reference voltage terminal (V_{AREF}) and the ADC ground terminal (AV_{SS}). The resistor ladder decoder driven by an 8-bit successive-approximation register (SAR) selects the voltage tap in the resistor ladder that matches the analog input voltage.

As an example, if bit 7 in the SAR is set at the start of the conversion process, the voltage tap equivalent to 1/2 VAREF is designated, and the voltage of the current analog input is compared with this value. If the voltage exceeds the value, SAR_7 remains set; if the voltage is less than this value, SAR_7 is cleared.

Then bit 6 in the SAR is set, and the voltage tap equivalent to either 3/4 or 1/4 V_{AREF} (depending on whether or not bit 7 is set) is designated, and the voltage on the analog input is compared with this value.

When all 8 bits have been set or cleared in this manner, a voltage tap in the resistor ladder whose voltage matches the analog input will have been found. The 8-bit number in the SAR is the digital representation of the instantaneous analog input and it will be latched into one of the CR registers.

Power Supply

An independent power supply line, AV_{CC}, is used for the ADC for greater voltage stability and noise immunity. In all CMOS parts, if V_{AREF} is reduced to 0 volts, the A/D converter power is off. This is a useful feature in standby modes. Capacitors should be connected from V_{AREF} to ground and from each analog input to ground to decouple noise. Capacitor size should be 100 pF to 1000 pF.

A/D Channel Mode Register

The A/D channel mode register (ANM) is 8 bits long and is used to establish operating parameters for the ADC. The format of the register is shown in figure 8-2.

Figure 8-1. Block Diagram of Analog-to-Digital Converter

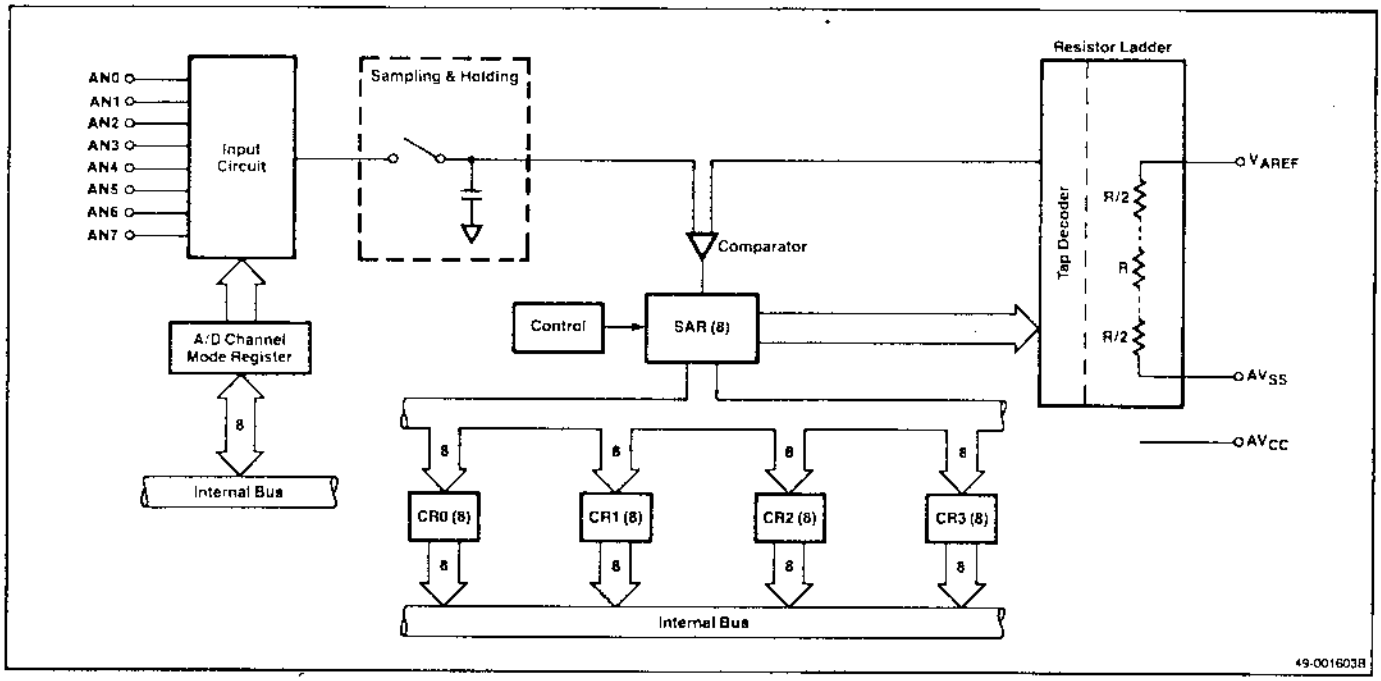
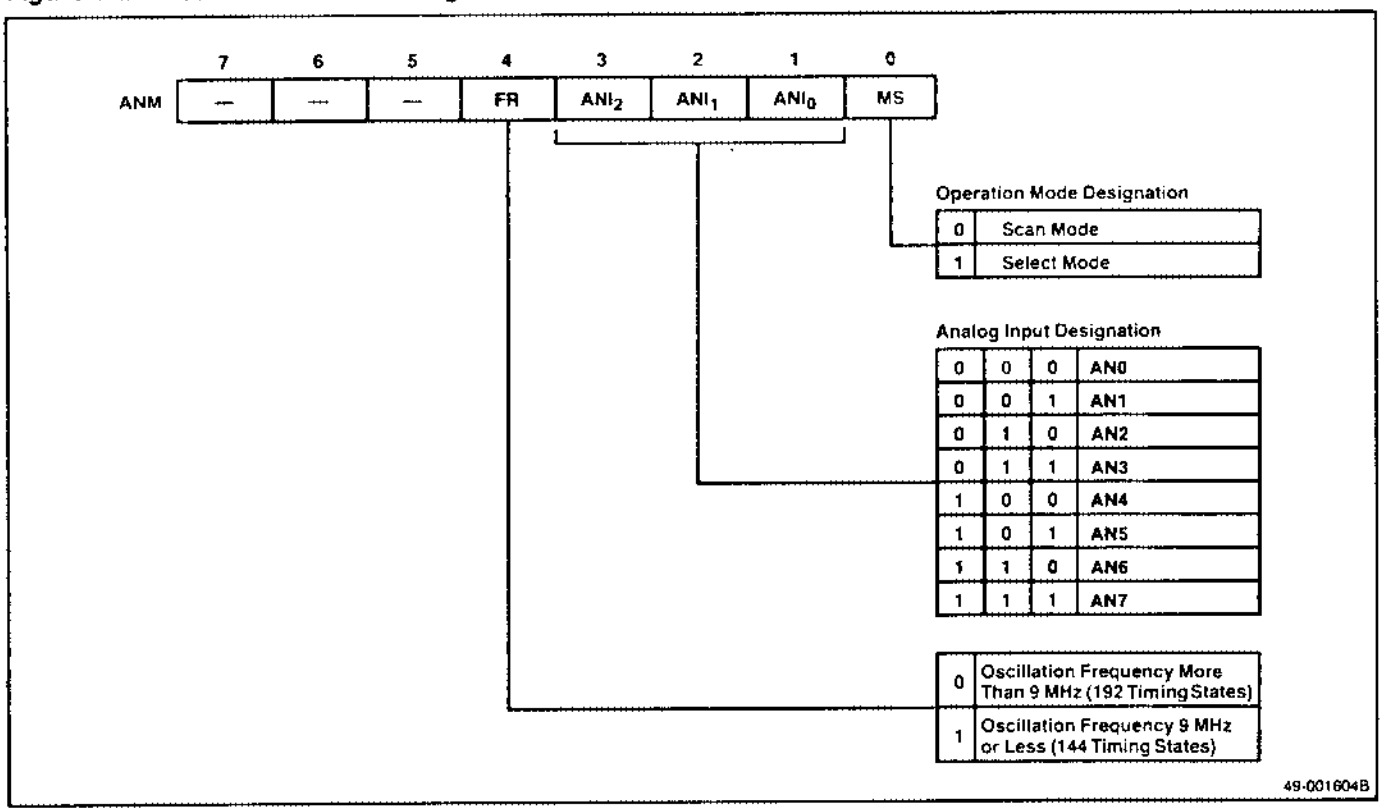


Figure 8-2. A/D Channel Mode Register



Bit 0 (MS) determines if scan mode or select mode is used. Bits 1, 2, and 3 (ANI_0-ANI_2) selectively enable the individual analog input lines. Optimum conversion frequency based on oscillation frequency f_{XTAL} is selected by bit 4 (FR) as follows.

FR = 0: Conversion rate = 192 timing states; f_{XTAL} above 9 MHz.

FR = 1: Conversion rate = 144 timing states; f_{XTAL} below 9 MHz.

Oscillation frequencies and corresponding conversion rates are shown in table 8-1. The ANM register can be read to determine which analog input is selected for the current conversion. A $\overline{\text{RESET}}$ clears the A/D channel mode register to 00H.

Table 8-1. Oscillation Frequencies and Conversion Rates

f_{XTAL} (MHz)	12	11	10	9	8	7	6
FR Bit	0	0	0	1	1	1	1
Conversion Speed (μs)	48	52.4	57.6	48	54	61.7	72

OPERATION

The ADC operates either in scan mode or select mode as designated by the MS bit in the A/D channel mode register (ANM.

Scan Mode

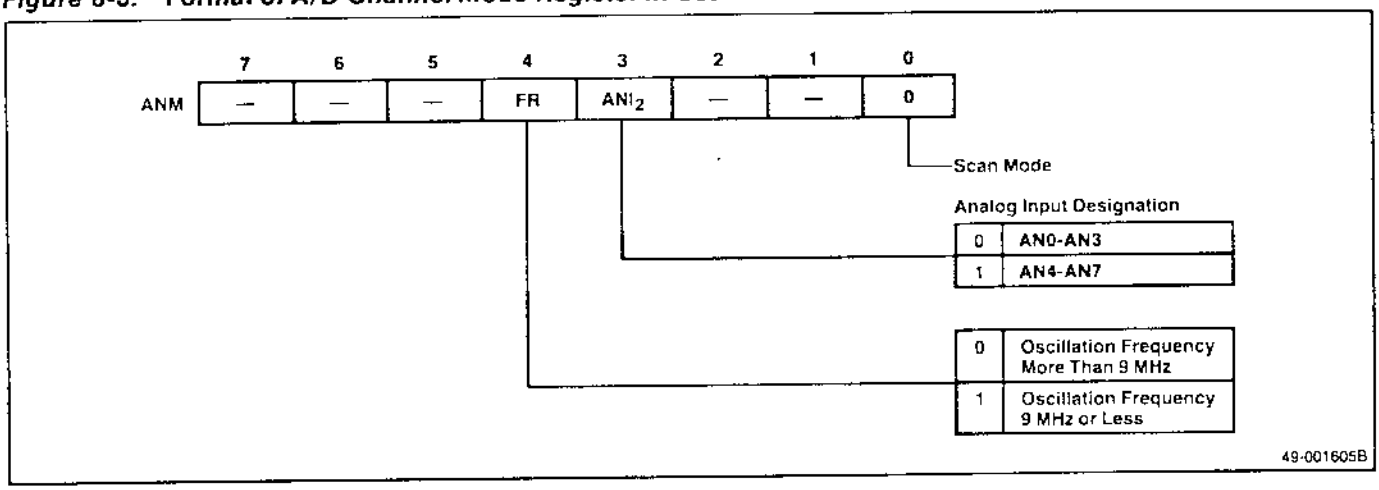
When bit 0 of ANM is cleared (MS = 0), scan mode is enabled. Bit 3 of the ANM (ANI_2) determines which four inputs are scanned. Analog input lines AN0-AN3 ($ANI_2 = 0$) or AN4-AN7 ($ANI_2 = 1$) are enabled in scan mode by the ANM register as shown in figure 8-3. Bits 1 and 2 have no function in scan mode.

When ANI_2 is cleared, analog input lines AN0-AN3 are enabled in order starting with AN0, and the digital equivalent of each converted analog value is stored in the respective conversion result register (i.e., the result from AN0 is stored in CR0, the result from AN1 is stored in CR1, etc.).

When ANI_2 is set, analog input lines AN4-AN7 are enabled in order starting with AN4. The results are stored in least-significant to most-significant order in the conversion result registers (i.e., the result from AN4 is stored in CR0, the result from AN5 in CR1, etc.).

When all four conversion result registers contain results, the INTAD interrupt request is generated. The INTAD interrupt can be masked by setting the MKAD bit in the interrupt mask register.

Figure 8-3. Format of A/D Channel Mode Register in Scan Mode



The conversion process is continued from either the AN0 or AN4 input, independent of whether or not the interrupt is accepted, with new results stored in the respective conversion result registers. This will continue until the ANM register is modified. In this mode, up to four analog signals can be converted to digital form with a minimal amount of software.

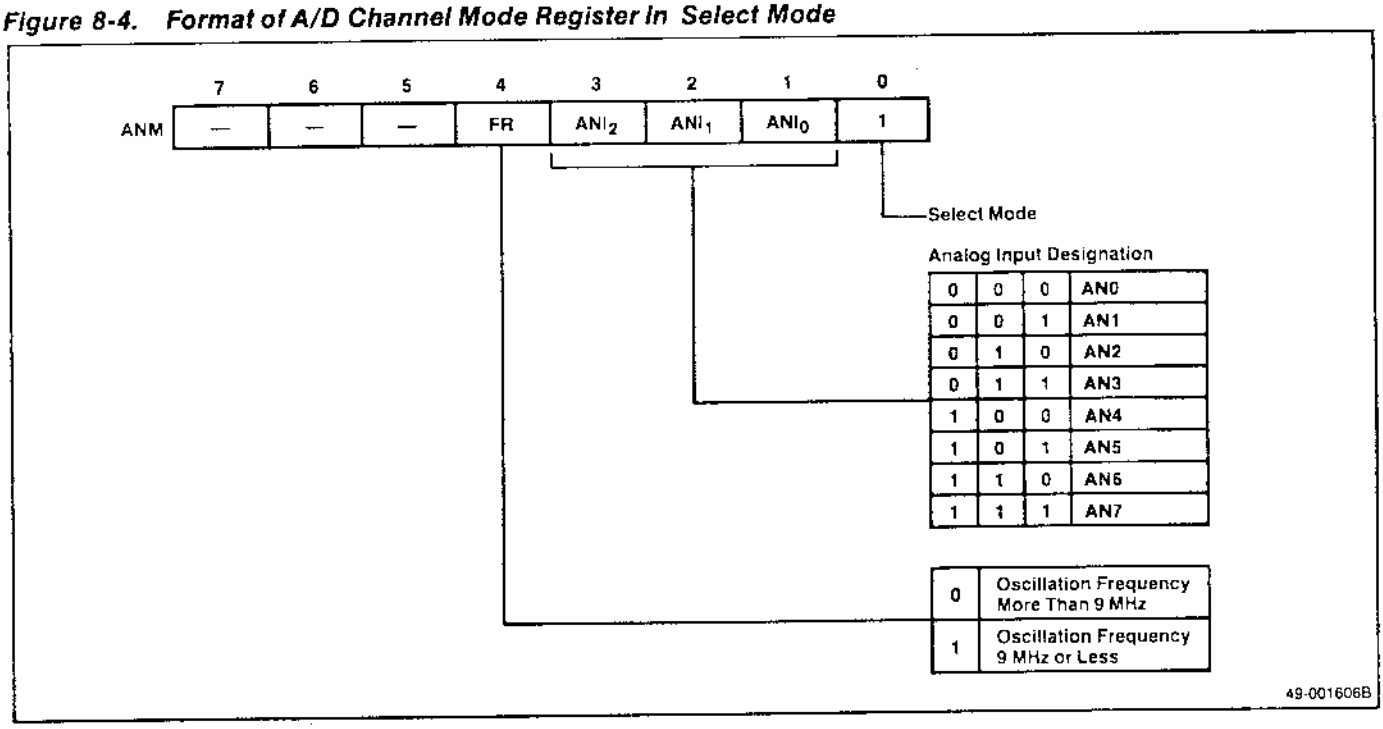
Select Mode

If bit 0 of ANM is set (MS = 1), select mode is enabled. ANM bits 1-3 (ANI_0-ANI_2) determine which of the eight analog inputs will be converted (see figure 8-4).

The analog input from the line designated in the ANM register is converted and stored in a conversion result register. The first conversion result is stored in CR0, the second in CR1, etc., until all four registers hold conversion results, at which time an INTAD interrupt is generated. This interrupt may be masked by setting the MKAD bit in the interrupt mask register.

New results are continually stored in the conversion registers as they are processed, regardless of whether or not the interrupt is acknowledged, so that the conversion registers always contain the most recent conversion results. This operation is continued until the ANM register is modified. This mode of operation is particularly useful in maximizing noise immunity or in averaging conversion signals.

Figure 8-4. Format of A/D Channel Mode Register In Select Mode



PROGRAMMING EXAMPLE

The following example illustrates operation and programming concepts of the ADC. The scan mode is used and the conversion results from analog lines AN0-AN7 are each sampled eight times and stored in memory locations 1000H-103FH (figure 8-5).

The signals on analog input lines AN0-AN3 are converted to digital form, and the results are stored in the following locations.

AN0	1000H-1003H
AN1	1008H-100BH
AN2	1010H-1013H
AN3	1018H-101BH

Next, the signals on analog input lines AN4-AN7 are converted to digital form, and the results are stored in the following locations

AN4	1020H-1023H
AN5	1028H-102BH
AN6	1030H-1033H
AN7	1038H-103BH

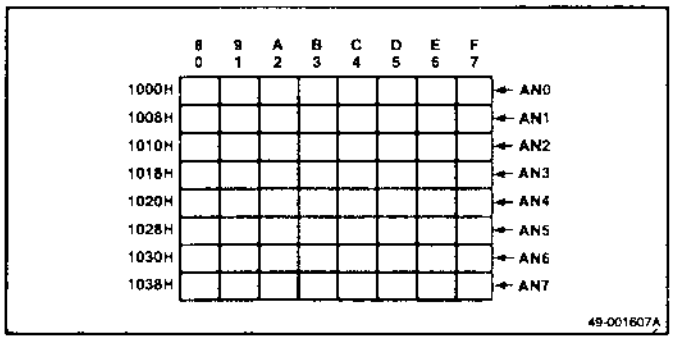
Once again, the signals on analog input lines AN0-AN3 are converted to digital form, with the results stored in new memory locations, as follows.

AN0	1004H-1007H
AN1	100CH-100FH
AN2	1014H-1017H
AN3	101CH-101FH

Finally, the conversion process is completed when the signals on analog input lines AN4-AN7 are converted to digital form for the second time, with the results stored in the following new locations.

AN4	1024H-1027H
AN5	102CH-102FH
AN6	1034H-1037H
AN7	103CH-103FH

Figure 8-5. Memory Map of ADC Programming Example



Initialization

The initialization phase of the conversion process is outlined in figure 8-6 and described below in steps (a) through (e).

(a) The starting address where the conversion results are to be stored is specified in register pair HL. In this example, the starting address is 1000H.

(b) General-purpose registers B, C, D, and E are used as counters to store each A/D converted value into the appropriate memory location. General-purpose register B is also used as an overall counter to test that four results are obtained for each set of four analog inputs (AN0-AN3 and AN4-AN7). Accordingly, register B is set to 03H, and registers C, D, and E are each set to 01H.

(c) The ANM register designates the scan mode and enables AN0-AN3 as the analog inputs as shown in figure 8-7.

(d) A skip instruction is used to clear the INTFAD interrupt request flag prior to clearing the MKAD interrupt mask bit. This ensures that the contents of the conversion registers are filled with newly-converted values before an interrupt is generated.

(e) The MKAD bit in the interrupt mask register is cleared, enabling the INTAD interrupt.

Figure 8-6. Flowchart of Initialization Phase of A/D Conversion Operation

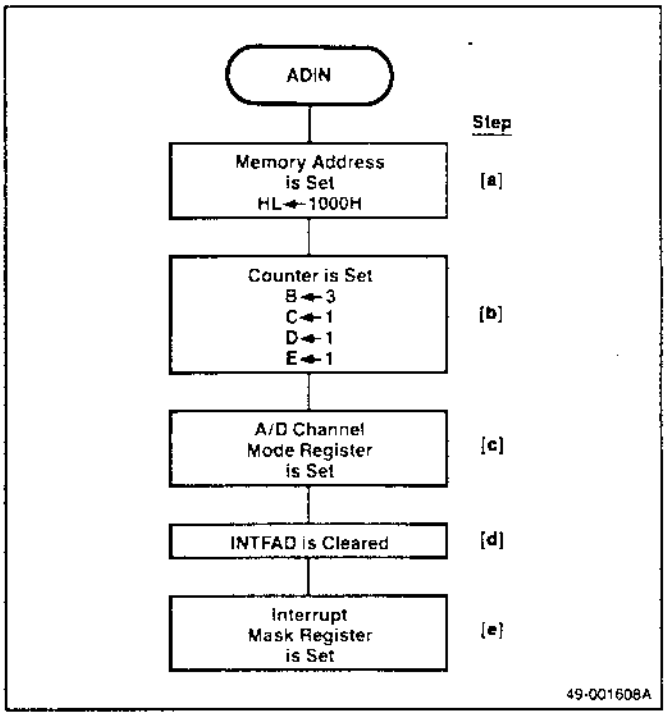


Figure 8-7. Setup of A/D Channel Mode Register for ADC Operation Example

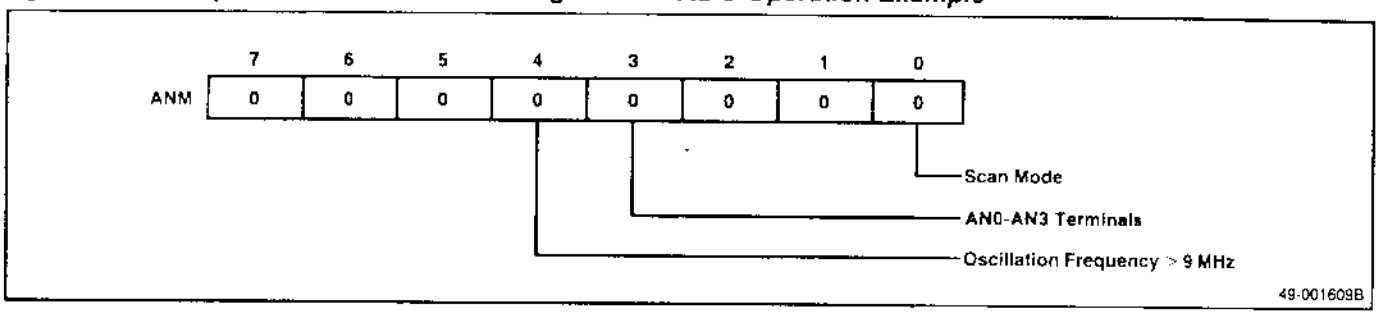


Table 8-2 is an example of program code for each step in the initialization phase of the ADC operation.

Table 8-2. Example of Program Code for Initialization Phase of ADC Operation

*****A/D CONVERTER INITIALIZATION*********				
ADIN:	LXI	H,1000H	;Set data pointer.	Step (a)
	LXI	B,0301H	;Set counters in B and C.	Step (b)
	LXI	D,0101H	;Set counters in D and E.	
	EXX		;Save register set	Step (c)
	MVI	ANM,00H	;for interrupt processing.	
	SKIT	FAD	;Reset INTFAD.	Step (d)
	NOP			
	ANI	MKH,0FEH	;INTAD enable.	Step (e)

Interrupt Processing

In the INTAD interrupt routine of ADC operation, the conversion results are transferred from the conversion result registers to specified locations in memory. Figure 8-8 is a flowchart of the INTAD interrupt processing phase described below in steps (a) through (f).

(a) The contents of CR0-CR3 are transferred to specified memory locations.

(b) Test to determine if four interrupts have been generated for each set of four analog inputs. If not, general-purpose register pair HL is incremented until the fourth interrupt is generated, at which time a jump is made to path (c) of the flow chart. General-purpose register B is used as a counter to tabulate the number of interrupts generated.

When step (b) is completed the first time, memory locations 1000H-1003H, 1008H-100BH, 1010H-1013H, and 1018H-101BH will have been filled with four sets of readings from AN0-AN3.

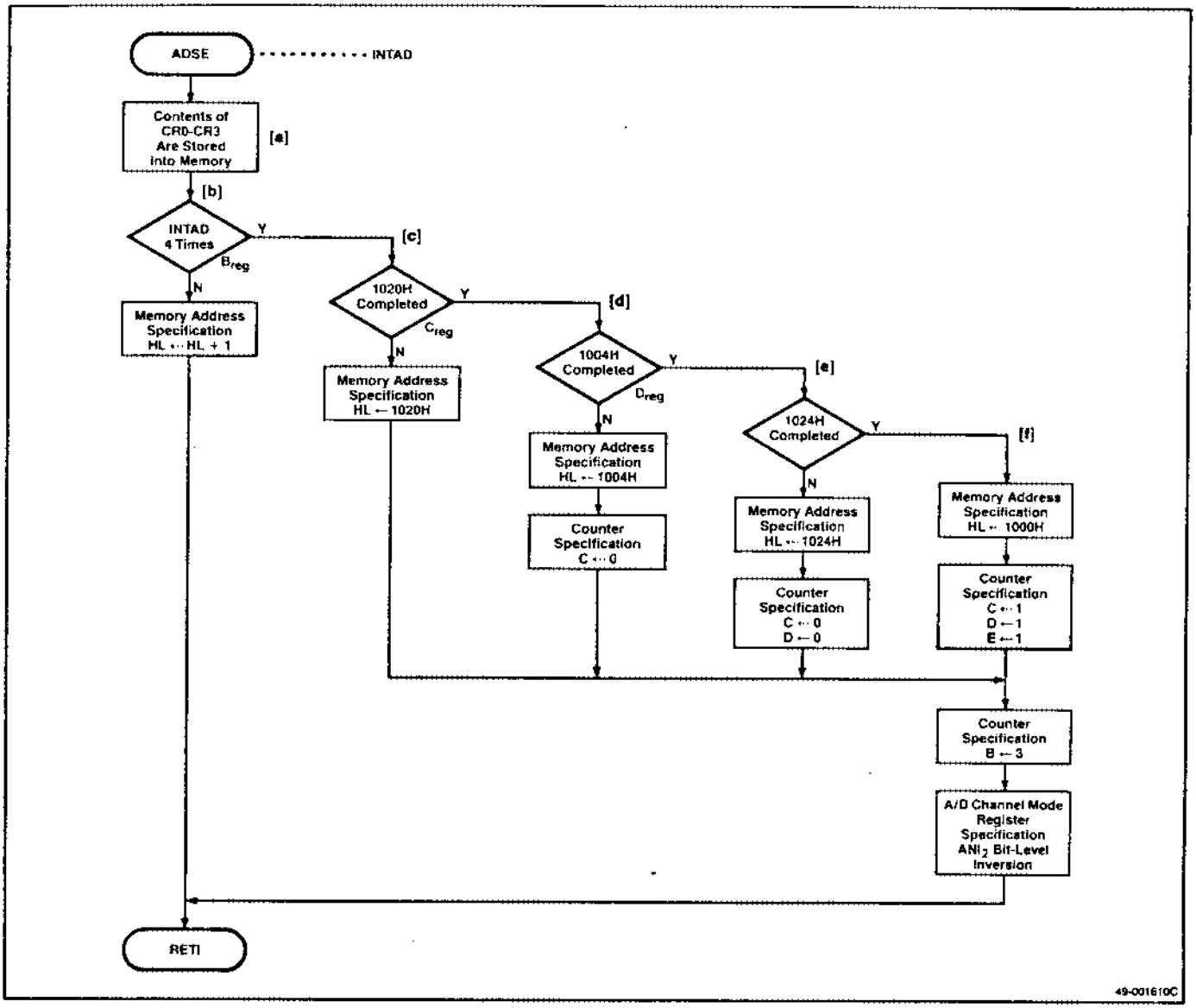
(c) Address 1020H is specified as the start of the next block (for analog inputs AN4-AN7) and general-purpose register B is set to 03H. To switch between one set of analog inputs and the other, the ANI_2 bit in the A/D channel mode register must be complemented. When step (c) is completed, memory locations 1020H-1023H, 1028H-102BH, 1030H-1033H, and 1038H-103BH will have been filled with four sets of readings from AN4-AN7.

After the results from analog input lines AN4-AN7 are stored in the specified memory locations beginning from address 1020H, a jump is made to path (d) of the flowchart. General-purpose register C is used as a counter to test if a result has been obtained for each of the four lines.

(d) The general-purpose register pair HL is used to specify the starting address (1004H) where the second conversion results of analog input lines AN0-AN3 are to be stored. Register B is set to 03H and register C to 00H. The ANI_2 bit in ANM must be inverted to switch from one set of analog input lines to the other. When step (d) is completed, memory locations 1004H-1007H, 100CH-100FH, 1014H-017H, and 101CH-101FH will have been filled with four sets of readings from AN0-AN3.

After the results from these inputs are stored in memory, a jump is made to path (e) of the flow chart. General-purpose register D is used as a counter to test if the A/D values have been stored in memory.

Figure 8-8. Flowchart of INTAD Interrupt Processing Phase of ADC Operation Example



(e) For the second conversion results of AN4-AN7, address 1024H is specified in general-purpose register pair HL as the beginning memory location. General-purpose register B is set to 03H, and general-purpose register D is cleared. A switch from one set of analog input lines to the other is made by inverting the ANI_2 bit in ANM. When step (e) is completed, memory locations 1024H-1027H, 102CH-102FH, 1034H-1037H, and 103CH-103FH will have been filled with four sets of readings from AN0-AN3.

After the results from lines AN4-AN7 have been transferred to the respective mapped memory locations, a jump is made to path (f) of the flowchart. General-purpose register E is used as a counter to test if a result has been obtained for all four lines.

(f) The conversion results are now stored in memory locations 1000H to 103FH.

Table 8-3 is an example of program code for each step in the INTAD interrupt processing routine of the ADC operation. For this example, the JMP ADSE instruction should be located at the INTAD interrupt vector address (0020H).

Table 8-3. INTAD Interrupt Processing Example

```
;*****A/D CONVERTER SERVICE*********
ADSE:   EXA             ;Save accumulator.
        EXX             ;Save register.
        MOV   A,CR0     ;Store A/D conversion data          Step (a)
        STAX  H         ;to memory.
        MOV   A,CR1
        STAX  H+8H      ;Store A/D conversion data
        MOV   A,CR2     ;to memory.
        STAX  H+10H     ;Store A/D conversion data
        MOV   A,CR3     ;to memory.
        STAX  H+18H     ;Store A/D conversion data
                        ;to memory.
        DCR   B         ;Decrement counter; skip           Step (b)
                        ;if borrow.
        JR    ARIN      ;Four not done.
        DCR   C         ;First AN0-AN3 done.               Step (c)
        JR    ARST0
        MOV   A,D       ;First AN4-AN7 done.
        DCR   A                                            Step (d)
        JR    ARST1
        MOV   A,E       ;Second AN0-AN3 done.
        DCR   A                                            Step (e)
        JR    ARST2
        LXI   H,1000H   ;Second AN4-AN7 done; set          Step (f)
                        ;data pointer.
        LXI   D,0101H   ;Set counter.
        MVI   C,01H     ;Set counter.
        JR    RET1
ARIN:   INX   H         ;Increment HL.                     Step (b)
        JR    RET2
ARST0:  LXI   H,1020H   ;Set data pointer.                 Step (c)
        JR    RET1
ARST1:  LXI   H,1004H   ;Set data pointer.                 Step (a)
        MOV   D,A
        JR    RET0
ARST2:  LXI   H,1024H   ;Set data pointer.                 Step (e)
        MOV   E,A
        MVI   D,00H
RET0:   MVI   C,00H
RET1:   MVI   B,03H
        XRI   ANM,04H   ;Invert ANI2 bit.
RET2:   EXA             ;Recover accumulator.
        EXX             ;Recover register.
        EI              ;Enable interrupt.
        RETI            ;Return.
```

INTERRUPT SOURCES

The interrupt structure includes three external interrupt sources ($\overline{NMI}$, INT1, $\overline{INT2}$), eight internal interrupt sources (INTT0, INTT1, INTE0, INTE1, INTEIN, INTAD, INTSR, and INTST), and one software interrupt (SOFTI). Interrupts are organized into seven priority levels, with an interrupt vector location assigned to each priority level. The priority organization is shown in table 9-1.

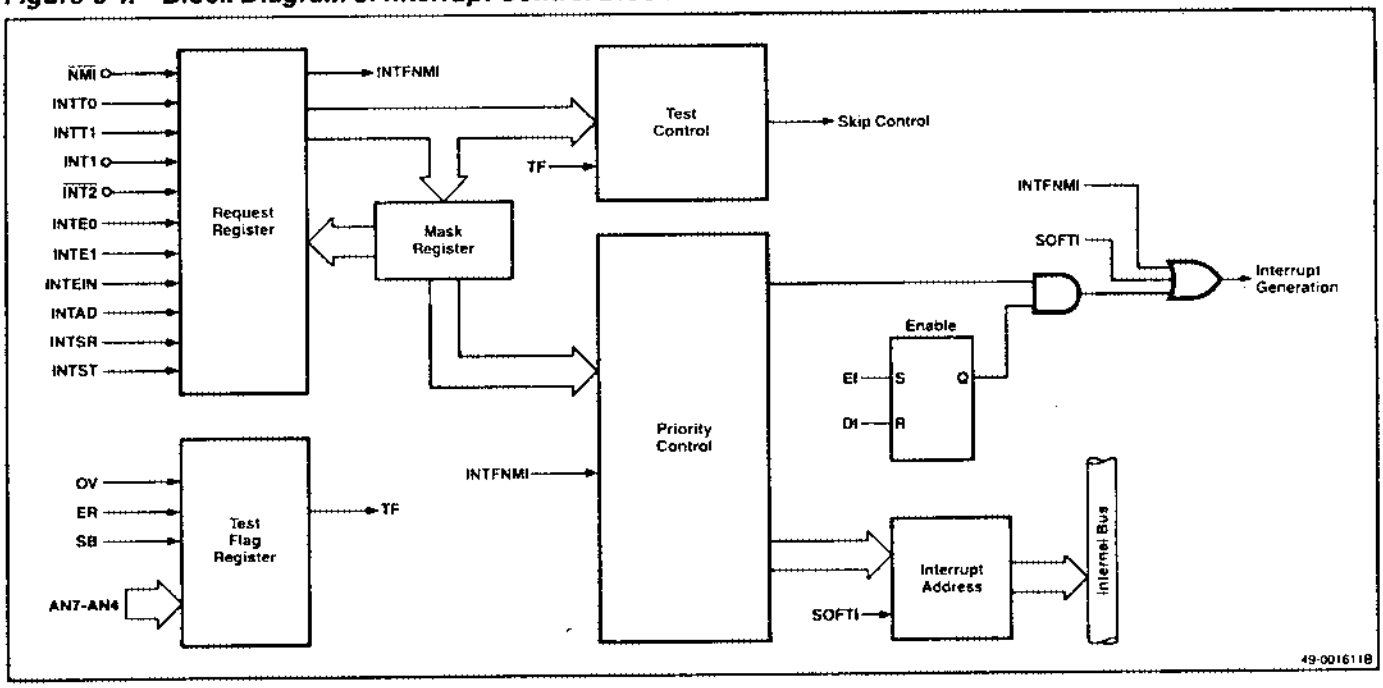
FUNCTIONAL ELEMENTS

The functional elements of the interrupt controller consist of the interrupt request register, interrupt mask register, priority control circuit, interrupt test control circuit, interrupt enable flip-flop, and interrupt test flag register. The block diagram is shown in figure 9-1.

Table 9-1. Interrupt Priority and Vector Locations

Priority	Internal/ External	Interrupt Source	Interrupt Condition	Interrupt Address Decimal	Interrupt Address Hexadecimal
1	External	$\overline{NMI}$	Falling edge	4	0004
2	Internal	INTT0	Interval counter TM0 = upcounter0	8	0008
		INTT1	Interval counter TM1 = upcounter1		
3	External	INT1	Rising edge	16	0010
		$\overline{INT2}$	Falling edge		
4	Internal	INTE0	When upcounter = ETM0 in the timer/event counter	24	0018
		INTE1	When upcounter = ETM1 in the timer/event counter		
5	Internal	INTEIN	In the timer/event counter, if the upcounter clock is (a) ϕ12: interrupt on falling edge of CI (b) CI: interrupt on falling edge of TO	32	0020
		INTAD	A/D converter registers CR0-CR3 have new values		
6	Internal	INTSR	Serial receive buffer full	40	0028
		INTST	Serial transmit buffer empty		
7	Internal	SOFTI instruction		96	0060

Figure 9-1. Block Diagram of Interrupt Control Block



Interrupt Request Register

This register contains the interrupt request flags that indicate which interrupt requests have been generated. The function of each flag is given in table 9-2.

Table 9-2. Interrupt Request Register

Flag	Function
INTFNMI	This flag is set at the falling edge of the $\overline{NMI}$ input.
INTFT0	Indicates a match between TIM0 and its upcounter has occurred.
INTFT1	Indicates a match between TIM1 and its upcounter has occurred.
INTF1	This flag is set at the rising edge of INT1.
INTF2	This flag is set at the falling edge of $\overline{INT2}$.
INTFE0	Indicates a match between ETM0 and its upcounter in the timer/event counter has occurred.
INTFE1	Indicates a match between ETM1 and its upcounter in the timer/event counter has occurred.
INTFEIN	Indicates a falling edge of CI input or falling edge of TO output has occurred.
INTFAD	Indicates that the last of four conversion results for registers CR0-CR3 is complete.
INTFSR	Indicates the receive buffer is full.
INTFST	Indicates the transmit buffer is empty.

Interrupt Mask Register

These two 8-bit registers (MKH and MKL) contain the mask bits corresponding to each maskable interrupt; $\overline{NMI}$ and SOFTI cannot be masked. When set, a bit masks the respective interrupt. A maskable interrupt is unmasked if the corresponding mask bit is cleared. All bits are set by a $\overline{RESET}$ input. Therefore, all maskable interrupts are masked after a $\overline{RESET}$. Figure 9-2 identifies the bits of the interrupt mask registers.

Priority Control Circuit

This circuit controls the order in which interrupts are to be accepted if more than one unmasked interrupt has been requested at the same time. Interrupts with the higher priority level are accepted and processed before those with a lower priority level (see table 9-1).

Interrupt Test Control Circuit

This circuit enables the execution of a skip instruction (SKIT or SKNIT) to test the status of the interrupt request flags and interrupt test flags.

Interrupt Enable Flip-Flop

This flip-flop determines whether or not all maskable interrupts are enabled. When this flip-flop is set by the EI instruction, all unmasked interrupts are enabled. When cleared by a $\overline{RESET}$ input or the DI instruction, all interrupts are disabled. When an interrupt request is accepted, the flip-flop is reset, inhibiting the acknowledgment of any interrupts while the current interrupt is being processed. If an EI instruction is executed in an interrupt service routine, the flip-flop is again set, and the highest unmasked interrupt request will be accepted.

Interrupt Test Flag Register

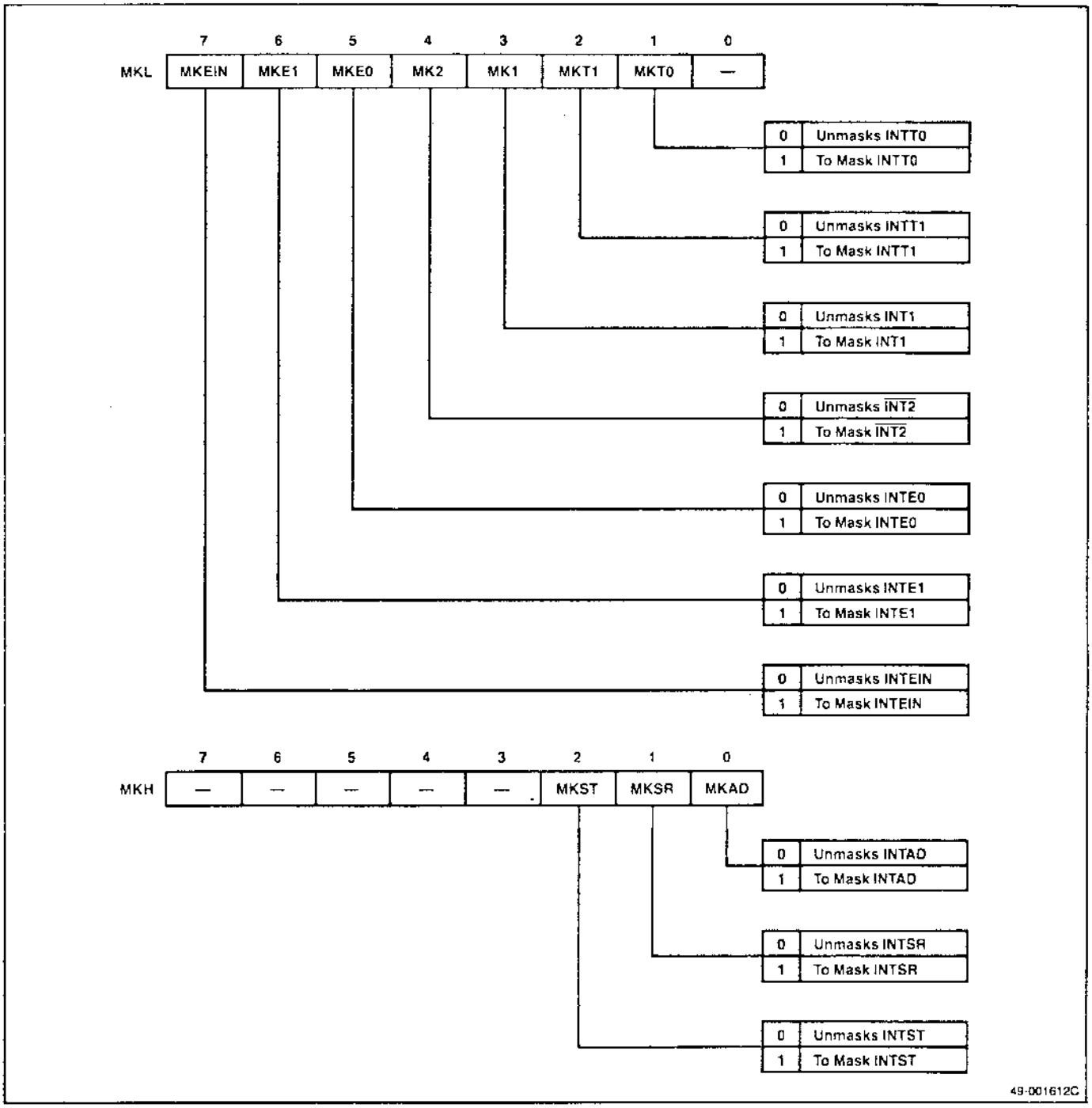
This register contains seven test flags that indicate various error conditions or special conditions. The function of each flag is shown in table 9-3.

Table 9-3. Interrupt Test Flag Register

Flag	Function
OV	The overflow flag is set when ECNT (upcounter in the timer/event counter) overflows from FFFFH to 0000H.
ER	The error flag is set when a parity error, framing error, or overrun error is detected in the reception of serial data.
SB	The standby flag is set by a rising V_{DD} input.
AN4-AN7	Each of these flags is set whenever a falling edge is sensed on the corresponding input line if that line is used as an edge-sensing input.

The status of each flag in the interrupt test flag register can be tested using a SKIT or SKNIT instruction.

Figure 9-2. Interrupt Mask Register



EXTERNAL INTERRUPT SAMPLING

To prevent improper operation due to noise interference, the active/inactive levels of the $\overline{NMI}$, INT1, and $\overline{INT2}$ signals are sampled by the ϕ12 clock (1 μs at 12 MHz). These signals must remain at a low or high level long enough to be sampled by three ϕ12 clock pulses in order to be considered valid.

As illustrated in figure 9-3, signals with a minimum duration of three ϕ12 clock periods (3 μs at 12 MHz) will be recognized as interrupt requests and signals with a duration of less than two ϕ12 clock periods (2 μs at 12 MHz) will be treated as noise and ignored. Signals between two and three clock periods long may or may not be recognized depending on where they occur with respect to the ϕ12 clock.

In the CMOS parts, the $\overline{NMI}$ input is protected from noise by an analog circuit that requires the high or low level to be stable for 10 μs. INT1 and $\overline{INT2}$ function the same way in the CMOS parts as in the NMOS parts.

$\overline{NMI}$ Input

This nonmaskable interrupt signal is falling-edge triggered. If the sampling pulse detects that the $\overline{NMI}$ signal falls from high level to low level, and the $\overline{NMI}$ signal is held at a low level for a minimum of three ϕ12 clock periods, then the INTFNMI flag in the interrupt request register is set. The CMOS parts require 10 μs.

The interrupt request register is checked at the end of each instruction, and if a nonmaskable interrupt has been generated, a call is made to the vector location of the service routine for the nonmaskable interrupt, regardless of the status of the interrupt enable flip-flop or interrupt mask register. The INTFNMI interrupt request flag is reset automatically when a jump is made to the vector location. The $\overline{NMI}$ input may be sampled by testing the INTFNMI flag with a SKIT or SKNIT instruction. The $\overline{NMI}$ input should be pulled up with an external resistor to prevent false interrupts. The recommended pull-up resistor value is 10 kΩ.

INT1 Input

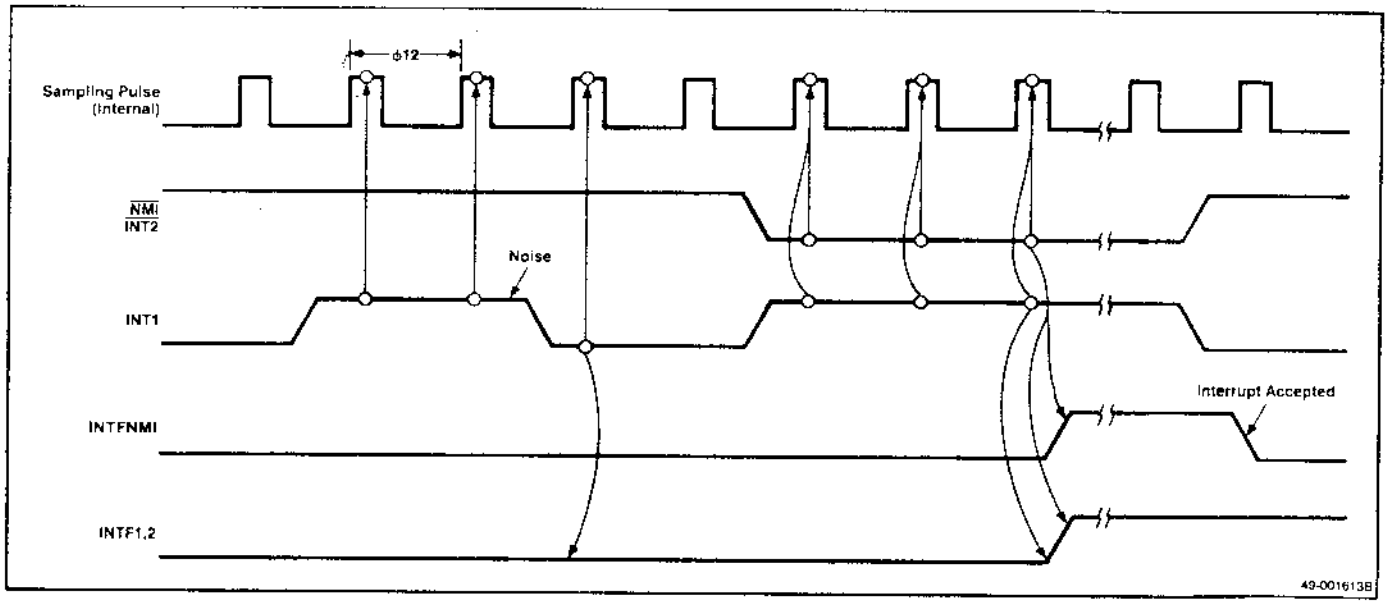
This is a rising-edge triggered maskable interrupt with a priority level of 3. If the INT1 signal rises from low level to high level, and the INT1 signal is held at a high level for a minimum of three ϕ12 clock periods, then the INTF1 interrupt request flag in the interrupt request register is set.

If, at the end of an instruction, a check of the interrupt request register indicates that the INTF1 flag is set, INT1 is unmasked, and no higher priority unmasked interrupts are present, a jump is made to vector location 0010H.

$\overline{INT2}$ Input

This interrupt functions the same as the INT1 interrupt, except that the interrupt occurs on the falling edge of the input.

Figure 9-3. Timing for Interrupt Sampling Pulses



SOFTI INSTRUCTION INTERRUPTS

The generation of a program-controlled, nonmasked interrupt is provided by the SOFTI instruction, which causes a call to vector location 0060H independent of the status of the interrupt enable flip-flop or interrupt mask register. Unlike other interrupts, execution of a SOFTI interrupt does not disable other interrupts.

If the SOFTI instruction is encountered and no interrupts are present, the SOFTI interrupt will be executed independent of DI or EI. Once the SOFTI instruction has been executed, NMI can interrupt. If interrupts are enabled, any other active unmasked interrupt will be accepted.

SOFTI is the lowest priority interrupt of all the interrupts. When the SOFTI instruction is fetched and decoded, the processor waits until the next M1T1 time and decides at that time what to do. If no other interrupt request flags are set, SOFTI starts. If any other interrupt flag is set, that interrupt will be serviced before SOFTI. When no interrupt request flags are set, SOFTI will be executed.

The SOFTI instruction is executed even if preceded by an instruction producing a skip condition (i.e. an arithmetic or logical operation, an autoincrement or autodecrement, a shift, a skip, or a return-from-subroutine instruction). When the SOFTI instruction is executed, the SK flag in the PSW is set and its value retained in the stack area. When the program returns from the SOFTI interrupt processing routine, the SK flag previously set in the PSW is restored, and the program skips the instruction following the SOFTI instruction.

INTERRUPT MASKING

Except for the $\overline{\text{NMI}}$ and the SOFTI interrupts, each priority level and corresponding vector location is shared by two different maskable interrupt sources, which may be independently or jointly enabled. The enabling/disabling of maskable interrupts is done through the use of mask bits in the interrupt mask register. Each mask bit in the interrupt mask register corresponds to a maskable interrupt request flag in the interrupt request register.

Enabling One Interrupt Source per Priority Level

To enable a single interrupt at a specific priority level, the mask bit corresponding to that interrupt is cleared and the mask bit for the other interrupt at that priority level is set. When the corresponding interrupt request flags in the interrupt request register are set, indicating that interrupt requests for those sources have been generated, the interrupt whose mask bit is set will be ignored, and the one whose mask bit is cleared will be accepted—provided no interrupts of a higher priority are pending and maskable interrupts have been enabled by the interrupt enable flip-flop.

The interrupt request flag for the masked interrupt will remain set until tested by a skip instruction. The interrupt request flag for the nonmasked interrupt will be automatically cleared by the hardware jump to the interrupt address.

Enabling Both Interrupts at the Same Priority Level

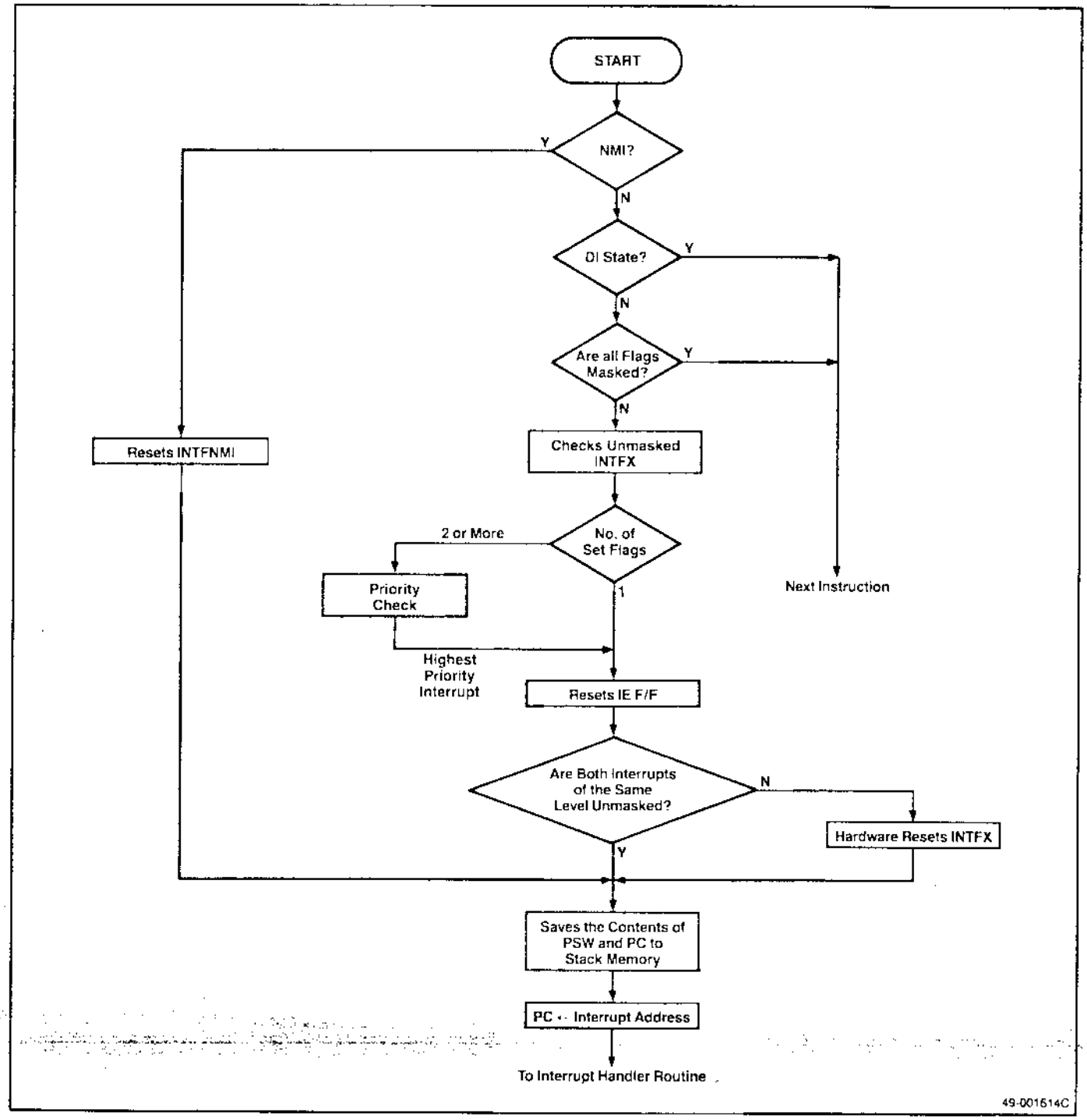
If the mask bits of both interrupt sources at the same priority level are cleared, then both interrupts will be enabled. Either interrupt can then be accepted if the corresponding interrupt request flag is set. The user must determine which interrupt has occurred by testing the interrupt flags with the SKIT or SKNIT instructions. If both interrupt request flags are set at the same time, the sequence of interrupt processing is determined by a skip instruction at the beginning of the interrupt service routine. When both interrupts at a given priority level are enabled, the interrupt request flags for those interrupts are not reset by hardware, but by the execution of the skip instruction.

SUMMARY OF OPERATION

For external interrupts, the ϕ12 clock sampling pulses determine if an interrupt request signal is legitimate or spurious noise. If legitimate, the corresponding interrupt request flag is set. For internal interrupts, the interrupt request flag is immediately set upon generation of the interrupt request. Once an interrupt request flag has been set, interrupt processing is identical for both external and internal interrupts. Figure 9-4 shows the following steps.

(1) The interrupt request flags are checked at the last timing cycle of every instruction. The flag for any interrupt that has a mask bit set is not checked.

Figure 9-4. Interrupt Operating Procedure



(2) When more than one interrupt request flag is found to be set, the priority level of each interrupt is checked. The interrupt with the highest priority is serviced first, and the interrupt request flags of the remaining interrupts will stay set until each is serviced in order of priority.

(3) When an interrupt is accepted, the interrupt enable flip-flop is reset, disabling maskable interrupts (NMI and SOFTI cannot be masked) during processing of the current interrupt. Any pending unmasked interrupts (interrupts whose corresponding interrupt request flags have remained set from the previous step) are serviced in order of priority. This process is repeated until all pending interrupts have been serviced.

(4) Whenever an interrupt is accepted, the corresponding interrupt request flag is automatically reset—except if both interrupt sources at the same priority level are enabled by the corresponding mask bits. In this case, the execution of a skip instruction (SKIT or SKNIT) resets each flag.

(5) When an interrupt is accepted, the contents of the PSW are saved in the stack. Next, the higher-order byte of the program counter is saved, followed by the lower-order byte of the program counter.

(6) The program then executes a call to the vector location for the interrupt accepted and executes the interrupt service routine. It takes 16 timing states (250 ns/state at 12 MHz) to complete these steps and begin the interrupt service routine. At the end of the interrupt service routine, an RETI instruction is executed. This instruction returns the program to the address following the last executed instruction prior to the interrupt. The saved contents of the PSW and program counter are loaded from the stack—restoring the values contained in the PSW and program counter at the point where the interrupt was accepted—allowing the program to resume operation from that point.

It takes 16 timing states (250 ns/state at 12 MHz) to go from the beginning of step (2) to the first interrupt instruction, which is located at the interrupt vector location.

Figure 9-5 depicts the sequence when only one interrupt source at a given priority level is enabled by the corresponding mask bit. Figure 9-6 depicts the sequence when both interrupt sources at the same priority level are enabled by the corresponding mask bits.

Figure 9-5. Flowchart 1: Interrupt Processing Sequence

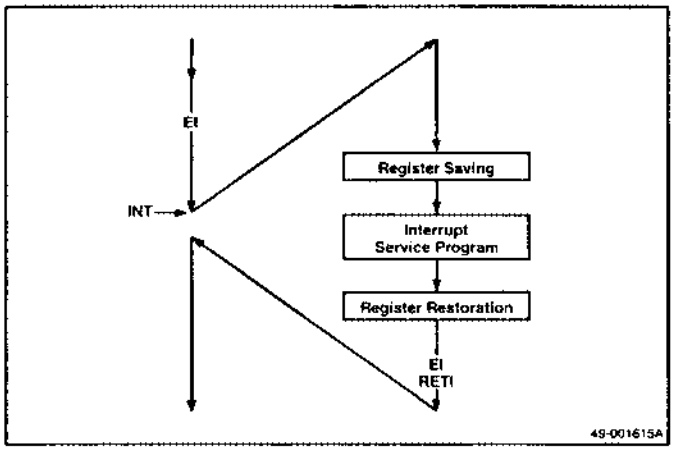
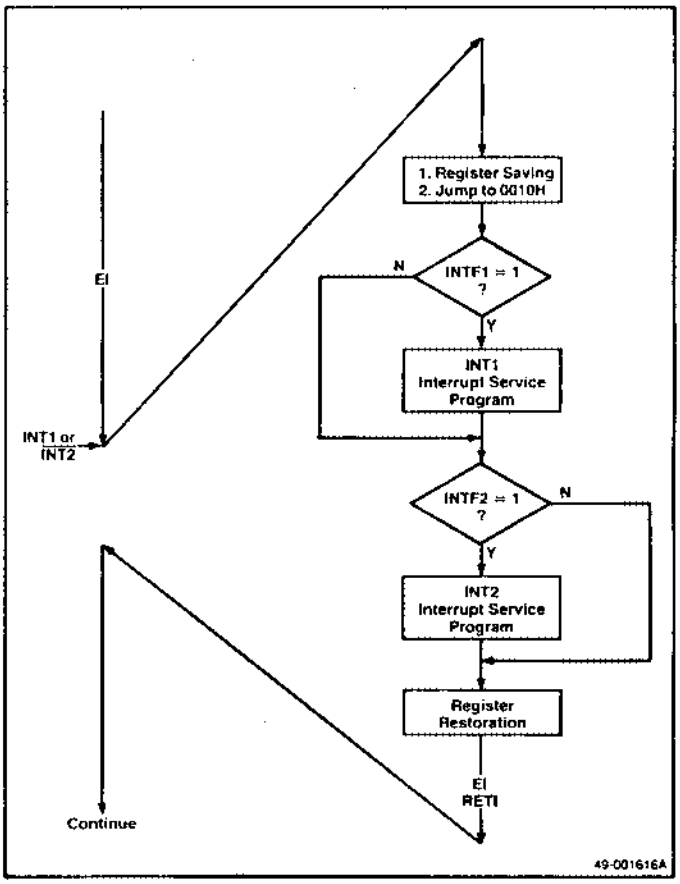


Figure 9-6. Flowchart 2: Interrupt Processing Sequence



INTERRUPT WAIT TIMES

Three factors determine external interrupt wait times. These factors and the corresponding number of timing states (one timing state = 250 ns at 12 MHz) for each are outlined in table 9-4. The sum of the timing states for these factors represents the wait time from the point at which an external interrupt request is asynchronously generated by the CPU to the point at which the first instruction in the interrupt service routine is executed. This wait time differs according to the type of instruction being executed at the time the interrupt is generated and the timing inherent in the interrupt generated.

Table 9-4 shows maximum wait times. For the first factor, the time required to set the interrupt request flag after an interrupt request has been generated is 14 timing states. This includes 12 timing states to ensure a legitimate external interrupt signal and 2 timing states to set the interrupt request flag.

A worst case is assumed for the second factor, allowing 59 timing states for a divide instruction, which requires the longest time of all instructions to complete. The interrupt request flags are checked at the end of every instruction (requiring two timing states) and the second factor time depends on the type of instruction being executed. This value can range from 4 timing states (minimum instruction time of 2 plus 2 to check the flags) to 59 timing states (maximum instruction time of 57 plus 2 to check the flags).

The third factor requires 16 timing states to allow for the time required to save the contents of the PSW and the program counter on the stack.

Table 9-4. Maximum External Interrupt Wait Time

Factor	Description	Number of States
1	Eliminate noise signals	14
2	Execute the instruction (division instruction)	59
3	Automatic saving	16
	TOTAL (One state = 250 ns at 12 MHz)	89 (22.25 μs)

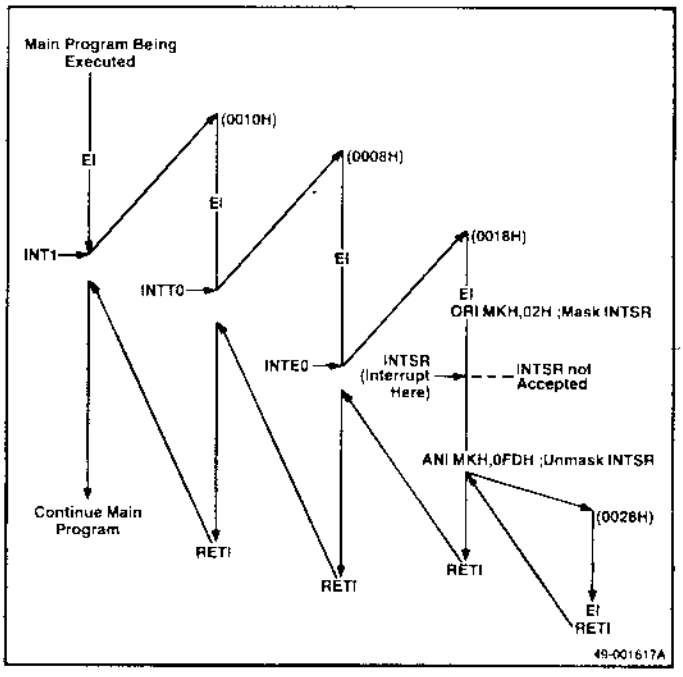
MULTILEVEL NESTING

All internal and external interrupts are recognized when an EI instruction is executed. During an interrupt subroutine, if the EI instruction is executed and the interrupt request flag of the current interrupt is reset, the highest priority interrupt of the pending interrupts will be serviced. The new interrupt could be of lower priority than the one being serviced. This permits the nesting of many interrupts, which are serviced in priority order with the lower-priority interrupts kept pending until all higher-priority interrupts have been serviced.

Note: If both interrupt sources for the same priority level are enabled, the interrupt source must be determined by the interrupt service routine.

Such interrupt nesting allows prioritized sequencing of interrupt-controlled subroutines. The number of subroutine levels can be extended all the way out to the full capacity of memory, allowing virtually unlimited subroutine nesting. An example of an interrupt sequence involving three levels of interrupts is shown in figure 9-7.

Figure 9-7. Flowchart of Processing Sequence for Three Levels of Interrupts



OPERATION, ALL PARTS

Reset Operation

If a low-level signal is input on the $\overline{\text{RESET}}$ line and held low for a minimum of 20 timing states (5 μs at 12 MHz in the 7810, 7810H, 7811, and 7811H) or for a minimum of 10 μs in the 78C10, 78C11, and 78C14, the microcomputer is initialized and the following operations are performed.

(1) Interrupt enable flip-flop is cleared, disabling the acknowledgement of maskable interrupts.

(2) Interrupt mask register is set, disabling all maskable interrupt sources; interrupt request flags are cleared and all pending interrupts are reset.

(3) PSW register is cleared.

(4) Program counter is cleared to 0000H.

(5) In the 7811, 7811H, 78C11, and 78C14 configuration, the mode A, mode B, mode C, and mode F registers are set to FFH. Bits 0-2 in the memory mapping register and mode control C register are cleared, initializing ports A, B, C, D, and F as 8-bit input ports (all lines in high-impedance state).

(6) In the 7810, 7810H, and 78C10, port D is initialized as 8-bit multiplexed address/data lines and port F is initialized as address outputs or port inputs as specified by the MODE0 and MODE1 pins. The remaining port A, B, and C lines are initialized as input ports.

(7) All test flags except the SB flag are cleared. The SB flag and RAE bit are unchanged.

(8) Timer mode register (TMM) is initialized to FFH and the timer flip-flop is reset.

(9) Timer/event counter mode registers (ETMM and EOM) are cleared.

(10) Serial mode high-byte register (SMH) is cleared and serial mode low-byte register (SML) is initialized to 48H.

(11) A/D channel mode register (ANM) is cleared.

(12) $\overline{\text{WR}}$ and $\overline{\text{RD}}$ signals go high in the NMOS parts. In the CMOS parts, $\overline{\text{WR}}$, $\overline{\text{RD}}$, and ALE become high impedance.

(13) For the 78C10, 78C11, and 78C14 only, bits ZC_1 and ZC_2 of the ZCM register are set to 1.

The contents of the following are undefined.

(1) Stack pointer (SP).

(2) Accumulators A and EA and alternate accumulators A' and EA'.

(3) All general-purpose registers (V, B, C, D, E, H, L, V', B', C', D', E', H', and L'.

(4) Output latches of all port lines.

(5) Timer 0 and timer 1 registers (TM0 and TM1).

(6) Timer/event counter registers (ETM0 and ETM1).

(7) RAE bit in the memory mapping register.

(8) Data memory.

After the $\overline{\text{RESET}}$ signal is released, program execution begins from address 0000H.

Halt Operation

When a HLT instruction is executed, the processor enters halt mode. The CPU stops operation and idles in the M3T2 state. A reset or an interrupt will cause the microcomputer to exit this mode. All on-chip peripherals will continue to operate normally during the halt mode.

In the halt mode, the CPU clock stops and program execution halts. However, the contents of all registers and internal RAM are maintained. The timers, timer/event counter, serial interface, A/D converter, and interrupt control circuitry operate normally. The status of the output pins is shown in table 10-1.

Table 10-1. Status of Output Pins

Pin	Microcomputer	External Expansion
PA_7-PA_0	Data holding	Data holding
PB_7-PB_0	Data holding	Data holding
PC_7-PC_0	Data holding	Data holding
PD_7-PD_0	Data holding	High impedance
PF_7-PF_0	Data holding	Next address holding (pins that output addresses) Data holding (pins that output port data)
$\overline{\text{WR}}$,$\overline{\text{RD}}$	High level	High level
ALE	High level	High level

When a HLT instruction is executed, halt mode is entered except if an interrupt flag for an unmasked interrupt is set. The halt mode is released by any unmasked interrupt or by a hardware reset. Since halt mode is released by an interrupt request, the CPU will not enter halt mode if an unmasked interrupt is pending. To enter halt mode in an area of program execution where an interrupt may be pending, you may:

(1) Process the pending interrupt.

(2) Reset the interrupt request flag via a skip instruction.

(3) Mask all interrupts that are not necessary for releasing the halt mode.

If halt mode is released by $\overline{\text{RESET}}$, the contents of RAM are retained, but the contents of the registers are undefined. The halt mode is released when $\overline{\text{RESET}}$ goes low. When $\overline{\text{RESET}}$ returns to a high level, program execution begins at address 0000H (see figure 10-1).

Halt Mode Release by Interrupt Requests

When halt mode is released by an $\overline{\text{NMI}}$ interrupt request, the program jumps to the $\overline{\text{NMI}}$ interrupt address (0004H). It does not go to the next location after the HLT instruction.

When halt mode is released by an unmasked interrupt, the operation following release depends on whether interrupts are enabled or disabled. If interrupts are enabled (EI flip-flop = 1), the program then jumps to the respective interrupt address (figure 10-2). If interrupts are disabled (EI flip-flop = 0), execution starts with the instruction following the HLT instruction (see figure 10-3). Note that the interrupt request flag (INTFX) remains set and must be tested with a skip instruction to be cleared.

Figure 10-1. Halt Mode Release Timing by $\overline{\text{RESET}}$

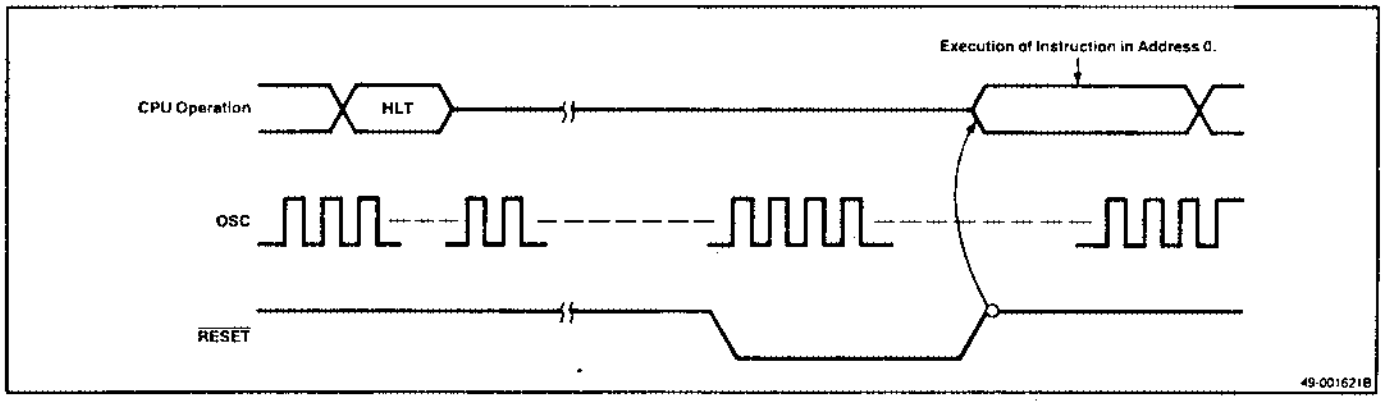
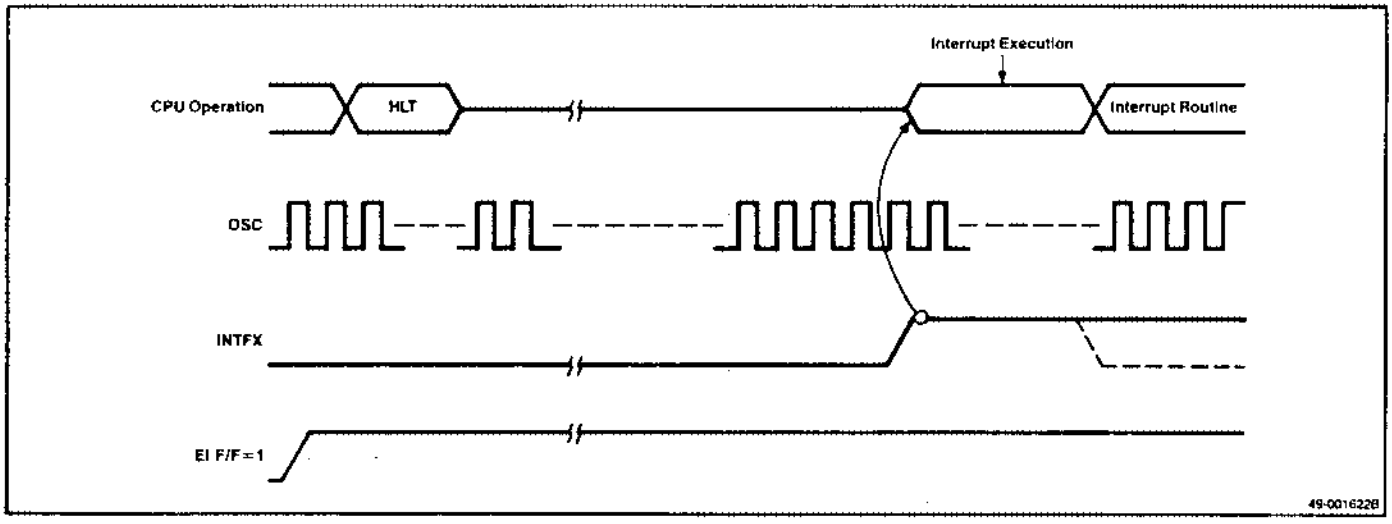


Figure 10-2. Halt Mode Release Timing by Interrupt With Interrupts Enabled



OPERATION, 7810/10H/11/11H

Parts 7810, 7810H, 7811, and 7811H contain a normal, standby, and halt mode as shown in figure 10-4. They do not contain a stop mode.

A special circuit is provided that allows backup power to be supplied to maintain 32 bytes of internal RAM (addresses 65,504-65,535) as well as the value of the RAE bit in the memory mapping register and the value of the SB (standby) flag in the event of loss of main power. The +5-volt main power supply is input to the V_{CC} line and the +5-volt backup power supply is input to the V_{DD} line. In normal operation, the single +5-volt main power supply input on the V_{CC} line is used.

Standby mode is initiated in order to use the backup power supply input on the V_{DD} line to power the RAE bit, SB flag, and the 32 bytes of internal RAM.

In standby mode, the RAE bit is cleared, preventing access of the internal RAM by the CPU, thus protecting the contents of RAM from being affected by power instability on the V_{CC} line. Also, $\overline{RESET}$ goes low when exiting standby mode, and program operation begins from address 0000H as in a normal reset operation; the SB flag indicates that recovery is being made from a standby condition.

Figure 10-3. Halt Mode Release Timing by Interrupt With Interrupts Disabled

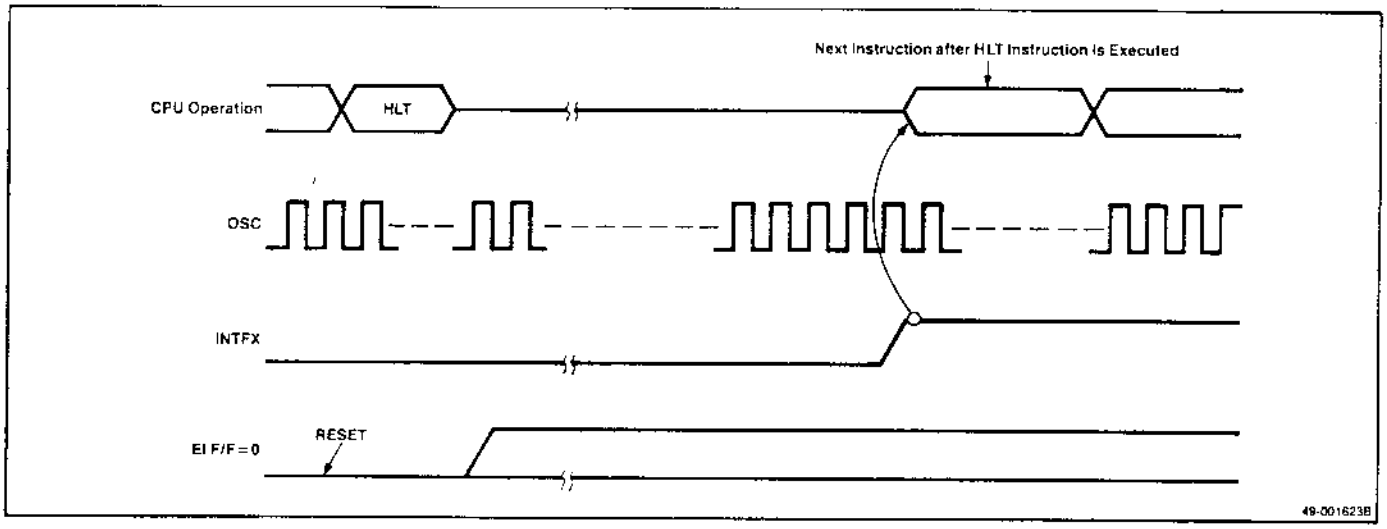
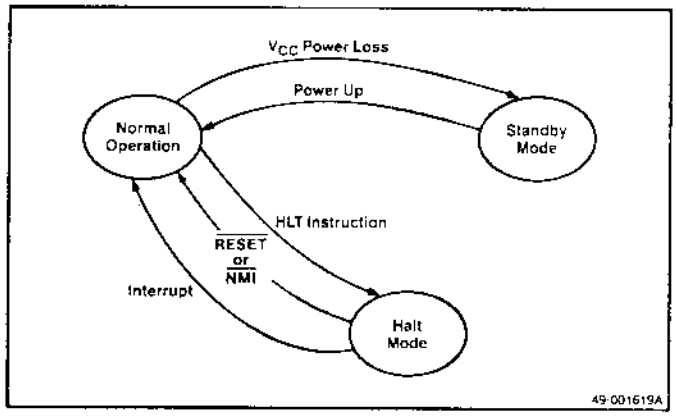


Figure 10-4. Standby Mode for 7810, 7810H, 7811, and 7811H



The SB flag is set at the rising edge of V_{DD} at power-up, and can be reset by a SKIT or SKNIT skip instruction. Recovery from a standby condition (as opposed to a power-up) can therefore be determined by testing the SB flag using a skip instruction. If the flag is set, a normal power-up has occurred, and if cleared, a standby recovery has occurred.

A standby operation consists of the following sequence of events (figure 10-5).

(1) A drop in V_{CC} voltage is detected by user-supplied circuitry, generating a nonmaskable interrupt ($\overline{NMI}$).

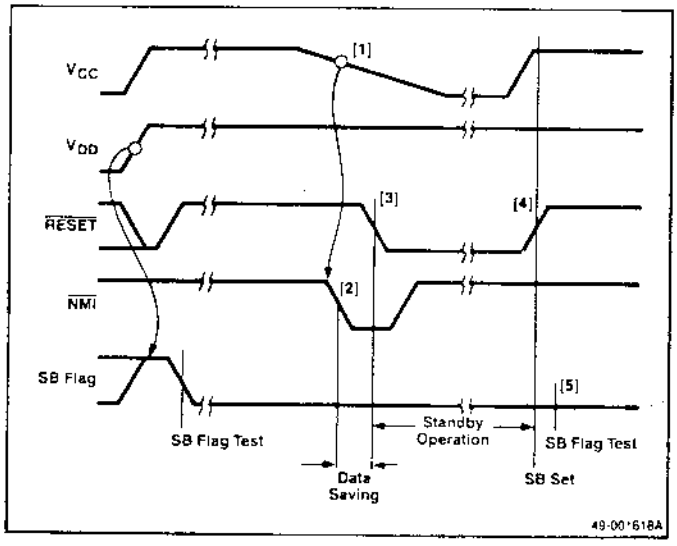
(2) Data to be maintained during power-down should be saved by the interrupt service routine in the protected 32-byte area of internal RAM prior to V_{CC} dropping below the specified operating level. The program should clear the RAE bit in the memory mapping register.

(3) $\overline{RESET}$ is asserted. The contents of the protected 32-byte internal RAM area and the value of the RAE bit and SB flag are maintained by the backup power supply even if V_{CC} voltage drops below the required operating level.

(4) When the voltage on V_{CC} returns to normal and the oscillator stabilizes, $\overline{RESET}$ is released and program operation begins from address 0000H.

(5) The SB flag is tested to determine if a power-up or a standby recovery has occurred. If returning from standby, SB = 0; if starting from power-up, SB = 1. For standby recovery, the RAE bit must be set to read the saved data.

Figure 10-5. Timing Sequence for Standby Operation



OPERATION, 78C10/C11/C14

Parts 78C10, 78C11, and 78C14 contain normal, halt, and stop modes, which interrelate as shown in figure 10-6. The standby mode consists of the halt, software stop, and hardware stop modes.

RAM Retention in Stop Mode

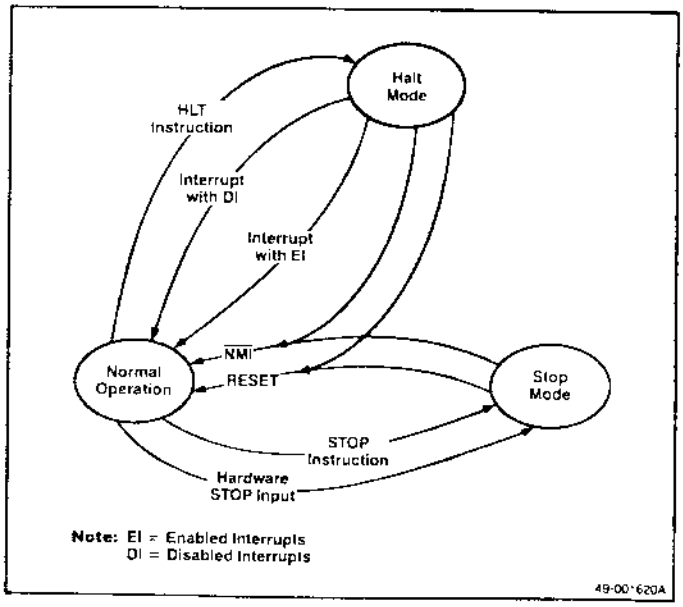
Internal RAM data will be retained during software and hardware stop modes as long as V_{DD} is maintained at or above 2.5 volts. Lowering V_{DD} to 2.5 volts reduces power consumption in the stop mode. V_{DD} can be lowered only after the stop mode is entered, and must return to operational level before the stop mode is released.

Software Stop Mode

When the STOP instruction is executed, the CPU enters the stop mode and all clocks are stopped. Program execution stops and the contents of internal RAM are saved, the timer upcounter is cleared, and all on-chip peripherals stop; only $\overline{NMI}$ and $\overline{RESET}$ circuitry remain active. The status of the output pins is the same as for the halt mode (see table 10-1).

Internal interrupts should be masked before execution of the STOP instruction. Otherwise, during the stop mode release, the oscillation stabilization period following the stop mode may cause an erroneous internal interrupt operation. The oscillator stabilization time is obtained from the crystal manufacturer's specification.

Figure 10-6. Standby Modes for 78C10, 78C11, and 78C14



A $\overline{\text{RESET}}$ input or an $\overline{\text{NMI}}$ request can release the software stop mode. Asserting $\overline{\text{RESET}}$ low releases the processor from the stop state, puts it in the reset state, and starts the clock oscillator. The circuit must hold $\overline{\text{RESET}}$ low (see manufacturer's specification for the time) by the user's circuit to allow the oscillator to stabilize (see figure 10-7). When $\overline{\text{RESET}}$ releases the software stop mode, the contents of the internal RAM are maintained and the contents of the registers are undefined.

When $\overline{\text{RESET}}$ goes high, program execution begins at 0000H, just as for a power-on reset. Since both $\overline{\text{RESET}}$ and power-up cause the CPU to start program execution at location 0000H, the standby flag (SB) can be used to distinguish between a power-up or $\overline{\text{RESET}}$ release of stop mode. When V_{DD} crosses the rated voltage going from a low to high level, the SB flag is set. Execution of a skip instruction clears the flag. By testing the SB flag at the start of the reset routine, the user can distinguish whether a $\overline{\text{RESET}}$ or power-up caused the start.

When an $\overline{\text{NMI}}$ interrupt request releases software stop mode, TIMER 1 is used to start CPU operation. Because of this feature, oscillator stabilization time can be achieved without adding external hardware. To use this feature, the timer must be programmed prior to execution of the STOP instruction. Using the stabilization time required by the oscillator, program the number of counts equivalent to this time into the timer register TM1, and then set the timer mode register (TMM) for timer mode. TIMER1 can generate a delay of up to 65 ms with a 12-MHz crystal.

The timer upcounter is cleared on entering the software stop mode. When the stop mode is released by an $\overline{\text{NMI}}$ input, the timer begins counting up, according to the mode specified prior to entering the stop mode. When the contents of the upcounter match the contents of TM1, program execution starts as shown in figure 10-8.

Figure 10-7. Software Stop Mode Release Timing Using $\overline{\text{RESET}}$

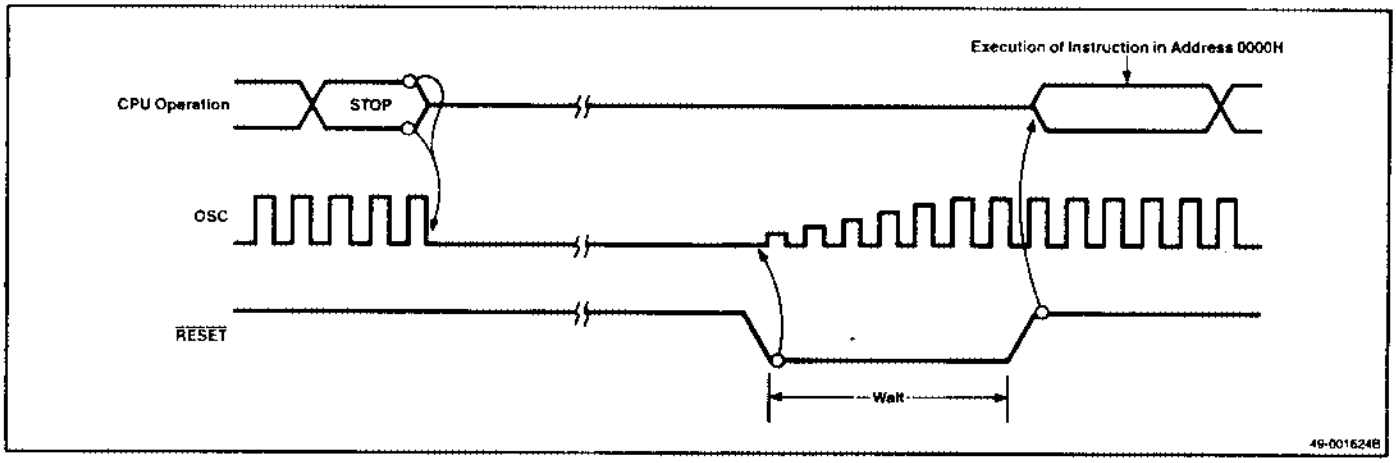
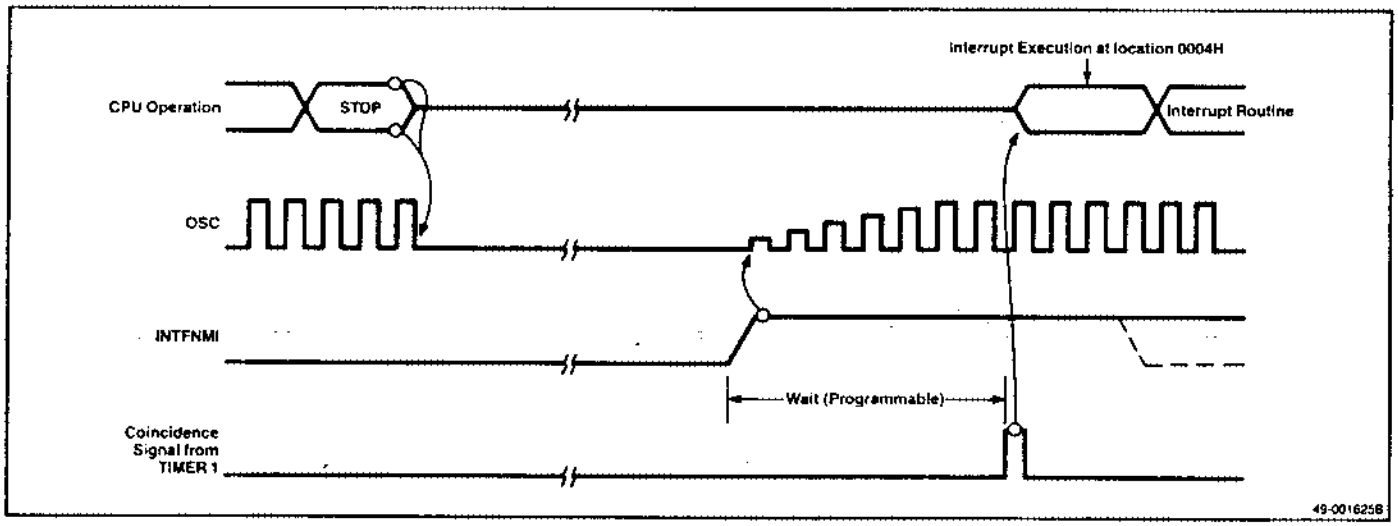


Figure 10-8. Software Stop Mode Release Timing



Unlike normal timer operation, when TIMER1 generates a match, the interrupt request flag is not set by the coincidence signal; the timer mode register (TMM) is set to FF, causing the timer to cease operation. Regardless of whether or not interrupts are disabled, the program jumps to address 0004H when the timer coincidence signal occurs.

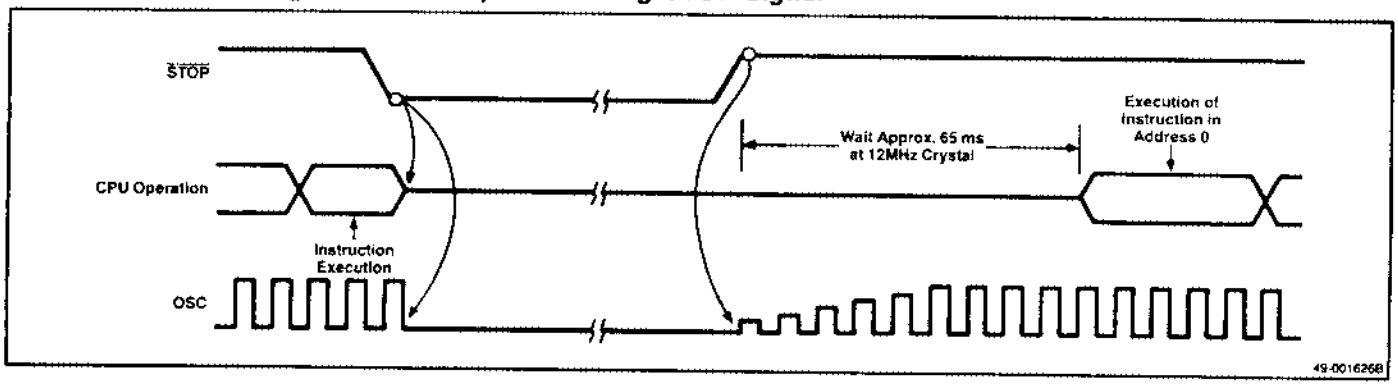
For long oscillator stabilization times, it may be necessary to use TIMER0 and TIMER1 in the 16-bit cascade mode.

Hardware Stop Mode

The processor also enters stop mode when the $\overline{STOP}$ input is asserted low for at least 500 ns. In hardware stop mode, all clocks are stopped and none of the on-chip peripherals function. Only the contents of internal RAM are retained and only the $\overline{STOP}$ and $\overline{RESET}$ circuitry operate. All output pins become high impedance. Following release of the stop mode, the contents of the internal registers are undefined.

The only way to release the hardware stop mode is to cause the $\overline{STOP}$ signal to go from low to high. When this happens, the oscillator starts. If $\overline{RESET}$ is high after $\overline{STOP}$ goes high, the CPU waits 65 ms and begins executing the program at location 0000H (see figure 10-9). The 65-ms delay allows the oscillator to stabilize.

Figure 10-9. Releasing Hardware Stop Mode Using $\overline{STOP}$ Signal



The hardware stop mode will not be released by asserting the $\overline{\text{RESET}}$ signal low; however, the stop mode may be released while $\overline{\text{RESET}}$ is low by changing the $\overline{\text{STOP}}$ signal from low to high. When this situation occurs (see figure 10-10), program execution will start from location 0000H when $\overline{\text{RESET}}$ goes from low to high without the CPU waiting 65 ms for the oscillator to stabilize. Thus, external circuitry must be provided to ensure that $\overline{\text{RESET}}$ stays low long enough for the oscillator to stabilize after $\overline{\text{STOP}}$ goes from low to high.

If the $\overline{\text{RESET}}$ signal goes low less than 65 ms after $\overline{\text{STOP}}$ goes high (see figure 10-11), the CPU will start its 65-ms delay, but the automatic delay circuitry will be terminated by $\overline{\text{RESET}}$ going low. Again, additional circuitry is required to ensure that the $\overline{\text{RESET}}$ signal stays low long enough to provide the required oscilator stabilization delay before $\overline{\text{RESET}}$ goes high and starts program execution. As in the software stop mode, the SB flag should be used to determine if program execution starting at location 0000H is the result of power-up or stop mode.

Figure 10-10. $\overline{\text{RESET}}$ Low Before $\overline{\text{STOP}}$ Goes High

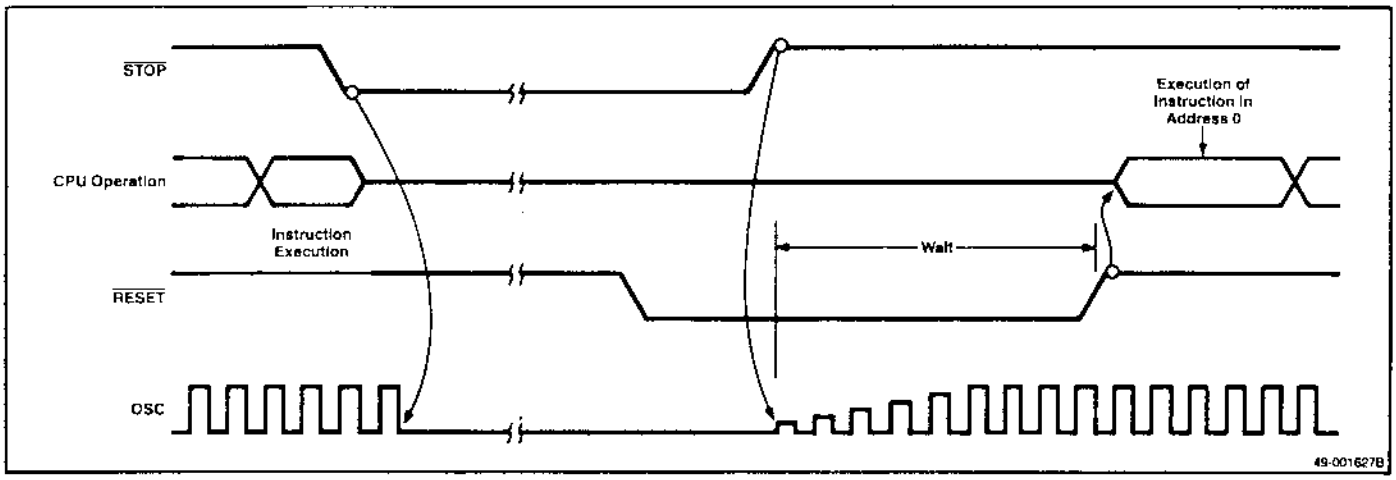
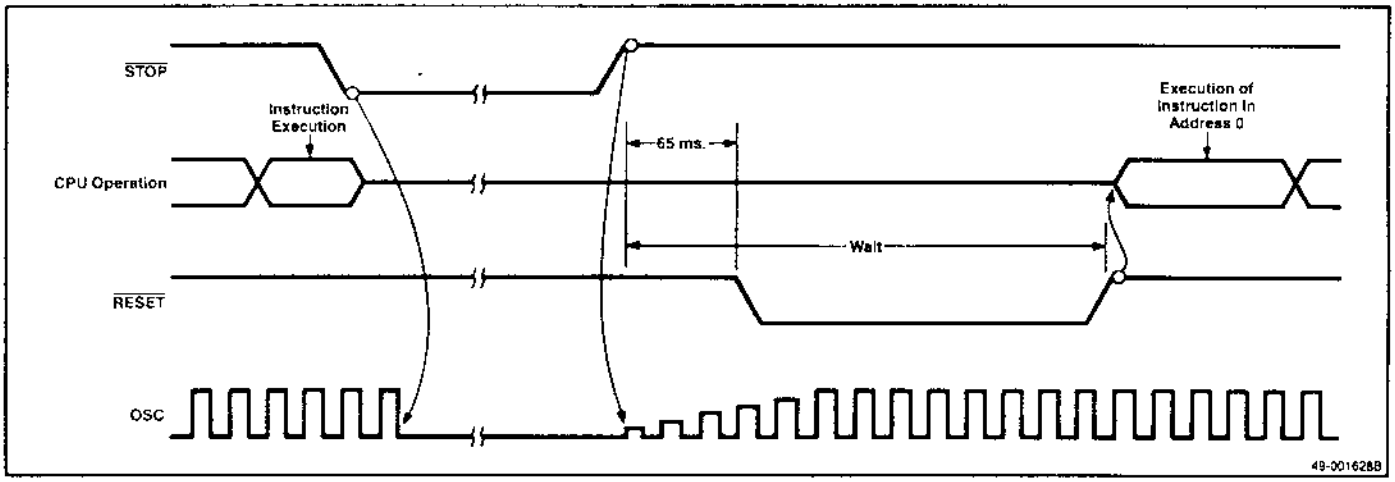


Figure 10-11. $\overline{\text{RESET}}$ After $\overline{\text{STOP}}$ Goes Low to High



OPERANDS

Table 11-1 lists the operand symbols appearing in the instruction set (Section 12) and abbreviated instruction set (Appendix B). Table 11-2 defines the operands and table 11-3 provides operand codes. Table 11-4 defines the graphic symbols used in describing instruction operation.

Table 11-1. Operand Symbols

Symbol	Allowable Operands
Registers	
r	V, A, B, C, D, E, H, L
r1	EAH, EAL, B, C, D, E, H, L
r2	A, B, C
Special Registers	
sr	PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, SML, EOM, ETMM, TMM, MM, MCC, MA, MB, MC, MF, TXB, TM0, TM1, ZCM (ZCM in 78C10/C11/C14 only)
sr1	PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, TMM, RXB, CR0, CR1, CR2, CR3
sr2	PA, PB, PC, PD, PF, MKH, ANM, MKL, SMH, EOM, TMM
sr3	ETM0, ETM1
sr4	ECNT, ECPT
Register Pairs	
rp	SP, B, D, H
rp1	V, B, D, H, EA
rp2	SP, B, D, H, EA
rp3	B, D, H
Register Pair Addressing	
rpa	B, D, H, D+ , H+ , D–, H–
rpa1	B, D, H
rpa2	B, D, H, D+, H+, D–, H–, D+byte, H+A, H+B, H+EA, H+byte
rpa3	D, H, D++, H++, D+byte, H+A, H+B, H+EA, H+byte
Flags	
f	CY, HC, Z
Interrupt Flags	
irf	INTFNMI, INTFT0, INTFT1, INTF1, INTF2, INTFE0, INTFE1, INTFEIN, INTFAD, INTFSR, INTFST, ER, OV, AN4, AN5, AN6, AN7, SB
Immediate Data	
wa	8-bit immediate data (low byte of working register address)
word	16-bit immediate data
byte	8-bit immediate data
bit	3-bit immediate data (b_2, b_1, b_0)

Table 11-2. Operand Definitions

Special Registers (sr-sr4)

PA = Port A	ECNT = Timer/event counter upcounter
PB = Port B	ECPT = Timer/event counter capture
PC = Port C	ETMM = Timer/event counter mode
PD = Port D	
PF = Port F	EOM = Timer/event counter output mode
MA = Mode A	
MB = Mode B	TXB = Transmit buffer
MC = Mode C	RXB = Receive buffer
MCC = Mode control C	SMH = Serial mode high
MF = Mode F	SML = Serial mode low
MM = Memory mapping	MKH = Mask high
TM0 = Timer register 0	MKL = Mask low
TM1 = Timer register 1	ANM = A/D channel mode
TMM = Timing mode	CR0 to CR3 = A/D conversion result 0-3
ETM0 = Timer/event counter register 0	
ETM1 = Timer/event counter register 1	
ZCM = Zero-cross mode control register	

Register Pairs (rp-rp3)

SP = Stack pointer	H = HL
B = BC	V = VA
D = DE	EA = Extended accumulator

Register Pair Addressing (rpa-rpa3)

B = (BC)	D++ = (DE)++
D = (DE)	H++ = (HL)++
H = (HL)	D+byte = (DE+byte)
D+ = (DE)+	H+byte = (HL+byte)
H+ = (HL)+	H+A = (HL+A)
D– = (DE)–	H+B = (HL+B)
H– = (HL)–	H+EA = (HL+EA)

Flags (f)

CY = Carry	HC = Half-carry	Z = Zero

Interrupt Flags (irf)

INTFNMI = NMI interrupt flag	INTFEIN = FEIN
	INTFAD = FAD
INTFT0 = FT0	INTFSR = FSR
INTFT1 = FT1	INTFST = FST
INTF1 = F1	ER = Error
INTF2 = F2	OV = Overflow
INTFE0 = FE0	AN4 to AN7 = Analog input 4-7
INTFE1 = FE1	SB = Standby

Table 11-3. Operand Codes

Registers (r, r2)

R_2	R_1	R_0	Reg	r	r2
0	0	0	V		
0	0	1	A		
0	1	0	B		
0	1	1	C		
1	0	0	D		
1	0	1	E		
1	1	0	H		
1	1	1	L		

Registers (r1)

T_2	T_1	T_0	Reg
0	0	0	EAH
0	0	1	EAL
0	1	0	B
0	1	1	C
1	0	0	D
1	0	1	E
1	1	0	H
1	1	1	L

Special Registers (sr, sr1, sr2)

S_5	S_4	S_3	S_2	S_1	S_0	Special Reg	sr	sr1	sr2
0	0	0	0	0	0	PA			
0	0	0	0	0	1	PB			
0	0	0	0	1	0	PC			
0	0	0	0	1	1	PD			
0	0	0	1	0	1	PF			
0	0	0	1	1	0	MKH			
0	0	0	1	1	1	MKL			
0	0	1	0	0	0	ANM			
0	0	1	0	0	1	SMH			
0	0	1	0	1	0	SML			
0	0	1	0	1	1	EOM			
0	0	1	1	0	0	ETMM			
0	0	1	1	0	1	TMM			
0	1	0	0	0	0	MM			
0	1	0	0	0	1	MCC			
0	1	0	0	1	0	MA			
0	1	0	0	1	1	MB			
0	1	0	1	0	0	MC			
0	1	0	1	1	1	MF			
0	1	1	0	0	0	TXB			
0	1	1	0	0	1	RXB			
0	1	1	0	1	0	TM0			
0	1	1	0	1	1	TM1			
1	0	0	0	0	0	CR0			
1	0	0	0	0	1	CR1			
1	0	0	0	1	0	CR2			
1	0	0	0	1	1	CR3			
1	0	1	0	0	0	ZCM			

Special Registers (sr3)

U_0	Special Reg
0	ETM0
1	ETM1

Special Registers (sr4)

V_0	Special Reg
0	ECNT
1	ECPT

Register Pairs (rp, rp2, rp3)

P_2	P_1	P_0	Reg Pair	rp	rp2	rp3
0	0	0	SP			
0	0	1	BC			
0	1	0	DE			
0	1	1	HL			
1	0	0	EA			

Register Pairs (rp1)

Q_2	Q_1	Q_0	Reg Pair
0	0	0	VA
0	0	1	BC
0	1	0	DE
0	1	1	HL
1	0	0	EA

Register Pair Addressing (rpa, rpa1, rpa2)

A_3	A_2	A_1	A_0	Addressing	rpa	rpa1	rpa2
0	0	0	0	—			
0	0	0	1	(BC)			
0	0	1	0	(DE)			
0	0	1	1	(HL)			
0	1	0	0	(DE)+			
0	1	0	1	(HL)+			
0	1	1	0	(DE)−			
0	1	1	1	(HL)−			
1	0	1	1	(DE+byte)			
1	1	0	0	(HL+A)			
1	1	0	1	(HL+B)			
1	1	1	0	(HL+EA)			
1	1	1	1	(HL+byte)			

Register Pair Addressing (rpa3)

C_3	C_2	C_1	C_0	Addressing
0	0	1	0	(DE)
0	0	1	1	(HL)
0	1	0	0	(DE)++
0	1	0	1	(HL)++
1	0	1	1	(DE+byte)
1	1	0	0	(HL+A)
1	1	0	1	(HL+B)
1	1	1	0	(HL+EA)
1	1	1	1	(HL+byte)

Table 11-3. Operand Codes (cont)

Flags (f)

F_2	F_1	F_0	Flag
0	0	0	—
0	1	0	CY
0	1	1	HC
1	0	0	Z

Interrupt Flags (irf)

I_4	I_3	I_2	I_1	I_0	Flag
0	0	0	0	0	NMI
0	0	0	0	1	FT0
0	0	0	1	0	FT1
0	0	0	1	1	F1
0	0	1	0	0	F2
0	0	1	0	1	FE0
0	0	1	1	0	FE1
0	0	1	1	1	FEIN
0	1	0	0	0	FAD
0	1	0	0	1	FSR
0	1	0	1	0	FST
0	1	0	1	1	ER
0	1	1	0	0	OV
1	0	0	0	0	AN4
1	0	0	0	1	AN5
1	0	0	1	0	AN6
1	0	0	1	1	AN7
1	0	1	0	0	SB

Table 11-4. Graphic Symbols

Symbol	Description
←	Transfer direction, result
∧	Logical product (logical AND)
∨	Logical sum (logical OR)
∀	Exclusive-OR
—	Complement
•	Concatenation

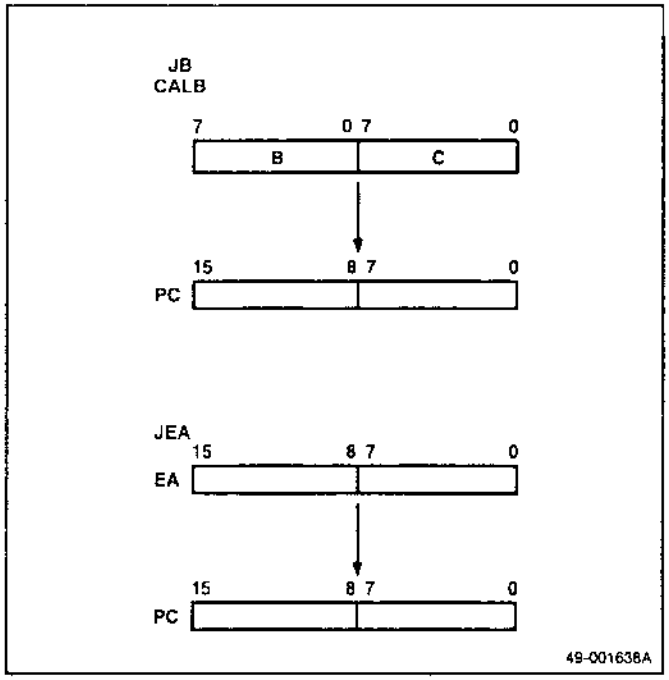
PROGRAM SEQUENCING

The address of the next instruction is generally indicated by the program counter (PC), which is usually automatically incremented by the number of bytes contained in the current instruction being executed. A branch operation, however, loads a given address into the PC, which causes the program to branch out of normal program sequence and go directly to that location. There are several branching methods, and these include: register branching, immediate-data branching, direct branching, relative branching, and extended-relative branching.

Register Branching

The contents of either the general-purpose register pair BC or the extended accumulator EA are loaded into the PC, causing a jump to the new location stored in the PC upon execution of the JB, CALB, or JEA instruction. See figure 11-1.

Figure 11-1. Execution of JB, CALB, or JEA Instruction



Immediate-Data Branching

The immediate data contained in the second and third bytes of the instruction is loaded into the PC, causing a branch to the new location stored in the PC upon execution of the JMP word, CALL word, or CALF word instruction. See figure 11-2.

For the CALF instruction, instead of the second and third bytes, the lower-order 3 bits of the first byte and the contents of the second byte are loaded into the PC as shown in figure 11-2. This results in a call within a fixed area of memory of 000-7FFH from the current PC location.

Direct Branching

The contents of the memory location specified by bits 0-4 of the data following the instruction code are used, causing a branch to the new location stored in the PC upon execution of the CALT instruction. See figure 11-3.

Relative Branching

The value obtained by adding the byte count of the current instruction plus 1 and an offset obtained from the lower-order 6 bits of the data following the instruction code is added to the PC. The offset is treated as a signed two's complement (−31 to + 31) with bit 5 used as the sign bit. This causes a jump to the new address in the PC upon execution of the JR instruction. See figure 11-4.

Figure 11-2. Execution of CALL, JMP, or CALF Instruction

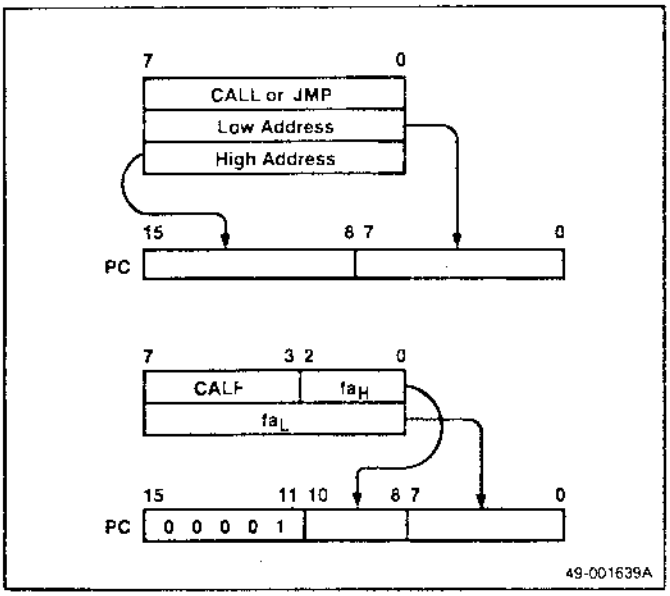


Figure 11-3. Execution of the CALT Instruction

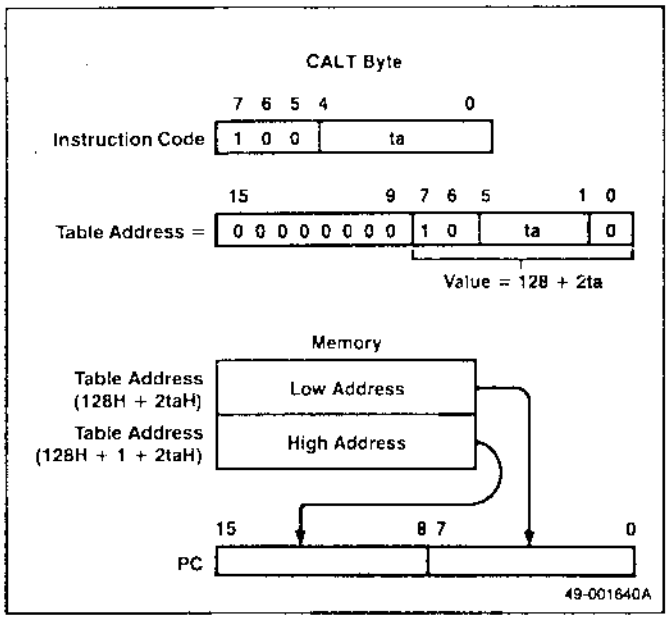
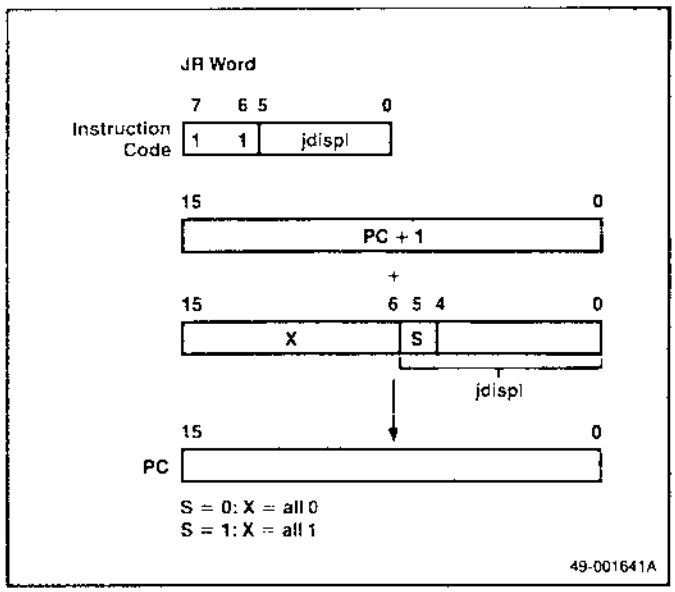


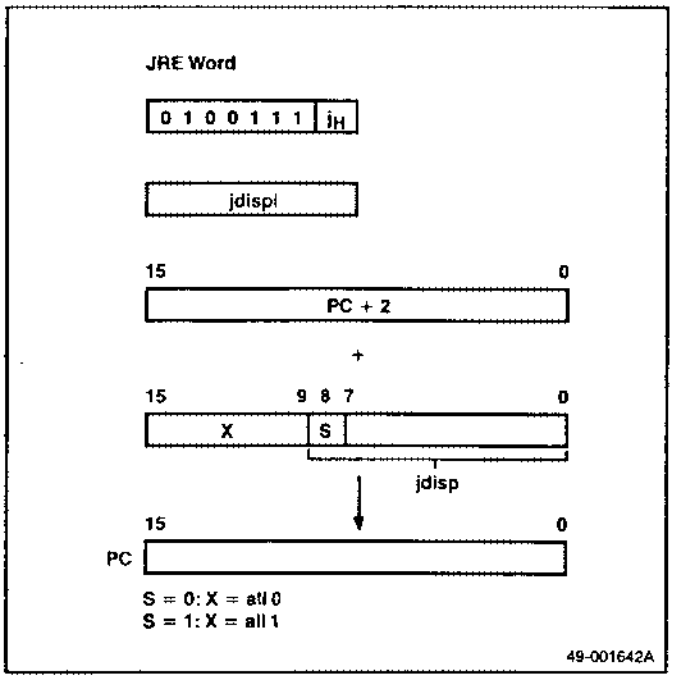
Figure 11-4. Execution of the JR Instruction



Extended-Relative Branching

The value obtained by adding the byte count of the next instruction and an offset obtained from the lower-order 9 bits of the immediate data following the instruction code is loaded into the PC. The offset is treated as a signed two's complement (−255 to +255). Bit 0 of the first byte of the instruction is used as the sign bit. This causes a jump to the new address in the PC upon execution of the JRE instruction. See figure 11-5.

Figure 11-5. Execution of the JRE Instruction



ADDRESSING MODES

Register Addressing

Registers are specified as shown in table 11-1. Both 8- and 16-bit registers may be specified, with general-purpose registers specified as individual 8-bit registers or as certain 16-bit pairs.

Example 1. In this example of a MOV r1,A instruction, E is used for operand r1 and a comment is appended to the instruction. A semicolon introduces the comment. The comment has no effect on instruction execution.

Instruction:
MOV E,A ;(E) ← (A)

Instruction Code:
0001 1101 (1DH)

Example 2. This example of a DCX rp instruction ($00P_1P_0$ 0011) uses register pair HL.

Instruction:
DCX H ;(HL) ← (HL) — 1

Instruction Code:
0011 0011 (33H)

Register-Indirect Addressing

This mode of addressing specifies the contents of a register pair as the operand address. A special form of this mode uses either register pair DE or HL as a 16-bit base register. In this mode, the contents of the base register specify the operand address in the auto-increment, autodecrement, double-autoincrement, base, and base-index addressing modes.

In register-indirect addressing, registers are specified as shown in table 11-1.

This example of an LDAX rpa2 instruction ($A_3010\ 1A_2A_1A_0$) uses register pair BC.

Instruction:
LDAX B ;(A) ← ((BC))

Instruction Code:
0010 1001 (29H)

Autoincrement Addressing

This is a special mode of register-indirect addresing using register pairs DE or HL. The contents of DE or HL specify the operand address. The contents are incremented by 1 after addressing memory. The operand is specified as follows.

Symbol	Operand
rpa	D+, H+
rpa2	D+, H+

STAX. This example of a STAX rpa2 instruction ($A_3011\ 1A_2A_1A_0$) uses register pair DE.

Instruction:
STAX D+ ;((DE)) ← (A), (DE) ← (DE) + 1

Instruction Code:
0011 1100 (3CH)

BLOCK. In the BLOCK instruction, the operands are implicit in the instruction. This operation requires the use of register pair HL as the source address register, register pair DE as the destination address register, and register C as a counter. After data is transferred from the source address to the destination address, both register pairs HL and DE are automatically incremented as follows.

```
BLOCK    ;((DE)) ← ((HL)),  (DE) ← (DE) + 1,
         ;(HL) ← (HL) + 1,  (C) ← (C) − 1
```

RET or POP. No operand is specified for either a return or pop instruction. Each is used to restore data saved on the stack by loading the contents of the stack pointer into the program counter and automatically incrementing the stack pointer. Symbolically, for the return instruction, the operation looks like this.

```
RET      ;(PCL) ← ((SP)),  (PCH) ← (((SP) + 1)),
         ;(SP) ← (SP) + 2
```

Autodecrement Addressing

The contents of either register pair DE or HL specify the operand address. After memory addressing, the contents are decremented by 1. The operand is specified as follows.

Symbol	Operand
rpa	D−, H−
rpa2	D−, H−

Example 1. This example of an ADDX rpa instruction (0111 0000 1100 $0A_2A_1A_0$) uses register pair HL.

Instruction:

```
ADDX  H−   ;(A) ← (A) + ((HL)), (HL) ← (HL) − 1
```

Instruction Code:

```
0111  0000  1100  0111     (70C7H)
```

Example 2. When an interrupt is accepted, or when a CALL or PUSH instruction is executed, no operand is specified. These instructions save the contents of the program counter and PSW in the stack and then decrement the stack pointer. Using the SOFTI interrupt, this operation is shown symbolically as follows.

```
SOFTI    ;((SP) − 1) ← (PSW),
         ;((SP) − 2) ← (PC) + 1(HB),
         ;(SP − 3) ← (PC) + 1(LB),
         ;(SP) ← (SP) − 3,
         ;(PC) ← 0060H
```

(HB) = high byte, (LB) = low byte

Double-Autoincrement Addressing

The contents of either register pair DE or HL specify the operand address. After memory transfer, the contents of the register pair are incremented by 2. This is used for 16-bit data transfers between the extended accumulator and mapped memory. The operand is specified as follows.

Symbol	Operand
rpa3	D++, H++

This example of an STEAX rpa3 instruction (0100 1000 1001 C_3-C_0) uses register pair HL.

Instruction:

```
STEAX  H++   ;((HL)) ← (EAL),
             ;((HL) + 1) ← (EAH),
             ;(HL) ← (HL) + 2
```

Instruction Code:

```
0100  1000  1001  0101     (4895H)
```

Base Addressing

The contents of either register pair DE or HL and an offset obtained from the data following the instruction code are added to specify the operand address. The operand is specified as follows:

Symbol	Operand
rpa2	D+byte, H+byte
rpa3	D+byte, H+byte

In this example of an STAX rpa2 instruction ($A_3$011 $1A_2A_1A_0$ d_7-d_4 d_3-d_0), the register pair is HL and the offset value is 10H.

Instruction:

```
STAX   H+10H   ;((HL) + 10H) ← (A)
```

Instruction Code:

```
1011  1111  0000  0010     BF02H
```

Base-Index Addressing

The sum of the contents of the base register (either register pair DE or HL) and the contents of a designated register (register A, B, or EA) specifies the operand address. The operand is specified as follows.

Symbol	Operand
rpa2	H+A, H+B, H+EA
rpa3	H+A, H+B, H+EA

In this example of an LDAX rpa2 instruction ($A_3$010 $1A_2A_1A_0$), register pair HL is used as the base register and register B provides the offset.

Instruction:
LDAX H+B ;(A) ← ((HL) +(B))

Instruction Code:
1010 1101 (ADH)

Working-Register Addressing

The vector register (register V) holds the upper byte of the operand address. The immediate data following the instruction code specifies the lower-order byte of the address. This allocates a 256-byte area in memory for use as working registers or scratch-pad memory. The operand is specified as follows:

Symbol	Operand
wa	A label not to exceed FFH

The following example of a DCRW wa instruction (0011 0000 d_7-d_4 d_3-d_0) specifies 77H in the operand.

Instruction:
DCRW 77H

Instruction Code:
0011 0000 0111 0111 (3077H)

If the contents of register V are equal to 20H, the operand address is 2077H and the working register is decremented by 1.

Accumulator-Indirect Addressing

The contents of the memory location addressed by PC + 3 + A are loaded into register C and the contents of the memory location addressed by PC + 3 + A + 1 are loaded into register B. This addressing mode is initiated by the TABLE instruction.

In this example, the contents of the accumulator are equal to 0, the contents of the PC are equal to 100H, and the instruction looks like this:

TABLE ;(C) ← (103H), (B) ← (104H)

Immediate Addressing

The operand is specified by the data byte immediately following the instruction code. The operand is specified as follows:

Symbol	Operand
byte	8 bits of data

In this example of an ADI A,byte instruction (0100 0110 d_7-d_4 d_3-d_0), 79H is specified in the immediate data.

Instruction:
ADI A,79H ;(A) ← (A) + 79H

Instruction Code:
0100 0110 0111 1001 (4679H)

Extended-Immediate Addressing

The address of the operand is specified by the data word immediately following the instruction code. The operand is specified as follows:

Symbol	Operand
byte	A value which does not exceed FFFFH

In this example of an LXI rp2,word instruction (0$P_2P_1P_0$ 0100 d_7-d_0 d_{15}-d_8), register pair HL is specified as the destination and 3F54H is specified in the immediate data.

Instruction:
LXI H,3F54H ;(HL) ← 3F54H

Instruction Code:
0011 0100 0101 0100 0011 1111
(34543FH)

Direct Addressing

The operand is located at the address immediately following the instruction code. The operand is specified as follows.

Symbol	Operand
word	A value not to exceed FFFFH

Example 1. In this example of a MOV r,word instruction (0111 0000 0110 $1R_2R_1R_0$ d_7-d_0 d_{15}-d_8), register B is specified as the destination and EEFFH is specified in the second word.

Instruction:
MOV B,0EEFFH ;(B) ← (EEFFH)

Instruction Code:
0111 0000 0110 1010 1111 1111 1110 1110
(706AFFEEH)

Example 2. In this example of an SDED word instruction (0111 0000 0010 1110 d_7-d_0 d_{15}-d_8), the label DST refers to location 4000H.

Instruction:
SDED DST

Instruction Code:
0111 0000 0010 1110 0000 0000 0100 0000
(702E0040H)

SKIP OPERATION TIMING

Descriptions of the instructions in Section 12 and Appendix B include the number of timing states. For the timing state requirements, two numbers are given. The first is the number of timing states required to fetch and execute the instruction. A second value in parentheses indicates the number of timing states used when the instruction is skipped; this includes fetching.

When an instruction is skipped, all bytes of the instruction are fetched but they are not executed. Idle states are generated, and the sum of idle states for all the bytes in the instruction equals the number of timing states used when that instruction is skipped. Specific bytes have a prescribed number of idle states; for instance, a byte of opcode generates four idle states and a byte of immediate data generates three idle states.

For example, MVI sr2,byte (0110 0100 $S_3$000 0$S_2S_1S_0$ d_7-d_0), is a three-byte instruction with the opcode consisting of the first and second bytes and the third byte containing the immediate data. The two opcode bytes generate 4 idle states each and the byte of immediate data generates 3 idle states, for a total of 11 timing states when this instruction is skipped.

OVERLAY INSTRUCTIONS

The instruction set contains three instructions divided into two groups, A and B.

Group	Instruction	Flag
A	MVI A,byte	L1
B	MVI L,byte and LXI H,word	L0

The overlay feature is used when one of these instructions is encountered in a program and one or more consecutive locations contain the same instruction. The program will execute the first instruction encountered in the sequence and the remaining instructions will be ignored as if they were NOP instructions. The program will continue to do this until an instruction of a different type is encountered. At this point the program will start executing the instructions normally.

If the program encounters an instruction in group A, the L1 flag is set. If the program encounters an instruction in group B, the L0 flag is set. These flags will stay set until the program encounters an instruction of a different type, at which time they reset. If an interrupt occurs during the execution of an overlay instruction, the L0 and L1 flags are saved in bits 2 and 3 respectively of the PSW. After servicing the interrupt, the PSW will be restored and the program can continue testing subsequent instructions against L0 and L1.

The following program illustrates the use of the overlay instructions. This program will clear one of these registers: W, X, Y, or Z. The registers are located in the following memory locations and are four bytes long.

Location: 1000H	Register W	Register X
1008H	Register Y	Register Z
1010H		

The program to clear one of the registers is:

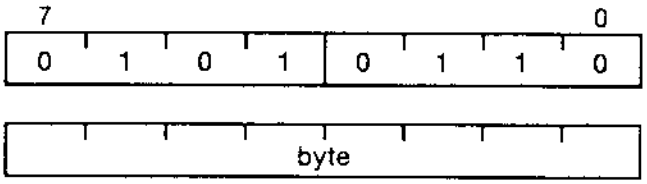
```
CLW:  MVI   L,00H    ;Clear register W.
CLX:  MVI   L,04H    ;Clear register X.
CLY:  MVI   L,08H    ;Clear register Y.
CLZ:  MVI   L,0CH    ;Clear register Z.
      MVI   H,10H    ;Load register H with
                     10H.
      MVI   C,03H    ;Set counter.
      XRA   A,A      ;Clear A.
LOOP: STAX  H+       ;((HL)) ← 0H,
                     ;(HL) ← (HL) + 1
      DCR   C        ;Skip if borrow.
      JR    LOOP
      RET
```

To clear register X, 04H is loaded into register L. MVI L,08H and MVI L,0CH are the remaining overlay instructions and they are ignored and replaced by 14 idle cycles (NOPs). The high-order byte of the register is put into register H by the MVI H,10H instruction and the starting address of register X (1004H) is contained in register pair HL. The remaining part of the program clears register X.

ACI A,byte

Add immediate data byte to A with carry.

(A) ← (A) + byte + (CY)



Instruction Code: 56H, byte

Adds the immediate data byte and CY flag to the contents of the accumulator and stores the result in the accumulator.

Bytes: 2

T-States: 7 (7)

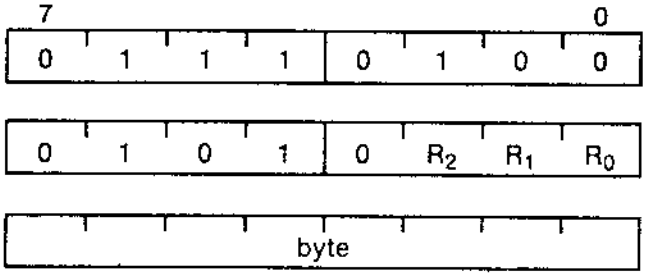
Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

Example:
```
ACI     A,20H     ;Add 20H to accumulator.
```

ACI r,byte

Add immediate data byte to register with carry.

(r) ← (r) + byte + (CY)



Instruction Code: 745(0-7)H, byte

Adds the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), and the CY bit with the immediate data byte. Leaves the result in the designated register.

Bytes: 3

T-States: 11 (11)

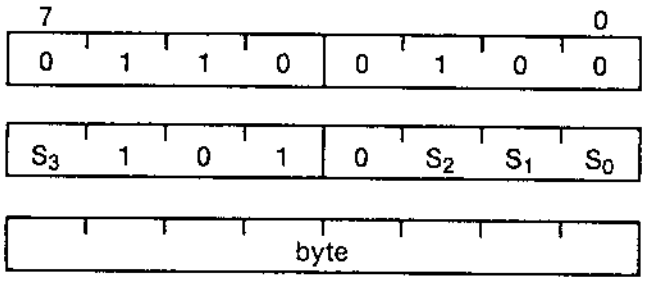
Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

Example:
```
ACI     D,20H     ;Add 20H to D-register and
                  ;include carry.
```

ACI sr2,byte

Add immediate data byte to special register with carry.

(sr2) ← (sr2) + byte + (CY)



Instruction Code:
645(0-3, 5-7)H, byte
64D(0, 1, 3, 5)H, byte

Adds the immediate data byte and the CY flag to the contents of special register sr2 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, or TMM), designated by S_3-S_0 (0-3, 5-9, B, D), and leaves the result in the designated special register.

Bytes: 3

T-States: 20 (11)

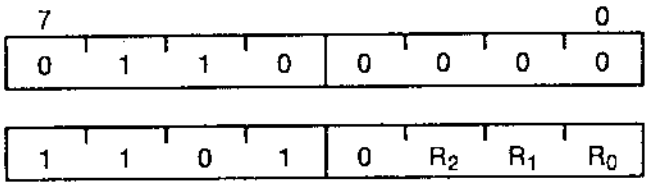
Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

Example:
```
ACI     PD,20H    ;Add 20H to port D and
                  ;include carry.
```

ADC A,r

Add register to A with carry.

(A) ← (A) + (r) + (CY)



Instruction Code: 60D(0-7)H

Adds the carry bit (CY) to the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), and the accumulator and leaves the sum in the accumulator.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
- Z
- SK ← 0
- HC
- L0 ← 0
- L1 ← 0
- CY

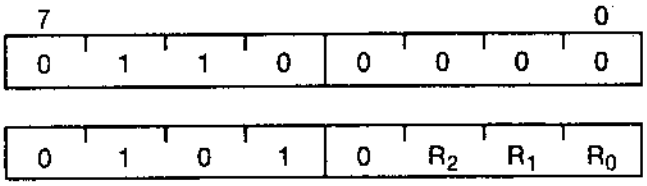
Example:

```
ADC     A,E     ;Add the contents of A and the
                ;E-register including the CY
                ;and leave the result in A.
```

ADC r,A

Add A to register with carry.

(r) ← (r) + (A) + (CY)



Instruction Code: 605(0-7)H

Adds the contents of the accumulator and that of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), and the CY flag and leaves the sum in the designated register.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
- Z
- SK ← 0
- HC
- L0 ← 0
- L1 ← 0
- CY

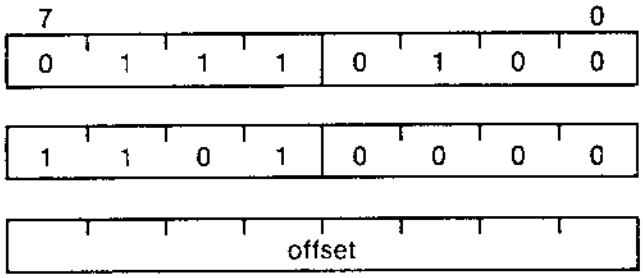
Example:
Routine to add register pair DE to register pair HL and store the sum in HL.

```
MOV     A,E     ;(A) ← (E)
ADD     L,A     ;(L) ← (L) + (A)
MOV     A,D     ;(A) ← (D)
ADC     H,A     ;(H) ← (H) + (A) + (CY)
```

ADCW wa

Add working register to A with carry.

(A) ← (A) + ((V) • wa) + (CY)



Instruction Code: 74D0H, offset

Adds the contents of the accumulator to the contents of the working register, including the carry, and stores the result in the accumulator. The working register is addressed by the V-register (high-order 8 bits) and the offset (low-order 8 bits).

Bytes: 3

T-States: 14 (11)

Flag Bits Affected:
- Z
- SK ← 0
- HC
- L0 ← 0
- L1 ← 0
- CY

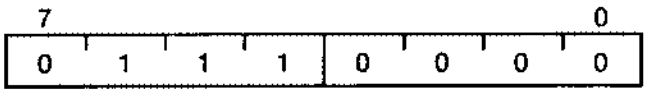
Example:

```
MVI     V,0FFH
ADCW    20H     ;Add contents of FF20H to
                ;accumulator.
```

ADCX rpa

Add memory addressed by register pair to A with carry.

(A) ← (A + ((rpa)) + (CY)



Instruction Code: 70D(1-7)H

Adds the contents of the accumulator, including the carry flag, to the contents of memory addressed by register pair rpa (BC, DE, HL, DE+, HL+, DE-, or HL-) and stores the result in the accumulator. The register pair is designated by A_2-A_0 (1-7).

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

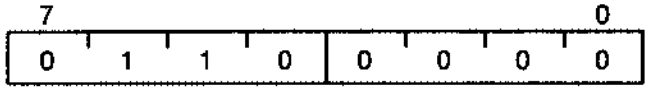
Example:

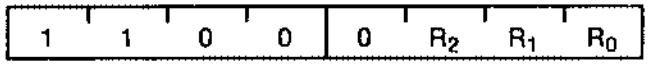
```
ADCX    H       ;Add memory location
                ;specified by HL to A with
                ;carry.
```

ADD A,r

Add register to A.

(A) ← (A) + (r)





Instruction Code: 60C(0-7)H

Adds the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), to the contents of the accumulator and stores the result in the accumulator.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

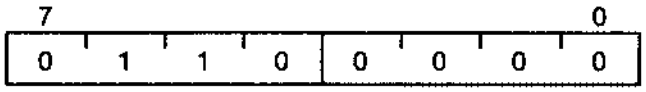
Example:

```
ADD     A,C     ;Add A to C and leave result in A.
```

ADD r,A

Add A to register.

(r) ← (r) + (A)



Instruction Code: 604(0-7)H

Adds the contents of the accumulator to register r (V, A, B, C, D, E, H, or L) and stores the result in the register. The register is designated by R_2-R_0 (0-7).

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

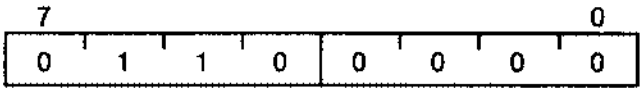
Example:

```
ADD     D,A     ;Add A to D and leave result in D.
```

ADDNC A,r

Add register to A; skip next instruction if carry not set.

(A) ← (A) + (r), SK/NC



Instruction Code: 60A(0-7)H

Adds the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), to the accumulator and stores the result in the accumulator. If no carry is generated, the next instruction is skipped.

Bytes: 2

T-States: 8 (8)

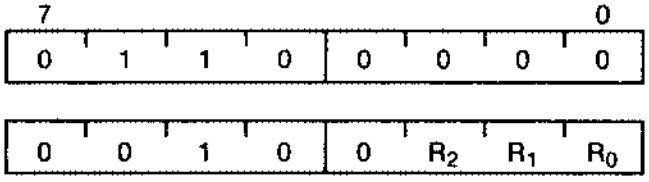
Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

Example:
```
ADDNC   A,D     ;Add A to D and leave result
                ;in A. Skip next instruction if
                ;no carry is generated.
```

ADDNC r,A

Add A to register; skip next instruction if carry not set.

(r) ← (r) + (A), SK/NC



Instruction Code: 602(0-7)H

Adds the contents of the accumulator to register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), and stores the result in the register. If no carry is generated, the next instruction is skipped.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

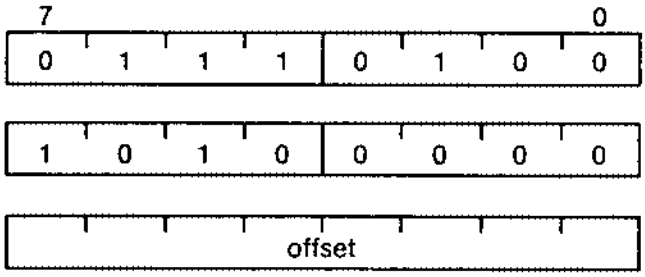
Example:
```
ADDNC   L,A     ;Add A to L and leave result
                ;in L. Skip next instruction if
                ;no carry is generated.
ADI     H,1     ;H ← H + 1
```

If no carry is present after adding A and L, the ADI instruction is skipped and the addition is complete. If a carry is present, finish the addition by adding 1 to H.

ADDNCW wa

Add working register to A; skip next instruction if no carry.

(A) ← (A) + ((V) • wa), SK/NC



Instruction Code: 74A0H, offset

Adds the contents of the accumulator to the contents of the working register and stores the result in the accumulator. If no carry is generated, the next instruction is skipped. The working register is addressed by the V-register (8 high-order address bits) and the offset (8 low-order address bits).

Bytes: 3

T-States: 14 (11)

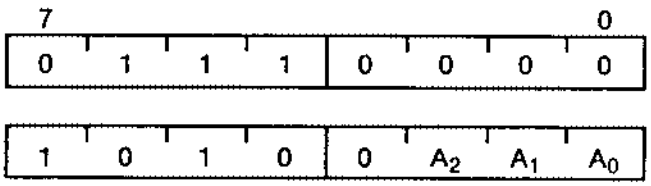
Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

Example:
```
MVI     V,0F0H
ADDNCW  10H     ;Add A to contents of location
                ;0F010H and leave result in A.
RET             ;Skip this instruction if no
                ;carry is generated.
```

ADDNCX rpa

Add memory addressed by register pair to A; skip next instruction if no carry.

(A) ← (A) + ((rpa)), SK/NC



Instruction Code: 70A(1-7)H

Adds the contents of memory addressed by register pair rpa (BC, DE, HL, DE+, HL+, DE-, or HL-), designated by A_2-A_0 (1-7), to the contents of the accumulator and stores the result in the accumulator. If no carry is generated by the addition, then the next instruction is skipped.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
- Z
- SK
- HC
- L0 ← 0
- L1 ← 0
- CY

Example:

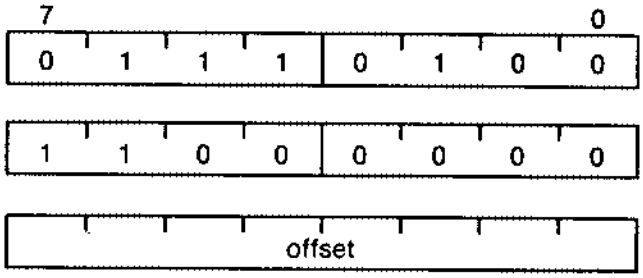
Add the contents of addresses 1100H and 1000H, and store the sum at location 1200H if a carry is generated. If no carry, jump to EXIT.

```
BEGIN:  LXI     H,1200H   ;(HL) ← 1200H
        LXI     D,1000H   ;(DE) ← 1000H
        MOV     A,1100H   ;(A) ← (1100H)
        ADDNCX  D+        ;(A) ←
                          ;(A) + ((DE)),
                          ;(DE) ← (DE) + 1
        STAX    H         ;((HL)) ← (A)
        JMP     EXIT      ;Finished; exit.
```

ADDW wa

Add working register to A.

(A) ← (A) + ((V) • wa)



Instruction Code: 74C0H, offset

Adds the contents of the accumulator to the contents of the working register and stores the result in the accumulator.

Bytes: 3

T-States: 14 (11)

Flag Bits Affected:
- Z
- SK ← 0
- HC
- L0 ← 0
- L1 ← 0
- CY

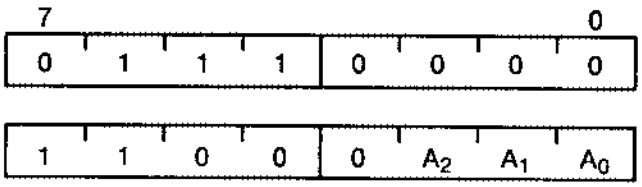
Example:

```
MVI     V,F0H   ;Add contents of location
ADDW    10H     ;F010H to A and leave result
                ;in A.
```

ADDX rpa

Add memory to A.

(A) ← (A) + ((rpa))



Instruction Code: 70C(1-7)H

Adds the contents of memory addressed by register pair rpa (BC, DE, HL, DE+, HL+, DE-, or HL-), designated by A_2-A_0 (1-7), to the contents of the accumulator and stores the result in the accumulator.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

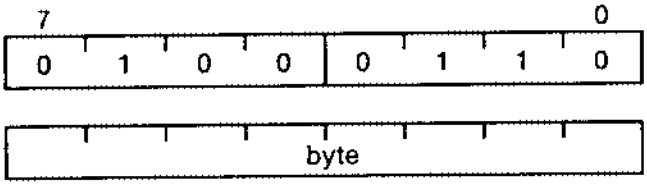
Example:
Add the contents of addresses 100H and 200H and store the sum in the accumulator.

```
BEGIN:  LXI     H,200H   ;(HL) ← 200H
        MOV     A,100H   ;(A) ← (100H)
        ADDX    H        ;(A) ← (A) + ((HL))
        JMP     EXIT     ;Finished; exit.
```

ADI A,byte

Add immediate data byte to A.

(A) ← (A) + byte



Instruction Code: 46H, byte

Adds the value of the accumulator to the immediate data byte and leaves the result in the accumulator.

Bytes: 2

T-States: 7 (7)

Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

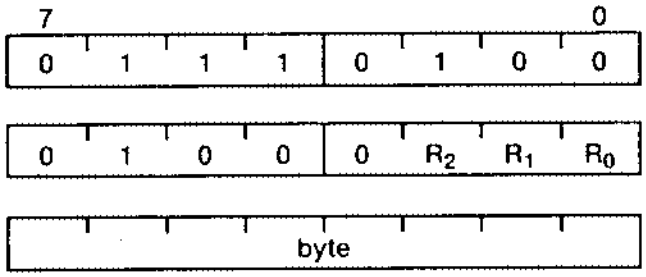
Example:

```
ADI     A,20H    ;Add 20H to the accumulator
                 ;without the carry.
```

ADI r,byte

Add immediate data byte to register.

(r) ← (r) + byte



Instruction Code: 744(0-7)H, byte

Adds the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), to the immediate data byte and leaves the result in the designated register.

Bytes: 3

T-States: 11 (11)

Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

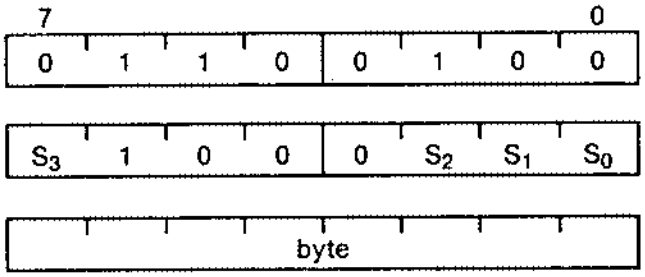
Example:

```
ADI     D,20H    ;Add 20H to the D-register.
```

ADI sr2,byte

Add immediate data byte to special register.

(sr2) ← (sr2) + byte



Instruction Code:
644(0-3, 5-7)H, byte
64C(0, 1, 3, 5)H, byte

Adds the contents of special register sr2 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, or TMM), designated by S_3-S_0 (0-3, 5-9, B, D), to the immediate data byte and leaves the result in the designated special register.

Bytes: 3

T-States: 20 (11)

Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

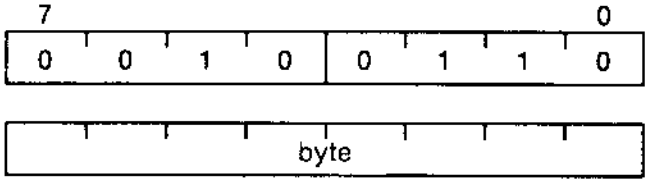
Example:

```
ADI     PD,20H    ;Add 20H to the contents of
                  ;port D.
```

ADINC A,byte

Add immediate data byte to A; skip next instruction if carry not set.

(A) ← (A) + byte, SK/NC



Instruction Code: 26H, byte

Adds the value of the accumulator to the immediate data byte and leaves the result in the accumulator. Skips the next instruction if no carry is generated.

Bytes: 2

T-States: 7 (7)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

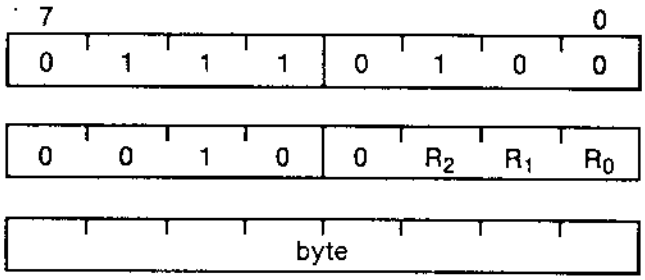
Example:

```
ADINC   A,20H     ;Add 20H to contents of
                  ;accumulator and skip next
                  ;instruction if no carry is
                  ;generated.
```

ADINC r,byte

Add immediate data byte to register; skip next instruction if no carry.

(r) ← (r) + byte, SK/NC



Instruction Code: 742(0-7)H, byte

Adds the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), to the immediate data byte and leaves the result in the designated register. If no carry is generated by the addition, then the next instruction is skipped.

Bytes: 3

T-States: 11 (11)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

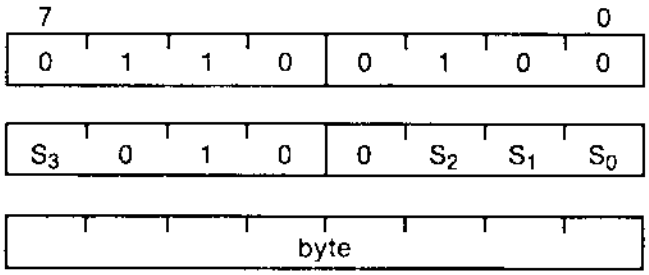
Example:

```
ADINC   D,20H     ;Add 20H to D-register.
MVI     A,D       ;Skip this instruction if no
                  ;carry is generated.
```

ADINC sr2,byte

Add immediate data byte to special register; skip next instruction if no carry.

(sr2) ← (sr2) + byte, SK/NC



Instruction Code:
642(0-3, 5-7)H, byte
64A(0, 1, 3, 5)H, byte

Adds the contents of special register sr2 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, or TMM), designated by S_3-S_0 (0-3, 5-9, B, D), to the immediate data byte and leaves the result in the designated special register. If no carry is generated, then the next instruction is skipped.

Bytes: 3

T-States: 20 (11)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

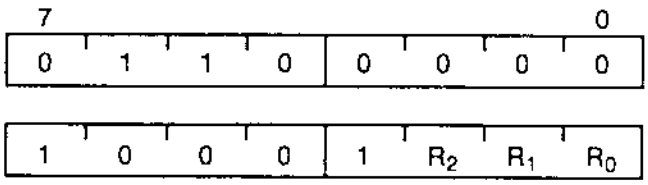
Example:

```
ADINC   PD,20H   ;Add 20H to contents of
                 ;port D.
MOV     A,PD     ;Skip this instruction if no
                 ;carry was generated.
```

ANA A,r

AND register with A.

(A) ← (A) AND (r)



Instruction Code: 608(8-F)H

Performs logical AND between the contents of the accumulator and register r (V, A, B, C, D, E, H, or L) designated by R_2-R_0 (0-7). Leaves the result in the accumulator.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

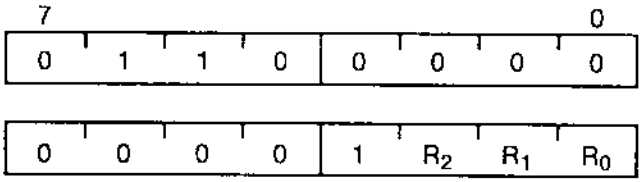
Example:

```
ANA     A,C      ;AND the accumulator with
                 ;the C-register.
```

ANA r,A

AND A with register.

(r) ← (r) AND (A)



Instruction Code: 600(8-F)H

Performs logical AND between the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), and that of the accumulator. Leaves the result in the register.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

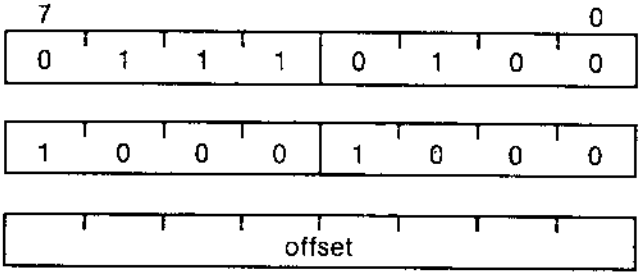
Example:

```
ANA     C,A      ;AND the accumulator with
                 ;the C-register.
```

ANAW wa

AND working register with A.

(A) ← (A) AND ((V) • wa)



Instruction Code: 7488H, offset

Instruction Code: 7488H, offset

Performs a logical AND between the contents of the accumulator and the contents of a working register, and leaves the result in the accumulator.

Bytes: 3

T-States: 14 (11)

Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

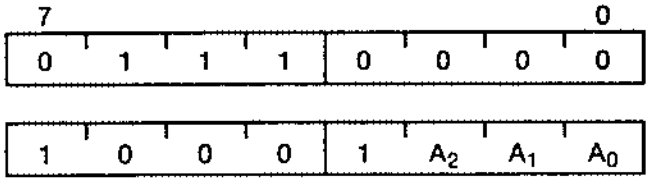
Example:

```
MVI     V,0FFH
ANAW    10H     ;AND the accumulator with
                ;the contents of memory
                ;location FF10H.
```

ANAX rpa

AND memory addressed by register pair with A.

(A) ← (A) AND ((rpa))



Instruction Code: 708(9-F)H

Performs a logical AND between the contents of the accumulator and the contents of the memory location addressed by register pair rpa (BC, DE, HL, DE+, HL+, DE−, or HL−). The register pair is designated by A_2-A_0 (1-7). Leaves the result in the accumulator.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

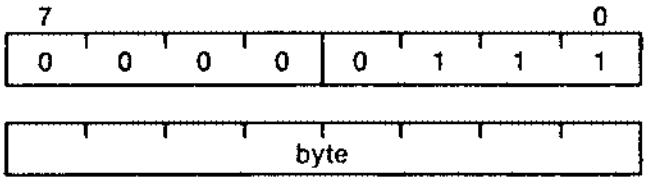
Example:

```
ANAX    D       ;(A) ← (A) AND ((DE)).
```

ANI A,byte

AND immediate data byte with A.

(A) ← (A) AND byte



Instruction Code: 07H, byte

Performs a logical AND of the contents of the accumulator with the immediate data byte and leaves the result in the accumulator.

Bytes: 2

T-States: 7 (7)

Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

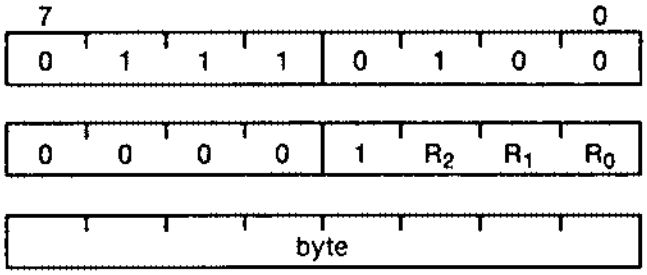
Example:

```
ANI     A,20H   ;AND 20H to the accumulator.
```

ANI r,byte

AND immediate data byte with register.

(r) ← (r) AND byte



Instruction Code: 740(8-F)H

Performs a logical AND of the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), with the immediate data byte. Leaves the result in the designated register.

Bytes: 3

T-States: 11 (11)

Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

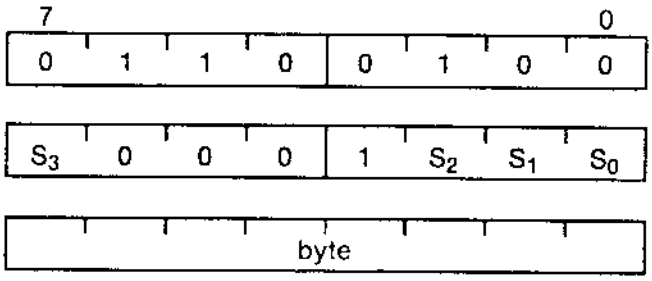
Example:

```
ANI     D,20H   ;AND 20H to D-register.
```

ANI sr2,byte

AND immediate data byte with special register.

(sr2) ← (sr2) AND byte



Instruction Code:
640(8-B, D-F)H, byte
648(8, 9, B, D,)H, byte

Performs a logical AND of the contents of special register sr2 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, or TMM), designated by S_3-S_0 (0-3, 5-9, B, D), with the immediate data byte, leaving the result in the designated special register.

Bytes: 3

T-States: 20 (11)

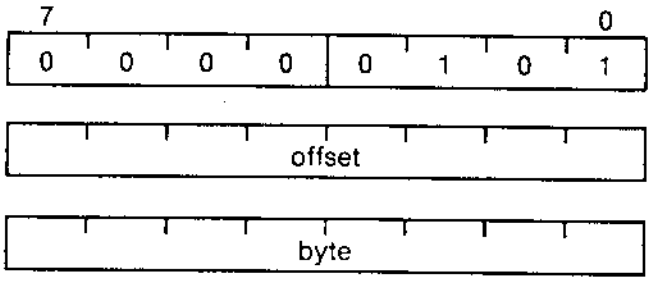
Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
ANI     PD,20H    ;AND 20H to port D.
```

ANIW wa,byte

AND immediate data byte with working register.

((V) • wa) ← ((V) • wa) AND byte



Instruction Code: 05H, offset, byte

Performs a logical AND of the immediate data byte with the contents of the working register and stores the result in the working register.

Bytes: 3

T-States: 19 (10)

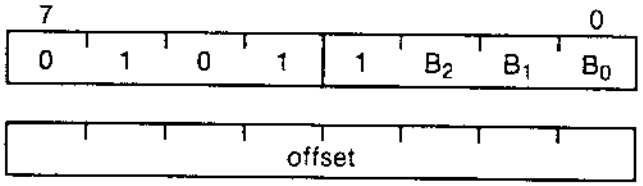
Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
MVI     V,0FFH
ANIW    10H,20H   ;AND 20H to contents of
                  ;memory location FF10H and
                  ;leave result in FF10H.
```

BIT bit,wa

Bit test working register and skip next instruction if set.

If (bit) = 1 then skip next instruction



Instruction Code: 5(8-F)H, offset

Skips the next instruction if the bit designated by B_2-B_0 (0-7) in the working register is set to 1.

Bytes: 2

T-States: 10 (7)

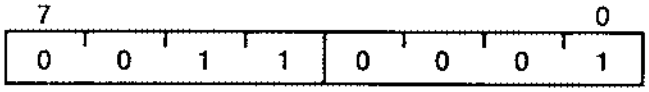
Flag Bits Affected:
SK
L0 ← 0
L1 ← 0

Example:
```
MVI     V,F0H     ;If bit 3 of the contents of
BIT     3,0FFH    ;address F0FFH is high, the JR
JR      $+1       ;instruction is skipped and the
RET               ;RET instruction is executed.
```

BLOCK

Block data transfer.

((DE)) ← ((HL)), (DE) ← (DE) + 1, (HL) ← (HL) + 1, (C) ← (C) − 1, end when decrementing C creates a borrow.



Instruction Code: 31H

Moves blocks of C + 1 bytes of data, up to 256 bytes. The contents of the source (pointed to by HL) are transferred to the destination (pointed to by DE). The number of bytes to be transferred is the contents of register C + 1. After each transfer, DE and HL are automatically incremented by 1 and C is decremented by 1. Execution ends when C is decremented from 00 to FFH. This instruction may be interrupted and the transfer will continue at the end of the interrupt service routine.

Bytes: 1

T-States: 13(C + 1), (4)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0
CY

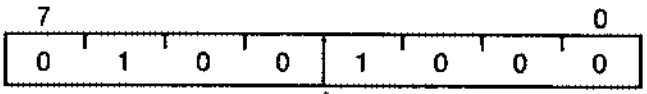
Example:

```
LXI     H,1000H  ;Source
LXI     D,2000H  ;Destination
MVI     C,30     ;Count
BLOCK            ;31 bytes will be transferred
                 ;from an area in memory
                 ;starting at location 1000H to
                 ;an area in memory starting at
                 ;location 2000H.
```

CALB

Call subroutine using BC indirect.

(PC) ← (PC) + 2, ((SP) − 1) ← (PCH), ((SP) − 2) ← (PCL), (SP) ← (SP) − 2, (PC) ← (BC)



Instruction Code: 4829H

Calls a subroutine whose starting address is in register pair BC. Saves the address of the next instruction on the stack. Transfers contents of BC into the program counter.

Bytes: 2

T-States: 17 (8)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

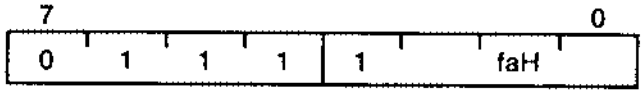
Example:

```
LXI     B,300H   ;The address we want to go to.
CALB             ;This causes the program to
                 ;go to 300H.
```

CALF word

Call subroutine in fixed area.

(PC) ← (PC) + 2, ((SP) − 1) ← (PCH), ((SP) − 2) ← (PCL), (SP) ← (SP) − 2, (PC_{15}-PC_{11}) ← 00001, (PC_{10}-PC_8 ← faH, (PC_7-PC_0) ← faL



Instruction Code: 7(8-F)(0-F)(0-F)H

Executes a subroutine located in the second 2K of memory. Causes the address of the next instruction to be pushed on the stack. The value 00001B is written over the high five bits of the program counter. The 11 bits of immediate data (faH and faL) form the lower 11 bits of the address that will be executed next. The address should be in the range of 0800H to 0FFFH.

Bytes: 2

T-States: 13 (7)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

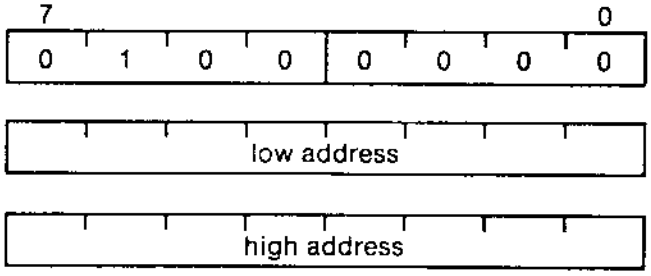
Example:

```
CALF    100H     ;Calls subroutine at 900H.
```

CALL word

Call subroutine direct.

(PC) ← (PC) + 3, ((SP) − 1) ← (PCH), ((SP) − 2) ← (PCL), (SP) ← (SP) − 2, (PC) ← call address



Instruction Code: 40H, low, high

Adds 3 to the program counter to get the address of the next instruction. Pushes the contents of the program counter onto the top of the stack. The operands specified by low and high addresses in the CALL instruction are then loaded into the PC to point to the address in memory where the first opcode of a subroutine is to be fetched.

Bytes: 3

T-States: 16 (10)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

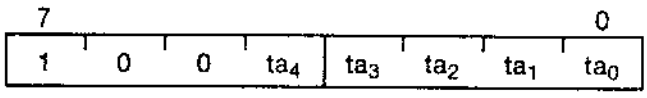
Example:

```
CALL    1234H    ;Causes the subroutine at
                 ;1234H to be executed.
```

CALT byte

Call table address.

(PC) ← (PC) + 1, ((SP) − 1) ← (PCH), ((SP) − 2) ← (PCL), (SP) ← (SP) − 2, (PCH) ← (129 + 2ta), (PCL) ← (128 + 2ta)



Instruction Code: (8-9)(0-F)H

Causes the address of the next instruction to be pushed on the stack and loads a memory address from a table into the PC. PCL is loaded from memory address 128H + 2taH and PCH is loaded from memory address 129H + 2taH. The table resides in memory addresses 128H-191H, and the subroutine address called is two bytes. See "Direct Branching" in Section 11.

Bytes: 1

T-States: 16 (4)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

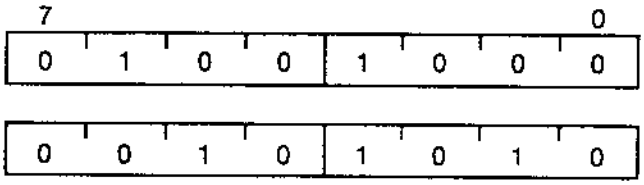
Example:

```
CALT    140      ;Call the subroutine whose
                 ;address is located at 140.
```

CLC

Clear carry.

(CY) ← 0



Instruction Code: 482AH

Clears carry flag.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0
CY ← 0

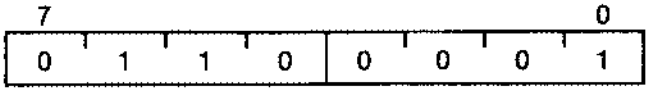
Example:

```
CLC              ;Causes the carry flag to be
                 ;cleared.
```

DAA

Decimal adjust A.

Condition		Operation
$(A_3\text{-}A_0) \leq 9$	$(A_7\text{-}A_4) \leq 9$ and (CY) = 0	(A) ← (A)
(HC) = 0	$(A_7\text{-}A_4) \geq 10$ or (CY) = 1	(A) ← (A) + 60H
$(A_3\text{-}A_0) \geq 10$	$(A_7\text{-}A_4) \leq 8$ and (CY) = 0	(A) ← (A) + 06H
(HC) = 0	$(A_7\text{-}A_4) \geq 9$ or (CY) = 1	(A) ← (A) + 66H
$(A_3\text{-}A_0) \leq 2$	$(A_7\text{-}A_4) \leq 9$ and (CY) = 0	(A) ← (A) + 06H
(HC) = 1	$(A_7\text{-}A_4) \geq 10$ or (CY) = 1	(A) ← (A) + 66H



Instruction Code: 61H

Converts the value in the accumulator from binary to binary-coded decimal. Judges the contents of the accumulator, carry and half-carry, and then decimal revises the accumulator according to the table above. This instruction has meaning only after executing an arithmetic operation.

Bytes: 1

T-States: 4 (4)

Flag Bits Affected:
- Z
- SK ← 0
- HC
- L0 ← 0
- L1 ← 0
- CY

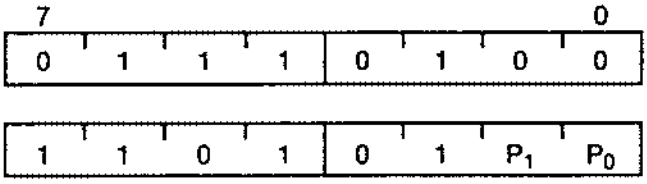
Example:

```
MVI     A,64
ADI     A,19    ;Accumulator contents = 7D.
DAA             ;Accumulator contents
                ;now = 83.
```

DADC EA,rp3

Add contents of register pair to EA with carry.

(EA) ← (EA) + (rp3) + (CY)



Instruction Code: 74D(5-7)H

Adds the CY flag and the contents of register pair rp3 (BC, DE, or HL), designated by P_1-P_0 (1-3), to the extended accumulator and stores the result in the extended accumulator.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
- Z
- SK ← 0
- HC
- L0 ← 0
- L1 ← 0
- CY

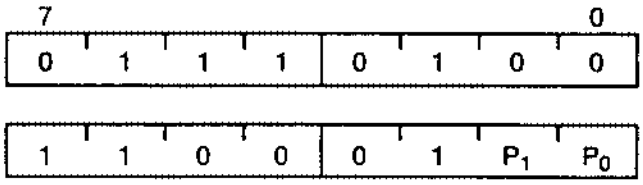
Example:

```
DADC    EA,H    ;Causes the 16-bit value in EA
                ;and 16-bit value in HL to be
                ;added together with the CY
                ;flag. Results are stored in EA.
```

DADD EA,rp3

Add register pair to EA.

(EA) ← (EA) + (rp3)



Instruction Code: 74C(5-7)H

Adds the contents of register pair rp3 (BC, DE, or HL), designated by P_1-P_0 (1-3), to the extended accumulator and stores the result in the extended accumulator.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
- Z
- SK ← 0
- HC
- L0 ← 0
- L1 ← 0
- CY

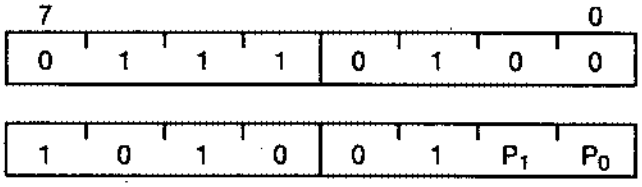
Example:

```
DADD    EA,H
```

DADDNC EA,rp3

Add register pair to EA and skip next instruction if no carry.

(EA) ← (EA) + (rp3), SK/NC



Instruction Code: 74A(5-7)H

Adds the contents of register pair rp3 (BC, DE, or HL), designated by P_1-P_0 (1-3), to the extended accumulator and stores the result in the extended accumulator. If no carry is generated, the next instruction is skipped.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

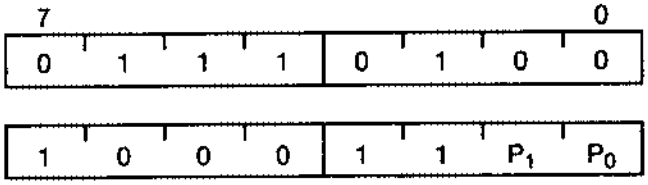
Example:

```
DADDNC  EA,H    ;Causes the 16-bit value in EA
JMP     $+5     ;and the 16-bit value in HL to
RET             ;be added together. The
                ;results are stored in EA. If no
                ;carry is generated, the JMP
                ;will be skipped.
```

DAN EA,rp3

AND register pair with EA.

(EA) ← (EA) AND (rp3)



Instruction Code: 748(D-F)H

Performs logical AND of the contents of register pair rp3 (BC, DE, or HL), designated by P_1-P_0 (1-3), with the extended accumulator. Stores the result in the extended accumulator.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

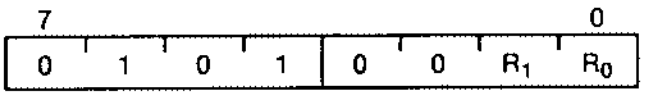
Example:

```
DAN     EA,H    ;Causes the 16-bit value in EA
                ;to be ANDed with the 16-bit
                ;value in HL. The results are
                ;stored in EA.
```

DCR r2

Decrement register and skip next instruction if borrow.

(r2) ← (r2) − 1, SK/B



Instruction Code: 5(1-3)H

Decrements by 1 the contents of register r2 (A, B, or C) designated by R_1-R_0 (1-3). If a borrow is generated (when (r2) goes from 00H to FFH) by the decrement, the next instruction is skipped.

Bytes: 1

T-States: 4 (4)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

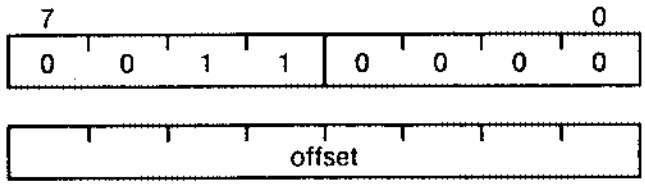
Example:

```
DCR     H       ;Causes the 16-bit value in HL
                ;to be decremented. The MOV
                ;instruction is skipped on
                ;borrow.
MOV     A,C
```

DCRW wa

Decrement working register and skip next instruction if borrow.

((V) • wa) ← ((V) • wa) − 1, SK/B



Instruction Code: 30H, offset

Decrements by 1 the contents of the working register. If a borrow is generated (when ((V) • wa) goes from 00H to FFH), then the next instruction is skipped.

Bytes: 2

T-States: 16 (7)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

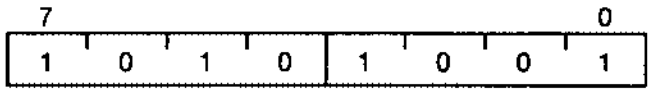
Example:

```
MVI     V,0FH
DCRW    20H     ;Decrements contents of
                ;location 0F20H. The MVI
                ;instruction is skipped on
                ;borrow.
MVI     A,10H
```

DCX EA

Decrement EA.

(EA) ← (EA) − 1



Instruction Code: A9H

Decrement EA by 1.

Bytes: 1

T-States: 7 (4)

Flag Bits Affected:

SK ← 0
L0 ← 0
L1 ← 0

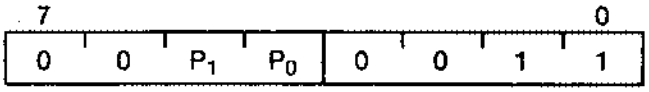
Example:

```
DCX     EA
```

DCX rp

Decrement register pair.

(rp) ← (rp) − 1



Instruction Code: (0-3)3H

Decrements by 1 a 16-bit register pair rp (SP, BC, DE, or HL). P_1-P_0 (0-3) select the register pair.

Bytes: 1

T-States: 7 (4)

Flag Bits Affected:

SK ← 0
L0 ← 0
L1 ← 0

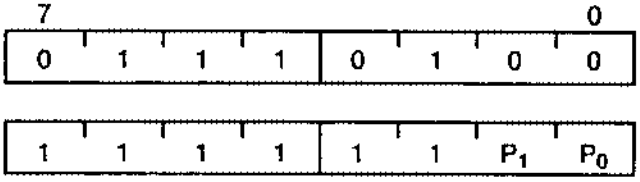
Example:

```
DCX     H
```

DEQ EA,rp3

Skip next instruction if register pair equals EA.

(EA) − (rp3), SK/(EA) = (rp3)



Instruction Code: 74F(D-F)H

Subtracts the contents of register pair rp3 (BC, DE, or HL), designated by P_1-P_0 (1-3), from the extended accumulator. If the subtraction results in zero, (EA) = (rp3), then the next instruction is skipped. The values of the register pair and the extended accumulator are not affected.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:

Z
HC
SK
L0 ← 0
L1 ← 0
CY

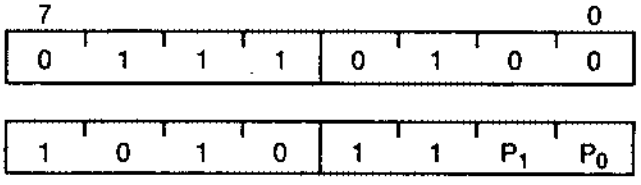
Example:

```
DEQ     EA,BC   ;If BC and EA are equal, the
RET             ;RET instruction is skipped.
```

DGT EA,rp3

Skip next instruction if EA greater than register pair.

(EA) − [(rp3) + 1], SK/(EA) > (rp3)



Instruction Code: 74A(D-F)H

Subtracts the contents plus 1 of register pair rp3 (BC, DE, or HL), designated by P_1-P_0 (1-3), from the extended accumulator. If the subtraction results in no borrow being generated, (EA) > (rp3), the next instruction is skipped. The contents of EA and the register pair are not affected.

Bytes: 2

T-States: 11 (8)

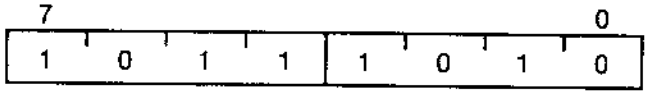
Flag Bits Affected:
Z
HC
SK
L0 ← 0
L1 ← 0
CY

Example:
```
DGT     EA,B    ;If EA is larger than BC, the
RET             ;RET instruction is skipped.
```

DI

Disable interrupt.

Disables maskable interrupts.



Instruction Code: BAH

Inhibits all interrupts except NMI and SOFTI instruction. Until the EI instruction is executed, no more maskable interrupts will be accepted.

Bytes: 1

T-States: 4 (4)

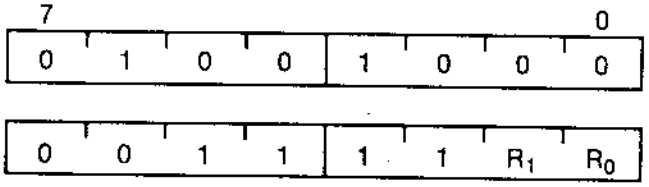
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
DI              ;Turn off all maskable interrupts.
```

DIV r2

Divide EA by register.

(EA) ← (EA)/(r2), r2 ← remainder



Instruction Code: 483(D-F)H

Unsigned division divides the contents of the extended accumulator by that of r2 (A, B, or C) designated by R_1-R_0 (1-3). Stores the quotient in the extended accumulator and the remainder in r2.

Bytes: 2

T-States: 59 (8)

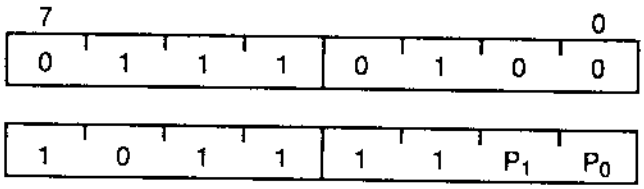
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
DIV     B       ;Divide EA by B.
```

DLT EA,rp3

Skip next instruction if register pair greater than EA.

(EA) − (rp3), SK/(rp3) > (EA)



Instruction Code: 74B(D-F)H

Subtracts the contents of register pair rp3 (BC, DE, or HL), designated by P_1-P_0 (1-3), from the extended accumulator. Skips the next instruction if a borrow is generated, (rp3) > (EA). The contents of EA and the register pair are not affected.

Bytes: 2

T-States: 11 (8)

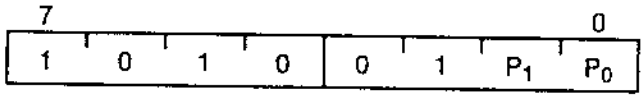
Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

Example:
```
DLT     EA,B    ;EA − BC
RET             ;If BC is larger than EA, the
                ;RET instruction is skipped.
```

DMOV EA,rp3

Move register pair to EA.

(EAL) ← (rp3L), (EAH) ← (rp3H)



Instruction Code: A(5-7)H

Moves the contents of register pair rp3 (BC, DE, or HL), designated by P_1-P_0 (1-3), into the extended accumulator.

Bytes: 1

T-States: 4 (4)

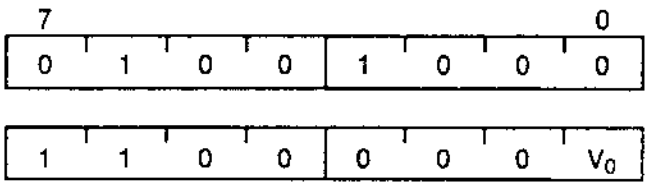
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
LXI     H,1234H  ;Put 1234H in HL.
DMOV    EA,H     ;Copies contents of HL into
                 ;accumulator.
```

DMOV EA,sr4

Move special register to EA.

(EA) ← (sr4)



Instruction Code: 48C(0, 1)H

Moves the contents of sr4 (ECNT or ECPT) designated by V_0 (0-1) into the extended accumulator.

Bytes: 2

T-States: 14 (8)

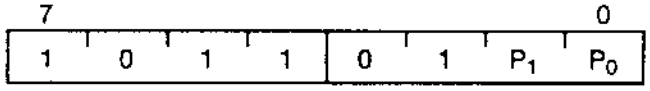
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
DMOV    EA,ECNT  ;Moves the value of ECNT
                 ;into EA.
```

DMOV rp3,EA

Move EA to register pair.

(rp3L) ← (EAL), (rp3H) ← (EAH)



Instruction Code: B(5-7)H

Moves the contents of the extended accumulator into register pair rp3(BC, DE, or HL), designated by P_1-P_0 (1-3).

Bytes: 1

T-States: 4 (4)

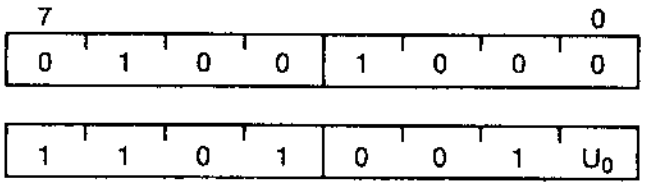
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
LXI     EA,1234H ;Moves 1234H to register
DMOV    H,EA     ;pair HL.
```

DMOV sr3,EA

Move EA to special register.

(sr3) ← (EA)



Instruction Code: 48D(2,3)H

Transfers the contents of EA into sr3 (ETM0 or ETM1), designated by U_0 (0-1).

Bytes: 2

T-States: 14 (8)

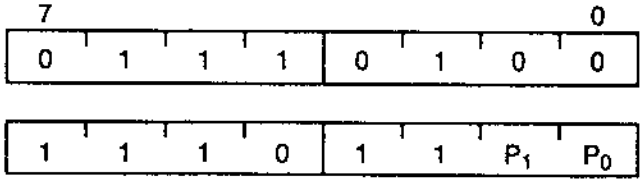
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
DMOV    ETM0,EA  ;Moves the value of EA into
                 ;ETM0.
```

DNE EA,rp3

Skip next instruction if register pair not equal to EA.

(EA) − (rp3), SK/(EA) ≠ (rp3)



Instruction Code: 74E(D-F)H

Subtracts the contents of register pair rp3 (BC, DE, HL), designated by P_1-P_0 (1-3), from the extended accumulator. If the subtraction is not zero, (EA) ≠ (rp3), then the next instruction is skipped. The contents of the extended accumulator and the register pair are not affected.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
HC
SK
L0 ← 0
L1 ← 0
CY

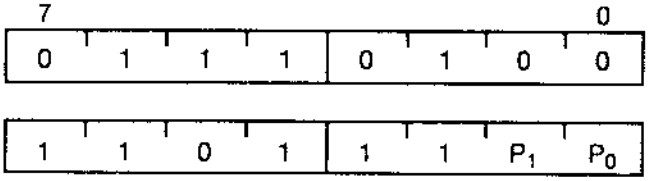
Example:

```
DNE     EA,BC   ;If BC and EA are not equal,
RET             ;the RET instruction is
                ;skipped.
```

DOFF EA,rp3

Skip next instruction if off-test register pair with EA is zero.

(EA) AND (rp3), SK/Z



Instruction Code: 74D(D-F)H

Performs a 16-bit logical AND between the extended accumulator and the register pair rp3 (BC, DE, or HL), designated by P_1-P_0 (1-3). If the logical AND results in zero, the next instruction is skipped. The contents of the extended accumulator and the register pair are not affected.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
SK
L0 ← 0
L1 ← 0

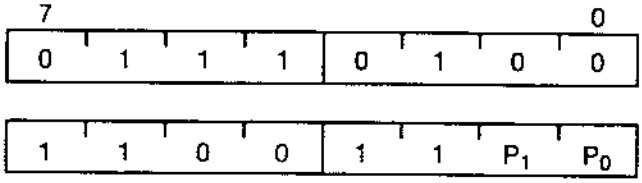
Example:

```
DOFF    EA,BC   ;If the logical AND between
RET             ;BC and EA equals 0, the RET
                ;instruction is skipped.
```

DON EA,rp3

Skip next instruction if on-test register pair with EA is not zero.

(EA) AND (rp3), SK/NZ



Instruction Code: 74C(D-F)H

Performs a 16-bit logical AND between the extended accumulator and register pair rp3 (BC, DE, or HL), designated by P_1-P_0 (1-3). If the logical AND is not zero, then the next instruction is skipped. The contents of the extended accumulator and register pair are not affected.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
SK
L0 ← 0
L1 ← 0

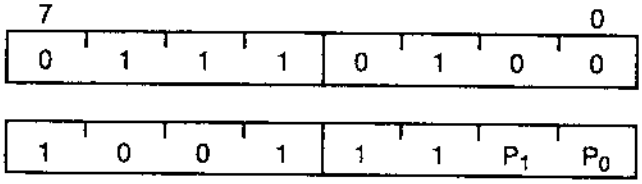
Example:

```
DON     EA,BC   ;If the logical AND between
RET             ;BC and EA is not equal to 0,
                ;the RET instruction is
                ;skipped.
```

DOR EA,rp3

OR register pair with EA.

(EA) ← (EA) OR (rp3)



Instruction Code: 749(D-F)H

Performs a 16-bit logical OR between the extended accumulator and register pair rp3 (BC, DE, or HL) designated by P_1-P_0 (1-3). The result is left in the extended accumulator.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

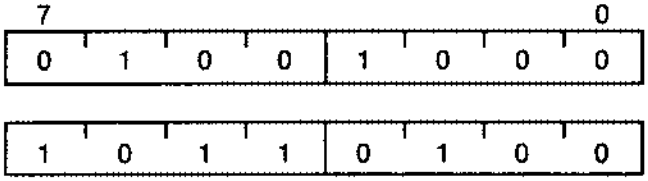
Example:

```
DOR     EA,BC   ;Do a logical OR between EA
                ;and BC.
```

DRLL EA

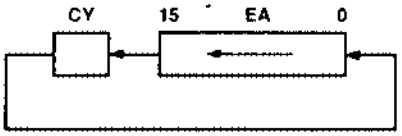
Rotate EA logical left one with carry.

$(EA_{n+1}) \leftarrow (EA_n)$, $(EA_0) \leftarrow (CY)$, $(CY) \leftarrow (EA_{15})$



Instruction Code: 48B4H

Performs a 16-bit rotate left by 1 of the extended accumulator through the CY bit.



Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0
CY

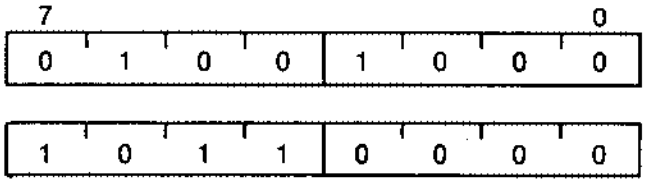
Example:

```
DRLL    EA
```

DRLR EA

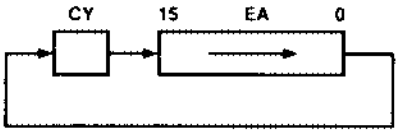
Rotate EA logical right one with carry.

$(EA_{n-1}) \leftarrow (EA_n)$, $(EA_{15}) \leftarrow (CY)$, $(CY) \leftarrow (EA_0)$



Instruction Code: 48B0H

Performs a 16-bit rotate right by 1 of the extended accumulator through the CY bit.



Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0
CY

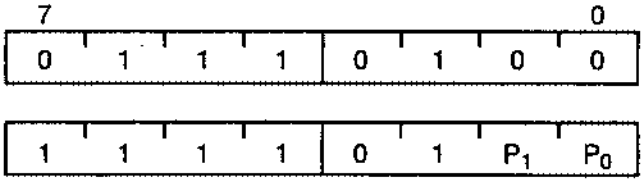
Example:

```
DRLR    EA
```

DSBB EA,rp3

Subtract register pair from EA with borrow.

(EA) ← (EA) − (rp3) − (CY)



Instruction Code: 74F(5-7)H

Subtracts the contents of register pair rp3 (BC, DE, or HL), designated by P_1-P_0 (1-3), from the extended accumulator including the CY flag, and stores the result in the extended accumulator.

Bytes: 2

T-States: 11 (8)

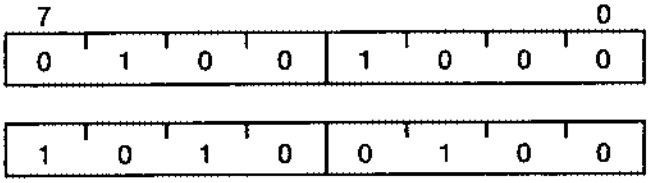
Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

Example:
DSBB EA,H

DSLL EA

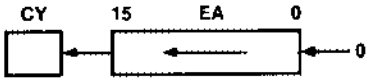
Shift EA logical left one into carry.

$(EA_{n+1}) \leftarrow (EA_n)$, $(EA_0) \leftarrow 0$, $(CY) \leftarrow (EA_{15})$



Instruction Code: 48A4H

Performs a 16-bit shift left by 1 of the extended accumulator into the CY. A zero is shifted into the LSB.



Bytes: 2

T-States: 8 (8)

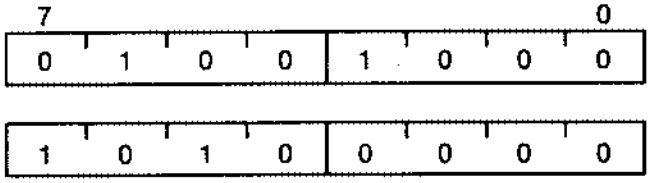
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0
CY

Example:
DSLL EA

DSLR EA

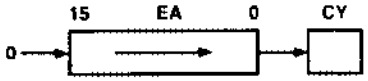
Shift EA logical right one into carry.

$(EA_{n-1}) \leftarrow (EA_n)$, $(EA_{15}) \leftarrow 0$, $(CY) \leftarrow (EA_0)$



Instruction Code: 48A0H

Performs a 16-bit shift right by 1 of the extended accumulator into the CY. A zero is shifted into the MSB.



Bytes: 2

T-States: 8 (8)

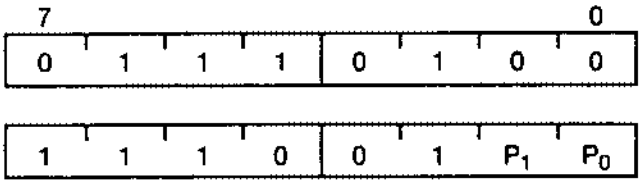
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0
CY

Example:
DSLR EA

DSUB EA,rp3

Subtract register pair from EA.

$(EA) \leftarrow (EA) - (rp3)$



Instruction Code: 74E(5-7)H

Subtracts the contents of register pair rp3 (BC, DE, or HL), designated by P_1-P_0 (1-3), from the extended accumulator and stores the result in the extended accumulator.

Bytes: 2

T-States: 11 (8)

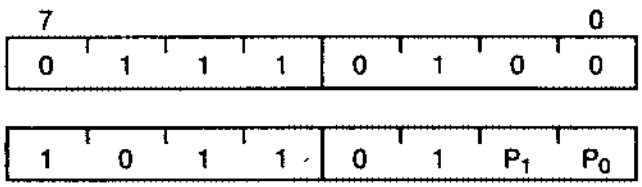
Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

Example:
DSUB EA,H

DSUBNB EA,rp3

Subtract register pair from EA and skip next instruction if EA greater than or equal to register pair.

$(EA) \leftarrow (EA) - (rp3)$, $SK/EA \geq (rp3)$



Instruction Code: 74B(5-7)H

Subtracts the contents of register pair rp3 (BC, DE, or HL), designated by P_1-P_0 (1-3), from the extended accumulator and stores the result in the extended accumulator. If no borrow is generated, (EA) ≥ (rp3), the next instruction is skipped.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
- Z
- SK
- HC
- L0 ← 0
- L1 ← 0
- CY

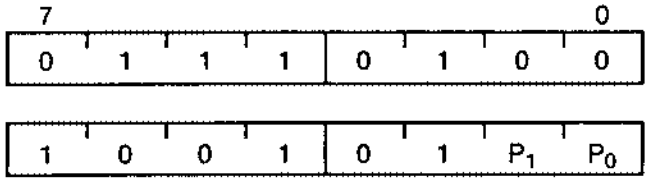
Example:

```
DSUBNB  EA,H    ;Subtract (HL) from (EA).
MVI     A,10H   ;Skip MVI instruction on
                ;borrow.
```

DXR EA,rp3

Exclusive-OR register pair with EA.

(EA) ← (EA) XOR (rp3)



Instruction Code: 749(5-7)H

Performs a 16-bit exclusive-OR of the contents of register pair rp3 (BC, DE, or HL), designated by P_1-P_0 (1-3), with the extended accumulator and stores the result in the extended accumulator.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
- Z
- SK ← 0
- L0 ← 0
- L1 ← 0

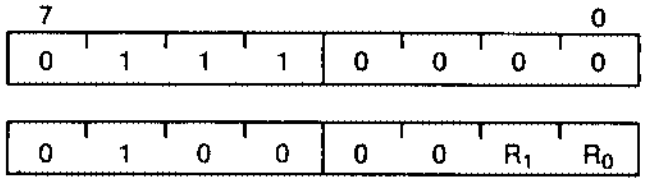
Example:

```
DXR     EA,H
```

EADD EA,r2

Add register to EA.

(EA) ← (EA) + (r2)



Instruction Code: 704(1-3)H

Adds the contents of register r2 (A, B, or C), designated by R_1-R_0 (1-3), to the lower 8 bits of the extended accumulator and stores the result in the extended accumulator.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
- Z
- SK ← 0
- HC
- L0 ← 0
- L1 ← 0
- CY

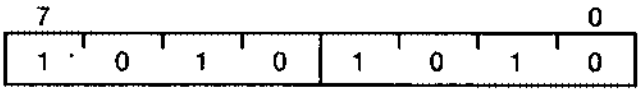
Example:

```
EADD    EA,B
```

EI

Enable interrupt.

Turn on maskable interrupts.



Instruction Code: AAH

Enables all unmasked interrupts. After the EI instruction is executed, no interrupts will be accepted until the next instruction is executed. Therefore, if EI is the last instruction preceding a return from interrupt, the return will be executed before another interrupt is accepted and the registers restored. This saves stack space. The NMI and SOFTI will always be recognized independent of DI or EI.

Bytes: 1

T-States: 4 (4)

Flag Bits Affected:
- SK ← 0
- L0 ← 0
- L1 ← 0

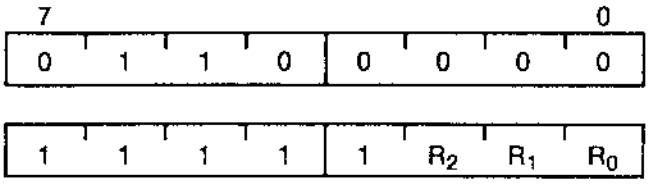
Example:

```
EI
```

EQA A,r

Skip next instruction if register equal to accumulator.

(A) − (r), SK/EQ



Instruction Code: 60F(8-F)H

Subtracts the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), from the accumulator. If the result is zero, (A) = (r), the next instruction is skipped. The contents of the accumulator and register are not affected.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
- Z
- SK
- HC
- L0 ← 0
- L1 ← 0
- CY

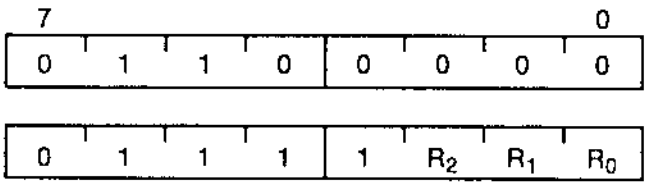
Example:

```
MVI     A,0FFH
EQA     A,C     ;If register C = FF, then the
RET             ;RET is skipped.
```

EQA r,A

Skip next instruction if accumulator equal to register.

(r) − (A), SK/EQ



Instruction Code: 607(8-F)H

Subtracts the contents of the accumulator from r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7). If the result is zero, (r) = A), then the next instruction is skipped. Neither the accumulator nor the register contents are affected.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
- Z
- SK
- HC
- L0 ← 0
- L1 ← 0
- CY

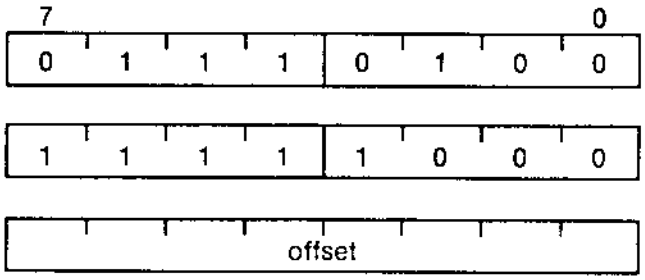
Example:

```
EQA     C,A     ;Compare A with C.
RET             ;Skip this instruction if A = C.
```

EQAW wa

Skip next instruction if working register equal to accumulator.

(A) − ((V) • wa), SK/EQ



Instruction Code: 74F8H, offset

Subtracts the contents of a working register to be addressed by the V-register (8-bit, high-order address) and the offset (8-bit, low-order address) from the accumulator. If the result is zero, (A) = ((V) • wa), the next instruction is skipped. Neither the accumulator nor the working register contents are affected.

Bytes: 3

T-States: 14 (11)

Flag Bits Affected:
- Z
- SK
- HC
- L0 ← 0
- L1 ← 0
- CY

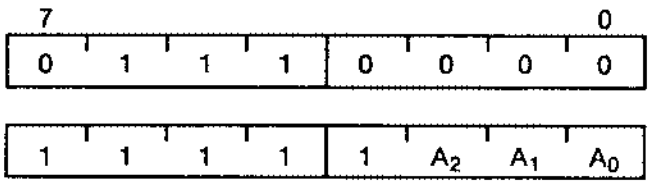
Example:

```
MVI     V,0FH
EQAW    10H     ;Compare the contents of
RET             ;location 0F10H with the
                ;accumulator. Skip this
                ;instruction if (A) = (0F10H).
```

EQAX rpa

Skip next instruction if memory addressed by register pair equal to accumulator.

(A) − ((rpa)), SK/EQ



Instruction Code: 70F(9-F)H

Subtracts the contents of memory addressed by register pair rpa (BC, DE, HL, DE+, HL+, DE−, or HL−), designated by A_2-A_0 (1-7), from the accumulator. If the result is zero, (A) = (rpa), the next instruction will be skipped. Neither the accumulator nor the memory contents are affected.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

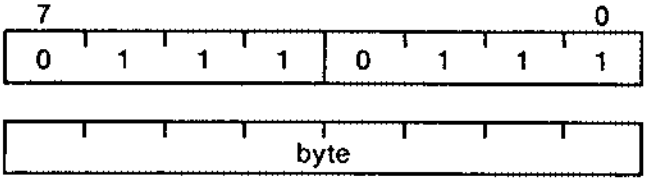
Example:

```
EQAX    D       ;Compare memory pointed to
RET             ;by D with the accumulator.
                ;Skip this instruction if the
                ;result is zero ((A) = (memory)).
```

EQI A,byte

Skip next instruction if immediate data byte equal to accumulator.

(A) − byte, SK/EQ



Instruction Code: 77H, byte

Subtracts the immediate data byte from the contents of the accumulator. If the result is zero, (A) = data byte, the next instruction is skipped. The contents of the accumulator are not affected.

Bytes: 2

T-States: 7 (7)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

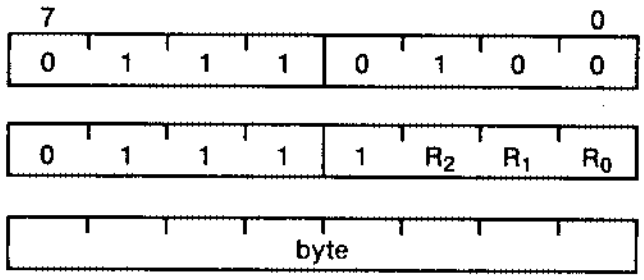
Example:

```
EQI     A,20H   ;Compare 20H with
                ;accumulator.
RET             ;Skip this instruction if
                ;(A) = 20H.
```

EQI r,byte

Skip next instruction if immediate data byte equal to register.

(r) − byte, SK/EQ



Instruction Code: 747(8-F)H, byte

Subtracts the immediate data byte from the contents of a register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), from the accumulator. Skips the next instruction if the result is zero, (r) = data byte. The contents of the register are not affected.

Bytes: 3

T-States: 11 (11)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

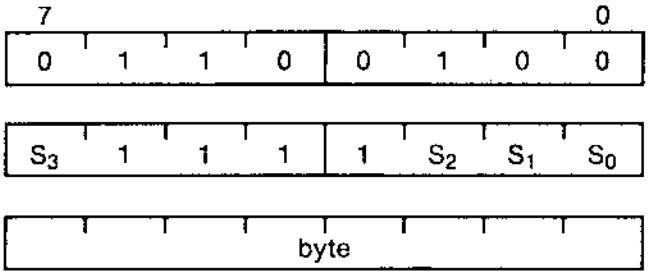
Example:

```
EQI     V,20H   ;Compare 20H with V-register.
RET             ;Skip this instruction if
                ;(V) = 20H.
```

EQI sr2,byte

Skip next instruction if immediate data byte equal to special register.

(sr2) − byte, SK/EQ



Instruction Code:
647(8-B, D-F)H, byte
64F(8, 9, B, D))H, byte

Subtracts the immediate data byte from the contents of special register sr2 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, or TMM) designated by S_3-S_0 (0-3, 5-9, B, D). If the result is zero, (sr2) = data byte, the next instruction is skipped. The contents of the special register are not affected.

Bytes: 3

T-States: 14 (11)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

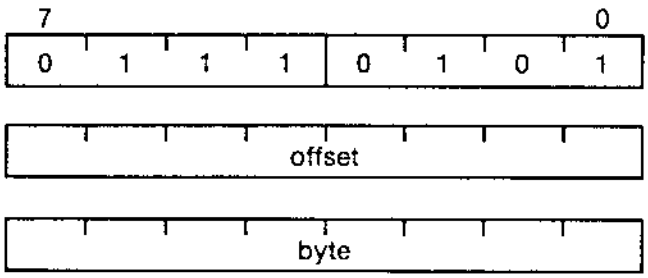
Example:

```
EQI     PD,20H   ;Compare 20H with PD.
RET              ;Skip this instruction if
                 ;(PD) = 20H.
```

EQIW wa,byte

Skip next instruction if immediate data byte equal to working register.

((V) • wa) − byte, SK/EQ



Instruction Code: 75H, offset, byte

Subtracts the immediate data byte from the contents of the working register addressed by the V-register (8 high-order address bits) and the offset (8 lower-order address bits). If the result is zero, ((V) • wa) = data byte, the next instruction is skipped. The contents of the working register are not affected.

Bytes: 3

T-States: 13 (10)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

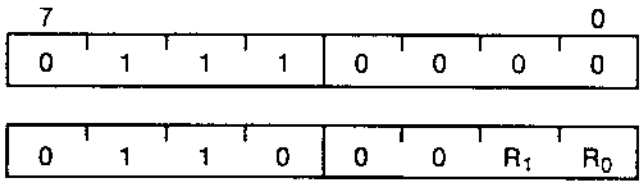
Example:

```
MVI     V,80H
EQIW    10H,20H  ;Compare 20H with contents
RET              ;of location 8010H. Skip RET if
                 ;the result is zero.
```

ESUB EA,r2

Subtract register from EA.

(EA) ← (EA) − (r2)



Instruction Code: 706(1-3)H

Subtracts the contents of register r2 (A, B, or C), designated by R_1-R_0 (1-3), from the extended accumulator and stores the result in the extended accumulator.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

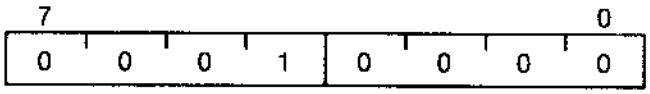
Example:

```
ESUB    EA,B
```

EXA

Exchange V, A, and EA with V', A', and EA'.

(V) ⟷ (V'), (A) ⟷ (A'), (EA) ⟷ (EA')



Instruction Code: 10H

Exchanges register V with V', A with A', EA with EA'.

Bytes: 1

T-States: 4 (4)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

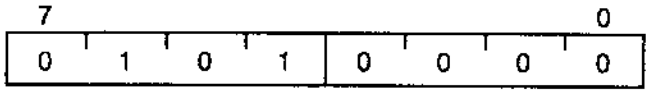
Example:

```
EXA          ;Exchange registers V, A, and
             ;EA with V', A', and EA'.
```

EXH

Exchange HL with H'L'.

(HL) ⟷ (H'L')



Instruction Code: 50H

Exchanges register pair HL with H'L'.

Bytes: 1

T-States: 4 (4)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

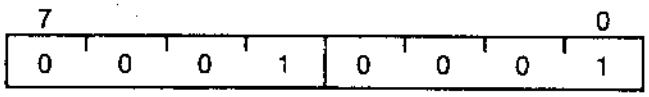
Example:

```
EXH          ;Exchange registers HL and H'L'.
```

EXX

Exchange register sets.

(B) ⟷ (B'), (C) ⟷ (C'), (D) ⟷ (D'),
(E) ⟷ (E'), (H) ⟷ (H'), (L) ⟷ (L')



Instruction Code: 11H

Exchanges the contents of registers B, C, D, E, H, and L with alternate registers B', C', D', E', H', and L'.

Bytes: 1

T-States: 4 (4)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

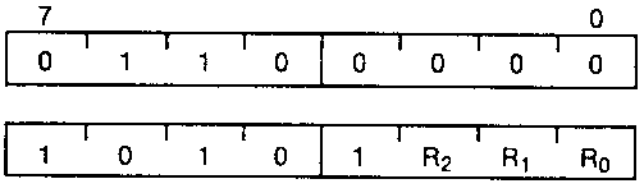
Example:

```
EXX          ;Exchange registers B, C, D,
             ;E, H, and L with B', C', D', E',
             ;H', and L'.
```

GTA A,r

Skip next instruction if A greater than register.

(A) − [(r) + 1], SK/(A) > (r)



Instruction Code: 60A(8-F)H

Subtracts the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), plus 1 from the accumulator. If no borrow is generated by the subtraction, (A) > (r), the next instruction is skipped. Does not affect the contents of the accumulator or the register.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

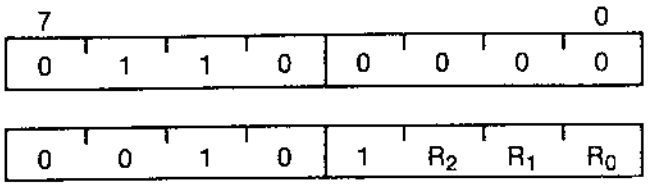
Example:

```
GTA     A,C  ;Compare A with C.
RET          ;This instruction is skipped if
             ;A > C.
```

GTA r,A

Skip next instruction if register greater than A.

(r) − [(A) + 1], SK/(r) > (A)



Instruction Code: 602(8-F)H

Subtracts the contents of the accumulator plus 1 from the contents of register r (V, A, B, C, D, E, H, or L) designated by R_2-R_0 (0-7). If no borrow is generated, (r) > (A), the next instruction is skipped. Does not affect the contents of the accumulator or the register.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
- Z
- SK
- HC
- L0 ← 0
- L1 ← 0
- CY

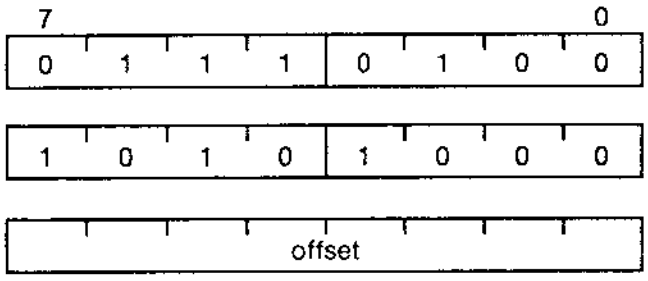
Example:

```
GTA     C,A     ;Subtract A from C.
RET             ;If C is greater than A, this
                ;instruction is skipped.
```

GTAW wa

Skip next instruction if A greater than working register.

(A) − [((V) • wa) + 1], SK/(A) > ((V) • wa)



Instruction Code: 74A8H, offset

Subtracts the contents of a working register to be addressed by the V-register (8-bit high-order address) and the offset (8-bit low-order address) plus 1 from the accumulator. If no borrow is generated, (A) > ((V) • wa), the next instruction is skipped. Does not affect the accumulator or the working register.

Bytes: 3

T-States: 14 (11)

Flag Bits Affected:
- Z
- SK ← 0
- HC
- L0 ← 0
- L1 ← 0
- CY

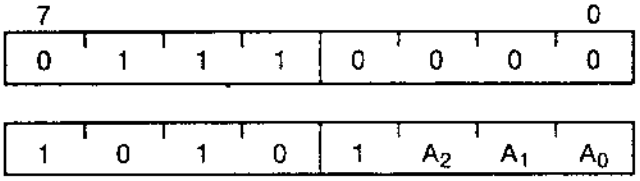
Example:

```
MVI     V,0FH
GTAW    10H     ;Compare the contents of
RET             ;location 0F10H with the
                ;accumulator. Skip RET if no
                ;borrow generated (A > V).
```

GTAX rpa

Skip next instruction if accumulator greater than memory addressed by register pair.

(A) − [((rpa)) + 1], SK/(A) > MEM



Instruction Code: 70A(9-F)H

Subtracts the contents of the memory location that is addressed by register pair rpa (BC, DE, HL, DE+, HL+, DE−, or HL−), designated by A_2-A_0 (1-7), plus 1 from the accumulator. The next instruction will be skipped if no borrow, (A) > memory location, is generated. Does not affect the contents of accumulator or memory.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

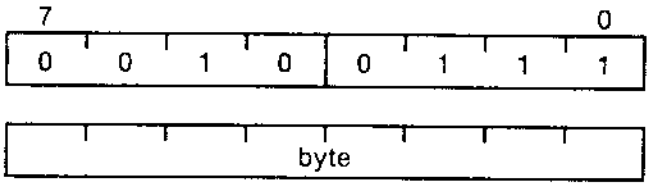
Example:

```
GTAX    D       ;Compare memory pointed to
                ;by DE with accumulator.
RET             ;This instruction is skipped if
                ;no borrow is generated
                ;((A) > ((DE))).
```

GTI A,byte

Skip next instruction if accumulator greater than immediate data byte.

(A) − [byte + 1], SK/(A) > byte



Instruction Code: 27H, byte

Subtracts the immediate data byte plus 1 from the accumulator. If no borrow is generated, (A) > data byte, then the next instruction is skipped. Does not affect the contents of the accumulator.

Bytes: 2

T-States: 7 (7)

Flag Bits Affected:
Z
HC
SK
L0 ← 0
L1 ← 0
CY

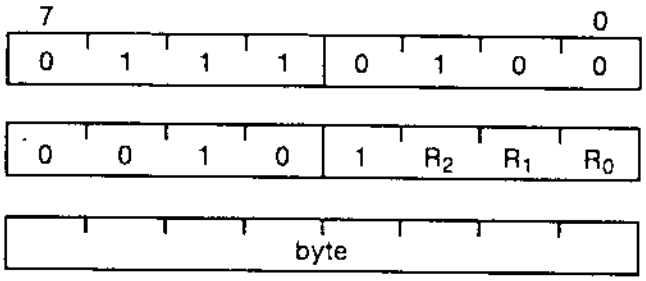
Example:

```
GTI     A,20H
RET             ;Skip this instruction if A >
                ;20H.
```

GTI r,byte

Skip next instruction if register greater than immediate data byte.

(r) − [byte + 1], SK/(r) > byte



Instruction Code: 742(8-F)H, byte

Subtracts the immediate data byte plus 1 from the contents of register r (V, A, B, C, D, E, H, or L) designated by R_2-R_0 (0-7). If no borrow is generated, (r) > data byte, the next instruction is skipped. Does not affect the contents of the register.

Bytes: 3

T-States: 11 (11)

Flag Bits Affected:
Z
HC
SK
L0 ← 0
L1 ← 0
CY

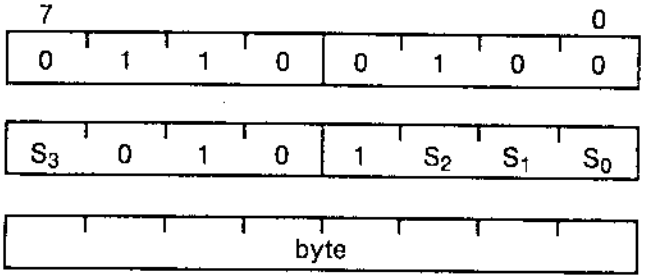
Example:

```
GTI     D,20H
RET             ;Skip this instruction if
                ;D > 20H.
```

GTI sr2,byte

Skip next instruction if special register greater than immediate data byte.

(sr2) − [byte + 1], SK/(sr2) > byte



Instruction Code:
642(8-B, D-F)H, byte
64A(8, 9, B, D)H, byte

Subtract the immediate data byte plus 1 from the contents of special register sr2 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, or TMM), designated by S_3-S_0 (0-3, 5-9, B, D). If no borrow is generated, (sr2) > data byte, skip the next instruction. Does not affect the contents of the register.

Bytes: 3

T-States: 14 (11)

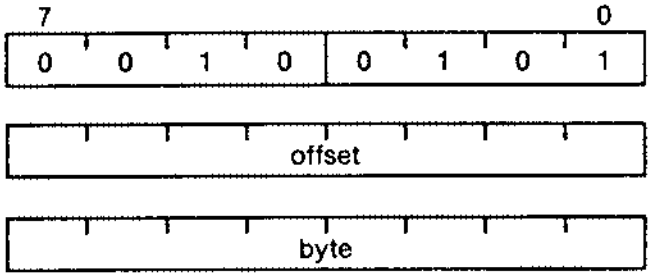
Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

Example:
```
GTI     PD,20H
RET             ;Skip this instruction if port D
                ;> 20H.
```

GTIW wa,byte

Skip next instruction if working register greater than immediate data byte.

((V) • wa) − [byte + 1], SK/((V) • wa) > byte



Instruction Code: 25H, offset, byte

Subtracts the immediate data byte plus 1 from the contents of the working register addressed by the V-register (8 high-order address bits) and the offset (8 lower-order address bits). If no borrow is generated, ((V) • wa) > data byte, the next instruction is skipped. Does not affect the contents of the working register.

Bytes: 3

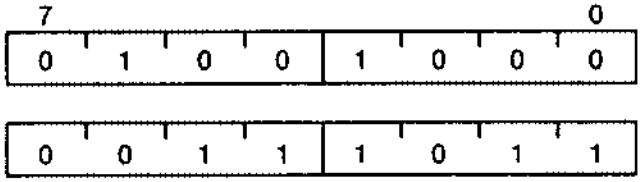
T-States: 13 (10)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

Example:
```
MVI     V,070H
GTIW    10H,20H
RET             ;Skip this instruction if the
                ;contents of location 7010H
                ;> 20H.
```

HLT

Stop processing.



Instruction Code: 483BH

Suspends CPU operation by setting the HALT flip-flop until a subsequent interrupt or reset is received. While in the halt state, the CPU repeats the M3 T2 cycle.

Bytes: 2

T-States:
11 (8), NMOS parts
12 (8), CMOS parts

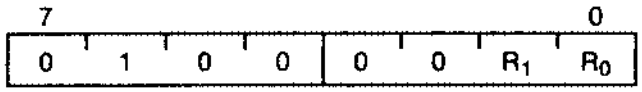
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
HLT             ;Stop all processing until an
                ;interrupt or reset.
```

INR r2

Increment register and skip next instruction if carry.

(r2) ← (r2) + 1, SK/CY



Instruction Code: 4(1-3)H

Increments by 1 the contents of register r2 (A, B, or C) designated by R_1-R_0 (1-3). If a carry is generated by the increment, the next instruction is skipped.

Bytes: 1

T-States: 4 (4)

Flag Bits Affected:
- Z
- SK
- L0 ← 0
- L1 ← 0
- HC
- CY

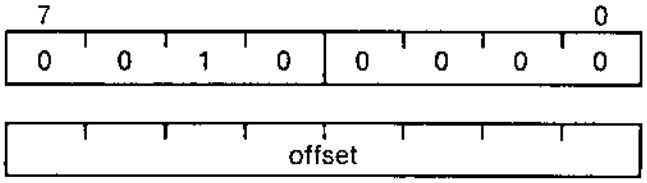
Example:

```
INR     A       ;Causes the value in A to be
                ;incremented by 1.
RET             ;This instruction is skipped if
                ;A increments from 0FFH
                ;to 00H.
```

INRW wa

Increment working register and skip next instruction if carry.

((V) • wa) ← ((V) • wa) + 1, SK/CY



Instruction Code: 20H, offset

Increments by 1 the contents of the working register designated by the V-register (high-order 8 bits) and the offset (low-order 8 bits). If a carry is generated, the next instruction is skipped.

Bytes: 2

T-States: 16 (7)

Flag Bits Affected:
- Z
- SK
- HC
- L0 ← 0
- L1 ← 0
- CY

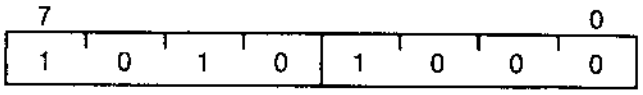
Example:

```
MVI     V,0FFH
INRW    20H     ;Increments location 0FF20H.
RET             ;If a carry is generated, RET
                ;is skipped.
```

INX EA

Increment EA.

(EA) ← (EA) + 1



Instruction Code: A8H

Increment the extended accumulator by 1.

Bytes: 1

T-States: 7 (4)

Flag Bits Affected:
- SK ← 0
- L0 ← 0
- L1 ← 0

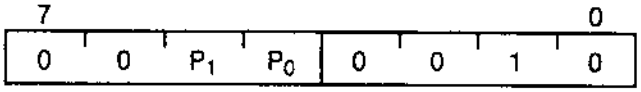
Example:

```
INX     EA      ;Causes EA to be incremented by 1.
```

INX rp

Increment register pair.

(rp) ← (rp) + 1



Instruction Code: (0-3)2H

Increments register pair rp (SP, BC, DE, or HL) by 1. P_1-P_0 (0-3) select the register pair.

Bytes: 1

T-States: 7 (4)

Flag Bits Affected:
- SK ← 0
- L0 ← 0
- L1 ← 0

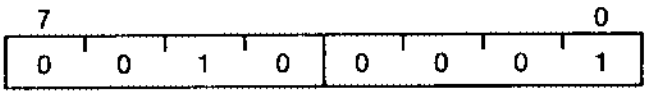
Example:

```
INX     H       ;Causes HL to be incremented by 1.
```

JB

Jump BC indirect.

(PCH) ← (B), (PCL) ← (C)



Instruction Code: 21H

Loads the contents of the BC register pair into the program counter. The BC register pair is not affected and the next opcode fetch will be from the address that is in BC.

Bytes: 1

T-States: 4 (4)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

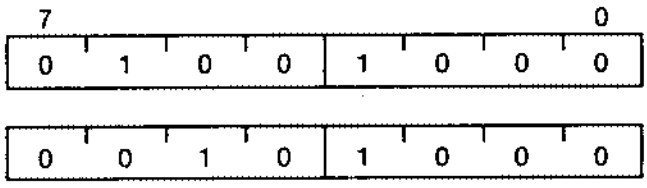
Example:

```
        MVI     A,0H    ;(A) = 0 offset into
                        ;table.
        TABLE           ;(BC) = address from
                        ;table.
        JB              ;Jump to address in BC.
TBO:    DW      WORD    ;Table addresses start
                        ;here.
```

JEA

Jump EA indirect.

(PC) ← (EA)



Instruction Code: 4828H

Loads the contents of the extended accumulator into the program counter. The extended accumulator is not affected and the next opcode fetch will be from the address that is in EA.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

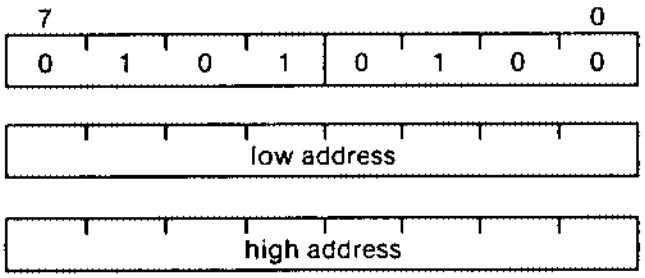
Example:

```
        JEA             ;Go to address in EA.
```

JMP word

Jump direct.

(PC) ← word



Instruction Code: 54H, low, high

Loads the immediate address into the program counter. The first byte of immediate data is loaded into the low byte of the program counter; the second byte is loaded into the high byte of the program counter. The program will start executing from the new address in the PC.

Bytes: 3

T-States: 10 (10)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

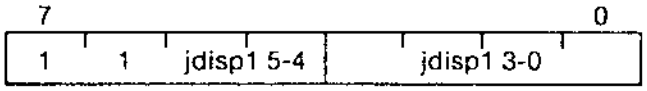
Example:

```
        JMP     NEXT    ;Go to address labeled
                        ;NEXT.
        MVI     A,5     ;This instruction will be
                        ;skipped.
NEXT:   RET             ;This is the next instruction.
```

JR word

Jump relative.

(PC) ← (PC) + 1 + jdisp1



Instruction Code: (C-F)(0-F)H

The contents in the program counter are incremented by 1, and this 6-bit signed displacement is added to the value in the program counter. This jump instruction is a one-byte instruction with a range of +32 to −31 bytes with respect to the current PC value. The jump displacement is added to the program counter, sign extended, causing the next M1 cycle to occur at this address. If the JR instruction is executed at 100, the range of the jump is from 69 to 132. See "Relative Branching" in Section 11.

Bytes: 1

T-States: 10 (4)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

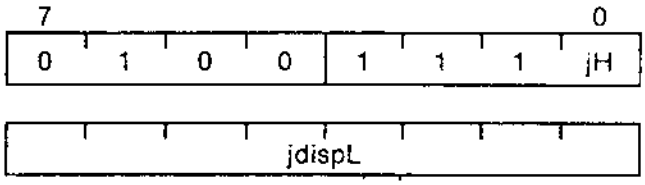
Example: Fill 20H bytes of memory with 30H.

```
CLM:    LXI   D,500H  ;Starting point.
        MVI   C,1FH   ;Amount of memory to
                      ;be filled.
        MVI   A,'0'   ;ASCII 30H.
LOOP:   STAX  D+      ;Put 30H in memory and
                      ;increment DE.
        DCR   C       ;Count off loop count.
        JR    LOOP    ;Loop until finished.
        RET           ;Back to caller all
                      ;finished.
```

JRE word

Jump relative extended.

(PC) ← (PC) + 2 + jdisp



Instruction Code: 4(E, F) (0-F)H

This jump instruction is a two-byte instruction with a range of +255 to −255 bytes. The current value of the program counter is incremented by 2; then, the jump displacement is added to the program counter. The second byte of the instruction is the displacement. The lowest bit of the first byte is the sign. If the JR instruction is executed at 1000, the range of the jump is from 747 to 1257.

Bytes: 2

T-States: 10 (7)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

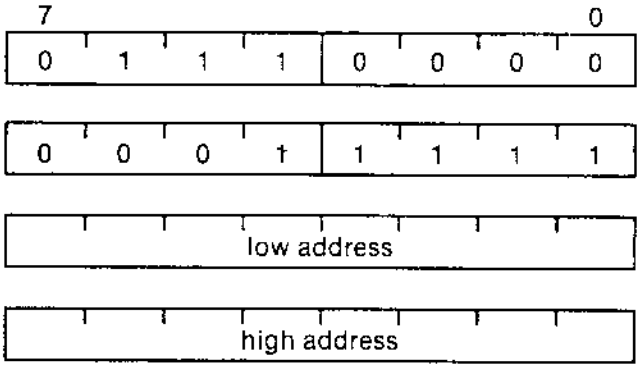
Example:

```
        JRE   ZIPPY   ;Jump around storage
                      ;locations.
        DS    200H    ;Defined storage of 200H
                      ;bytes.
ZIPPY:  MVI   A,3     ;Put 3 in A.
```

LBCD word

Load register pair BC direct.

(C) ← (word), (B) ← (word + 1)



Instruction Code: 701FH, low, high

Loads 16 bits into register pair BC. The contents of C are replaced by the contents of memory at the location pointed to by immediate address. The contents of B are replaced by the contents of memory at the next higher location.

Bytes: 4

T-States: 20 (14)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

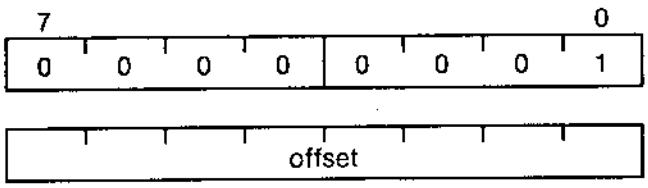
Example:

```
        LBCD  ZIPPY   ;Load BC with the
                      ;contents of ZIPPY.
        RET           ;At this point,
                      ;BC = 1234H.
ZIPPY:  DW    1234H   ;16-bit number.
```

LDAW wa

Load A with working register.

(A) ← ((V) • wa)



Instruction Code: 01H, offset

Loads the accumulator with the contents of the working register. The working register is designated by concatenating the contents of the V-register with the offset.

Bytes: 2

T-States: 10 (7)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

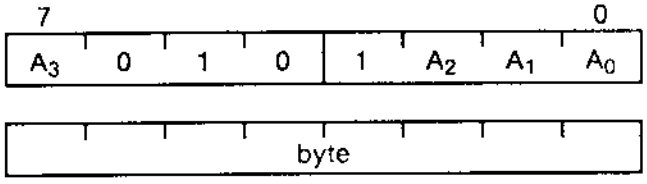
Example:

```
MVI   V,80H   ;Load the accumulator with
LDAW  23H     ;contents of 8023H.
```

LDAX rpa2

Load A with memory addressed by register pair.

(A) ← ((rpa2))



Instruction Code:
2(9-F)H, byte
A(B-F)H, byte

Loads the accumulator with the contents of memory addressed by register pair rpa2 (BC, DE, HL, DE+, HL+, DE−, HL−, DE+byte, HL+A, HL+B, HL+EA, or HL+byte) as designated by A_3-A_0 (1-7, B-F). If auto-increment or decrement is designated, the contents of register pair (DE or HL) are automatically incremented or decremented by 1 after loading the accumulator. If rpa2 is designated with DE+byte or HL+byte, the memory address is the sum of the contents of DE or HL and the data byte of the instruction. Similarly, the contents of HL and that of register A, B, or EA are added if HL+A, HL+B, or HL+EA is designated.

The number of bytes and T-states varies depending on the register pair designated (see following table).

rpa2	B, D, H, D+, H+, D−, H−	D+byte	H+A, H+B, H+EA	H+byte
Bytes	1	2	1	2
T-states	7 (4)	13 (7)	13 (7)	13 (7)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

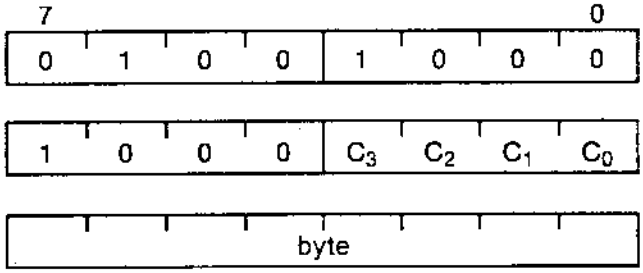
Example: To load memory address 120H to A.

```
EXAMP:  LXI   H,100H   ;(HL) ← 100H
        MVI   B,20H    ;(B) ← 20H
        LDAX  H+B      ;(A) ← (120H)
```

LDEAX rpa3

Load EA with contents of memory addressed by register pair.

(EAL) ← ((rpa3)), (EAH) ← ((rpa3) + 1)



Instruction Code: 488(2-5, B-F)H, byte

Loads the lower-order bits (EAL) of the extended accumulator with the contents of memory addressed by register pair rpa3 (DE, HL, DE++, HL++, DE+byte, HL+A, HL+B, HL+EA, or HL+byte) as designated by C_3-C_0 (2-5, B-F). Loads EAH with the contents addressed by (rpa3) + 1.

If DE+byte or HL+byte is designated as rpa3, the memory is addressed by the sum of the contents of DE or HL and the data byte. If the source memory is addressed by HL+A, HL+B, or HL+EA, memory is addressed by the sum of the contents of HL and register (A, B, or EA).

The number of bytes and T-states varies depending on the register pair designated (see following table).

rpa3	D, H, D++, H++	D+byte	H+A, H+B, H+EA	H+byte
Bytes	2	3	2	3
T-states	14 (8)	20 (11)	20 (11)	20 (11)

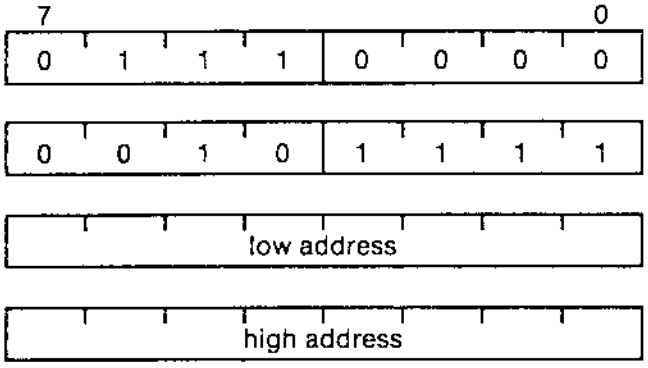
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

Example:
EXAMP: LDEAX HL++

LDED word

Load register pair DE direct.

(E) ← (word), (D) ← (word +1)



Instruction Code: 702FH, low, high

The contents of E are replaced by the contents of memory pointed to by word. The contents of D are replaced by the contents of memory at location word + 1.

Bytes: 4

T-States: 20 (14)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

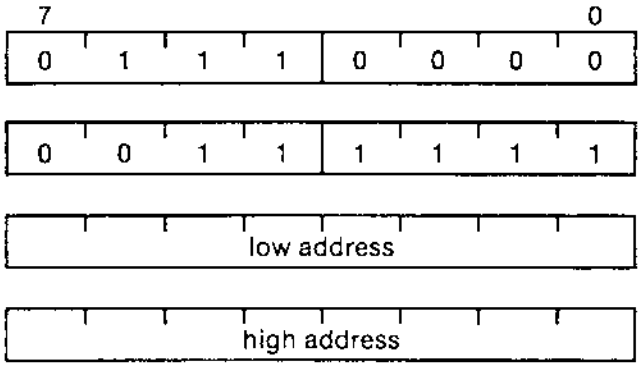
Example:

```
        LDED   ZIPPY    ;Load DE with the
                        ;contents of ZIPPY.
        RET             ;At this point,
                        ;DE = 1234H,
ZIPPY:  DW     1234H    ;just a number.
```

LHLD word

Load register pair HL direct.

(L) ← (word), (H) ← (word) +1)



Instruction Code: 703FH, low, high

The contents of L are replaced by the contents of the memory location pointed to by word. The contents of H are replaced by the contents of memory at location word + 1.

Bytes: 4

T-States: 20 (14)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

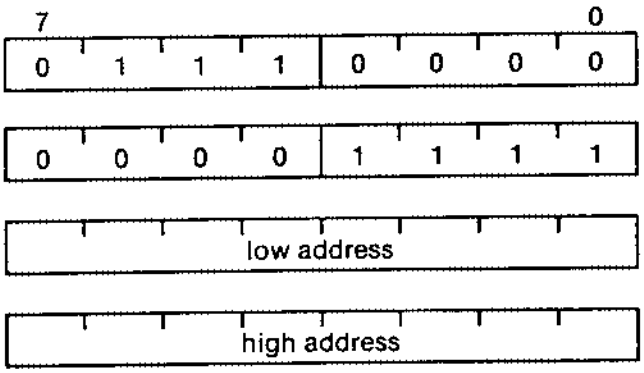
Example:

```
        LHLD   ZIPPY    ;Load HL with the
                        ;contents of ZIPPY.
        RET             ;At this point,
                        ;HL = 1234H,
ZIPPY:  DW     1234H    ;just a number.
```

LSPD word

Load SP register direct.

(SPL) ← (word), (SPH) ← (word + 1)



Instruction Code: 700FH, low, high

The contents of SPL are replaced by the contents of memory at location word and the contents of SPH are replaced by the contents of memory at location word + 1.

Bytes: 4

T-States: 20 (14)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

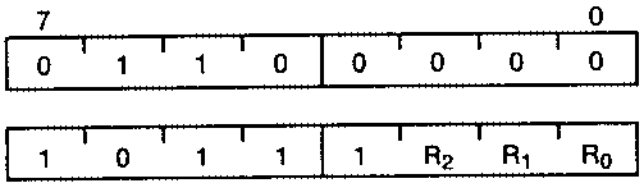
Example:

```
        LSPD  ZIPPY   ;Load SP with the
                      ;contents of ZIPPY.
        RET           ;(SP) = 1234H,
ZIPPY   DW    1234H   ;just a number.
```

LTA A,r

Skip next instruction if register greater than A.

(A) − (r), SK/(r) > (A)



Instruction Code: 60B(8-F)H

Subtracts the contents of register r (V, A, B, C, D, E, H, or L) designated by R_2-R_0 (0-7) from the contents of the accumulator. If a borrow is generated, (r) > (A), the next instruction is skipped. Does not affect the contents of the accumulator or the register.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

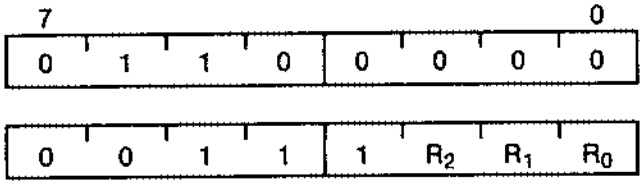
Example:

```
        LTA     A,C
        RET             ;This instruction is skipped if
                        ;C > A.
```

LTA r,A

Skip next instruction if A greater than register.

(r) − (A), SK/(A) > (r)



Instruction Code: 603(8-F)H

Subtracts the contents of the accumulator from the contents of register r (V, A, B, C, D, E, H, or L) designated by R_2-R_0 (0-7). If a borrow is generated by the subtraction, (A) > (r), the next instruction is skipped. The contents of the accumulator and the register are not affected.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

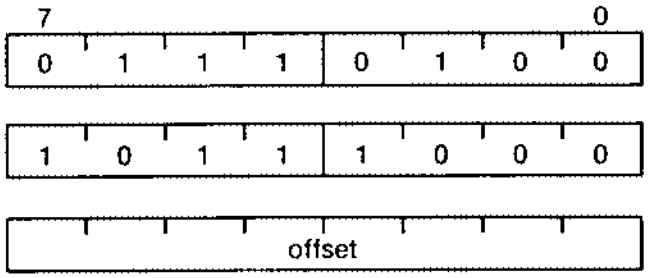
Example:

```
        LTA     C,A
        RET             ;This instruction is skipped if
                        ;A > C.
```

LTAW wa

Skip next instruction if working register greater than A.

(A) − ((V) • wa), SK/((V) • wa) > (A)



Instruction Code: 74B8H, offset

Subtracts the contents of the working register from the accumulator. If a borrow is generated, ((V) • wa) > (A), the next instruction is skipped. Does not affect the contents of the working register or the accumulator.

Bytes: 3

T-States: 14 (11)

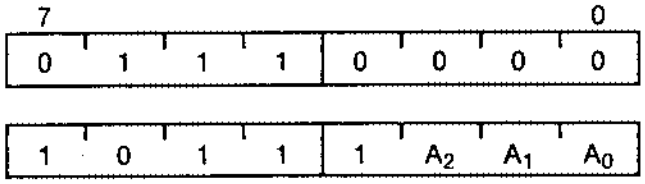
Flag Bits Affected:
- Z
- SK
- HC
- L0 ← 0
- L1 ← 0
- CY

Example:
```
MVI     V,0FH
LTAW    10H
RET             ;Skip this instruction if
                ;(0F10H) > (A).
```

LTAX rpa

Skip next instruction if memory addressed by register pair greater than accumulator.

(A) – ((rpa)), SK/((rpa)) > (A)



Instruction Code: 70B(9-F)H

Subtracts the contents of the memory location that is addressed by register pair rpa (BC, DE, HL, DE+, HL+, DE–, or HL–) designated by A_2-A_0 (1-7) from the accumulator. The next instruction will be skipped if a borrow is generated, ((rpa)) > (A). Does not affect the contents of the accumulator or memory.

Bytes: 2

T-States: 11 (8)

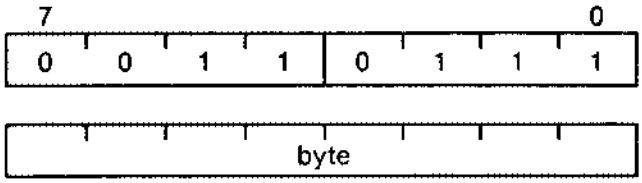
Flag Bits Affected:
- Z
- SK
- HC
- L0 ← 0
- L1 ← 0
- CY

Example:
```
LTAX    D
RET             ;This instruction skipped if
                ;memory greater than A.
```

LTI A,byte

Skip next instruction if immediate data byte greater than A.

(A) – byte, SK/byte > (A)



Instruction Code: 37H, byte

Subtracts the immediate data byte from the accumulator. If a borrow is generated, data byte > (A), then the next instruction is skipped. Does not affect the contents of the accumulator.

Bytes: 2

T-States: 7 (7)

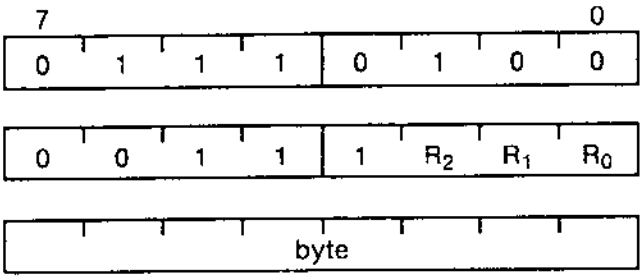
Flag Bits Affected:
- Z
- SK
- HC
- L0 ← 0
- L1 ← 0
- CY

Example:
```
LTI     A,20H
RET             ;Skip this instruction if
                ;20H > A.
```

LTI r,byte

Skip next instruction if immediate data byte greater than register.

(r) − byte, SK/byte > (r)



Instruction Code: 743(8-F)H, byte

Subtracts the immediate data byte from the contents of register r (V, A, B, C, D, E, H, or L) designated by R_2-R_0 (0-7). If a borrow is generated, data byte > (r), then the next instruction is skipped. Does not affect the contents of the register.

Bytes: 3

T-States: 11 (11)

Flag Bits Affected:
- Z
- SK
- HC
- L0 ← 0
- L1 ← 0
- CY

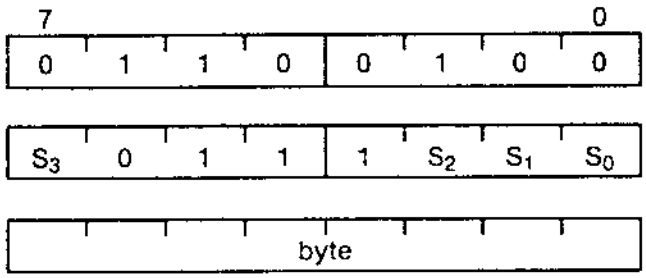
Example:

```
LTI     D,20H   ;Compare (D) with 20H.
RET             ;Skip this instruction if a
                ;borrow (D < 20H).
```

LTI sr2,byte

Skip next instruction if immediate data byte greater than special register.

(sr2) − byte, SK/byte > (sr2)



Instruction Code:
643(8-B, D-F)H, byte
648(8, 9, B, D)H, byte

Subtracts the immediate data byte from the contents of special register sr2 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, or TMM) designated by S_3-S_0 (0-3, 5-9, B, D). If a borrow is generated, data byte > (sr2), then the next instruction is skipped. Does not affect the contents of the register.

Bytes: 3

T-States: 14 (11)

Flag Bits Affected:
- Z
- SK
- HC
- L0 ← 0
- L1 ← 0
- CY

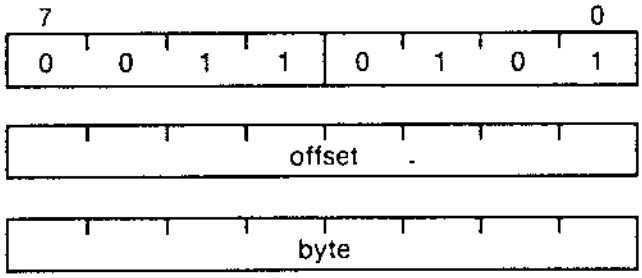
Example:

```
LTI     PD,20H  ;Subtract 20H from port D.
RET             ;Skip this instruction if a
                ;borrow is generated.
```

LTIW wa,byte

Skip next instruction if immediate data byte greater than working register.

((V) • wa) − byte, SK/byte > ((V) • wa)



Instruction Code: 35H, offset, byte

Subtracts the immediate data byte from the contents of the working register addressed by the V-register (8 high-order address bits) and the offset (8 lower-order address bits). If a borrow is generated, data byte > ((V) • wa), the next instruction is skipped. The contents of the working register are not affected.

Bytes: 3

T-States: 13 (10)

Flag Bits Affected:
- Z
- SK
- HC
- L0 ← 0
- L1 ← 0
- CY

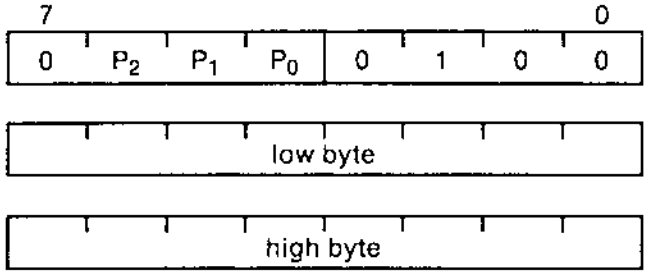
Example:

```
MVI     V,80H
LTIW    10H,20H
RET             ;Skip this instruction if
                ;(8010H) < 20H.
```

LXI rp2,word

Load register pair with immediate data.

(rp2) ← word



Instruction Code: (0-4)4H, low, high

Loads the immediate data bytes into register pair rp2 (BC, DE, HL, SP, or EA) designated by P_2-P_0 (0-4). If HL is the designated register pair, then the L0 flag is set. Subsequent LXI H,word instructions are skipped until another type of instruction is executed.

Bytes: 3

T-States: 10 (10)

Flag Bits Affected:

SK ← 0
L0 ← (rp2 = HL)
L0 ← (rp2 ≠ HL)
L1 ← 0

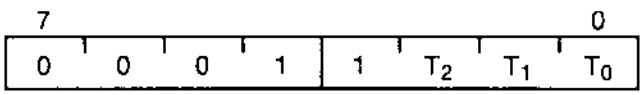
Example:

```
L1:  LXI    H,2000H  ;Depending on where this
L2:  LXI    H,4000H  ;sequence is entered, either
L3:  LXI    H,6000H  ;2000H, 4000H, or 6000H
     SHLD   0FF00H   ;will be stored at location
                     ;FF00H.
```

MOV r1,A

Move A to register.

(r1) ← (A)



Instruction Code: 1(8-F)H

Transfers 8 bits of data from the contents of the accumulator to register r1 (EAH, EAL, B, C, D, E, H, or L) as designated by T_2-T_0 (0-7).

Bytes: 1

T-States: 4 (4)

Flag Bits Affected:

SK ← 0
L0 ← 0
L1 ← 0

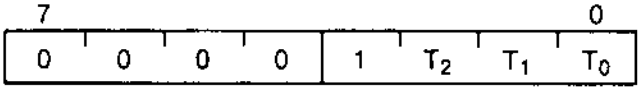
Example:

```
MOV     C,A     ;Move the contents of A into C.
```

MOV A,r1

Move register to A.

(A) ← (r1)



Instruction Code: 0(8-F)H

Transfers 8 bits of data from the contents of register r1 (EAH, EAL, B, C, D, E, H, or L) as designated by T_2-T_0 (0-7H) to the accumulator.

Bytes: 1

T-States: 4 (4)

Flag Bits Affected:

SK ← 0
L0 ← 0
L1 ← 0

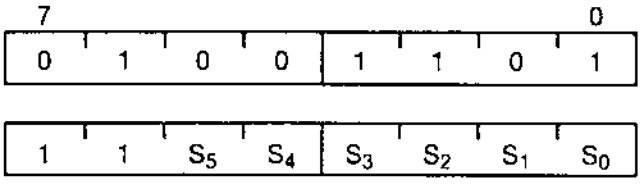
Example:

```
MOV     A,C     ;Move the contents of C into A.
```

MOV sr,A

Move A to special register.

(sr) ← (A)



Instruction Code:

4DC(0-3, 5-D)H
4DD(0-4, 7, 8, A, B)H
4DE8H

Transfers 8 bits of data from the accumulator to special register sr (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, SML, E0M, ETMM, TMM, MM, MCC, MA, MB, MC, MF, ZCM, TXB, TM0, or TM1) as designated by S_5-S_0 (0-3, 5-D, 10-14, 17, 18, 1A, 1B, or 28).

Bytes: 2

T-States: 10 (8)

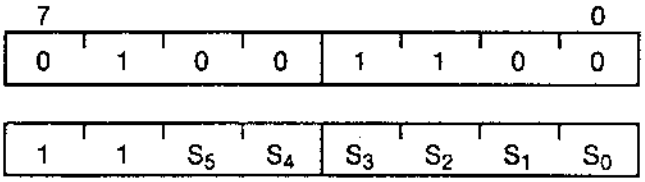
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
MOV     PA,A    ;Move the contents of A to
                ;port A latch.
```

MOV A,sr1

Move special register to A.

(A) ← (sr1)



Instruction Code:
4CC(0-3, 5-9, B, D)H
4CD(9)H
4CE(0-3)H

Transfers 8 bits of data from the contents of special register sr1 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, TMM, RXB, CR0, CR1, CR2, or CR3) as designated by S_5-S_0 (0-3, 5-9, B, D, 19, 20-23) to the accumulator.

Bytes: 2

T-States: 10 (8)

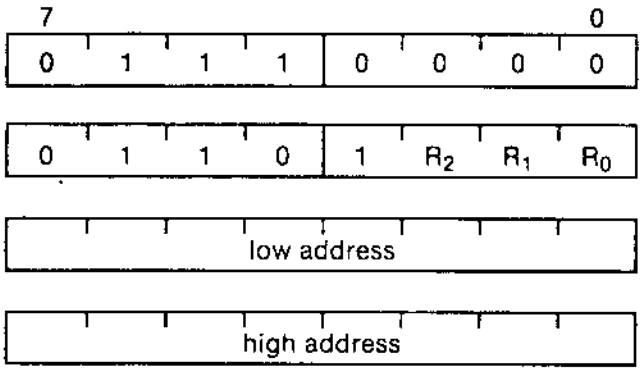
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
MOV     A,TMM   ;Move the contents of timer
                ;mode register into A.
```

MOV r,word

Move memory to register.

(r) ← (word)



Instruction Code: 706(8-F)H, low, high

Transfers 8 bits of data from the contents of memory as addressed by the third (low address) and fourth (high address) bytes to register r (V, A, B, C, D, E, H, or L) as designated by R_2-R_0 (0-7).

Bytes: 4

T States: 17 (14)

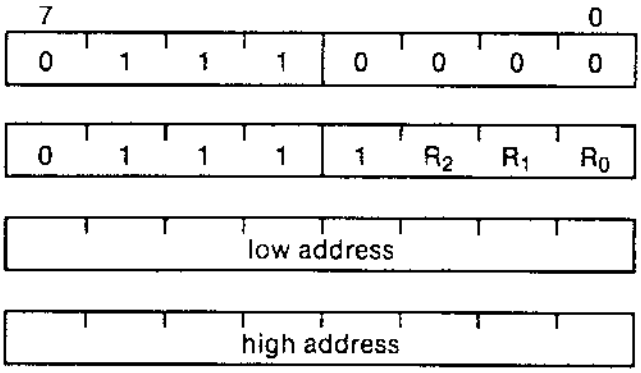
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
MOV     V,1234H ;Move the contents of memory
                ;location 1234H into V.
```

MOV word,r

Move register to memory.

(word) ← (r)



Instruction Code: 707(8-F)H, low, high

Transfers 8 bits of data from the contents of register r (V, A, B, C, D, E, H, or L) as designated by R_2-R_0 (0-7) to memory as addressed by the third (low address) and fourth (high address) bytes.

Bytes: 4

T-States: 17 (14)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

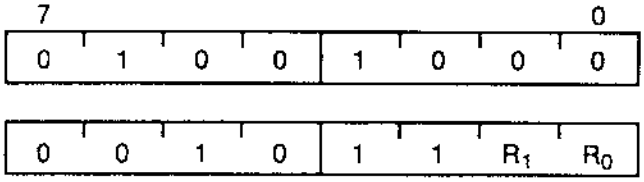
Example:

```
MOV     1234H,V  ;Move the contents of V into
                 ;memory location 1234H.
```

MUL r2

Multiply A by register.

(EA) ← (A) x (r2)



Instruction Code: 482(D-F)H

Performs an 8-bit by 8-bit unsigned multiplication of the contents of the accumulator by that of register *r2* (A, B, or C) as designated by R_1-R_0 (1-3) and leaves the 16-bit product in the extended accumulator.

Bytes: 2

T-States: 32 (8)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

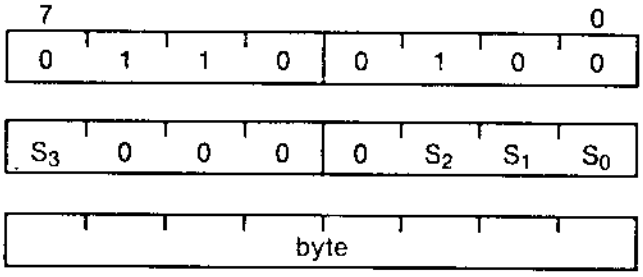
Example:

```
MUL     C        ;Multiply A by C, result in EA.
```

MVI sr2,byte

Move immediate data byte to special register.

(sr2) ← byte



Instruction Code:
640(0-3, 5-7)H, byte
648(0, 1, 3, 5)H, byte

Transfers the immediate data byte (third byte) to special register sr2 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, or TMM), designated by S_3-S_0 (0-3, 5-9, B, D).

Bytes: 3

T-States: 14 (11)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

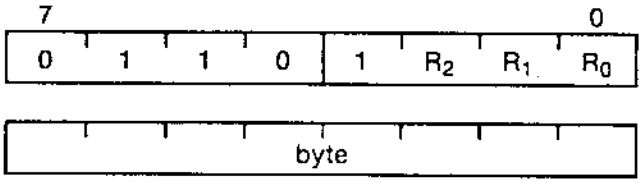
Example:

```
MVI     PD,20H   ;Move 20H into port D.
```

MVI r,byte

Move immediate data byte to register.

(r) ← byte



Instruction Code: 6(8-F)H, data

Transfers the immediate data byte to register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7). If A is the designated register, then all further MVI A,byte instructions will be skipped until another instruction type is encountered. Then, L1 is cleared. If L is the designated register, then all further MVI L,byte instructions will be skipped until another instruction type is encountered; then, L0 is cleared.

Bytes: 2

T-States: 7 (7)

Flag Bits Affected:

r = A	r = L	All Others
SK ← 0	SK ← 0	SK ← 0
L0 ← 0	L0 ← 1	L0 ← 0
L1 ← 1	L1 ← 0	L1 ← 0

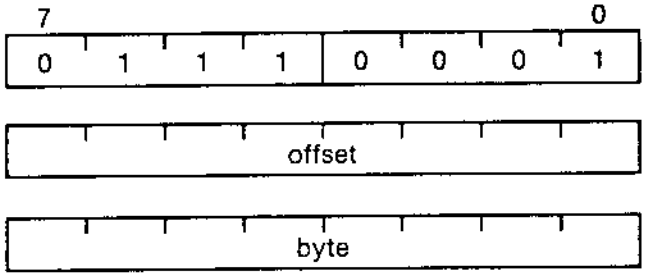
Example:

```
MVI     A,30H     ;Depending on where this
MVI     A,31H     ;sequence is entered, ASCII 0,
MVI     A,32H     ;1, or 2 is output to port A.
MOV     PA,A
```

MVIW wa,byte

Move immediate data byte to working register.

((V) • wa) ← byte



Instruction Code: 71H, offset, byte

Transfers the immediata data byte to the working register addressed by the V-register (8 high-order address bits) and the offset (8 lower-order address bits).

Bytes: 3

T-States: 13 (10)

Flag Bits Affected:

SK ← 0
L0 ← 0
L1 ← 0

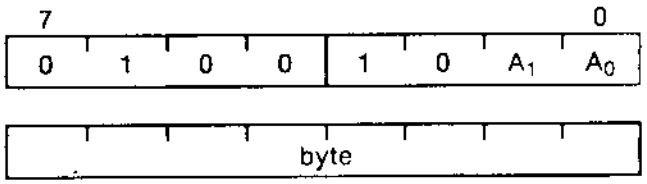
Example:

```
MVI     V,12H     ;Put 12H in V.
MVIW    10H,20H   ;Store 20H at address 1210H.
```

MVIX rpa1,byte

Move immediate data byte to memory.

((rpa1)) ← byte



Instruction Code: 4(9-B)H, byte

Transfers the immediate data byte to the memory addressed by register pair rpa1 (BC, DE, or HL) as designated by A_1-A_0 (1-3).

Bytes: 2

T-States: 10 (7)

Flag Bits Affected:

SK ← 0
L0 ← 0
L1 ← 0

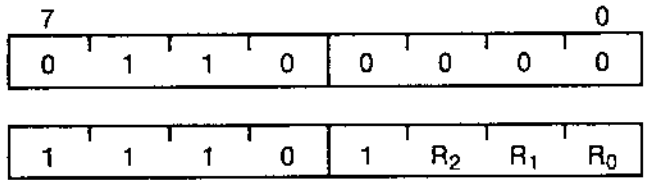
Example:

```
MVIX    B,9       ;Put a 9 at memory location
                  ;specified by register pair BC.
```

NEA A,r

Skip next instruction if register not equal to A.

(A) − (r), SK/(r) ≠ (A)



Instruction Code: 60E(8-F)H

Subtracts the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), from the contents of the accumulator. If the result is other than zero, (A) ≠ (r), the next instruction is skipped. Does not affect the contents of the accumulator or the register.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:

Z
SK
HC
L0 ← 0
L1 ← 0
CY

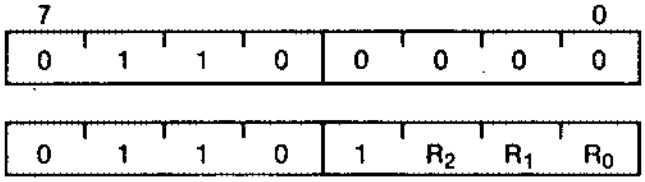
Example:

```
NEA     A,C
RET             ;This instruction is skipped if
                ;A ≠ C.
```

NEA r,A

Skip next instruction if A not equal to register.

(r) – (A), SK/(A) ≠ (r)



Instruction Code: 606(B-F)H

Subtracts the contents of the accumulator from register r (V, A, B, C, D, E, H, or L) designated by R_2-R_0 (0-7). If the result is other than zero, (A) ≠ (r), the next instruction is skipped. The contents of the register or accumulator are not affected.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:

Z
SK
HC
L0 ← 0
L1 ← 0
CY

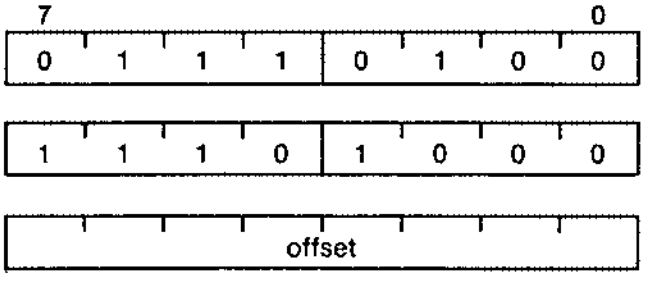
Example:

```
NEA     C,A
RET             ;If A not equal to C, RET is
                ;skipped.
```

NEAW wa

Skip next instruction if A not equal to the working register.

(A) – ((V) • wa), SK/(A) ≠ ((V) • wa)



Instruction Code: 74E8H, offset

Subtracts the contents of the accumulator from the contents of the working register addressed by the V-register (8-bit high-order address) and the offset (8-bit low-order address). If the result is other than zero, (A) ≠ ((V) • wa), the next instruction is skipped. The contents of the accumulator or working register are not affected.

Bytes: 3

T-States: 14 (11)

Flag Bits Affected:

Z
SK
HC
L0 ← 0
L1 ← 0
CY

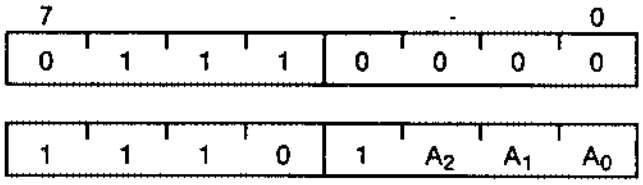
Example:

```
MVI     V,0AH
NEAW    10H
RET             ;Skip this instruction if
                ;contents of accumulator ≠ to
                ;contents of memory location
                ;0A10H.
```

NEAX rpa

Skip next instruction if A not equal to memory addressed by register pair.

(A) – ((rpa)), SK/(A) ≠ ((rpa))



Instruction Code: 70E(9-F)H

Subtracts the contents of the memory location addressed by register pair rpa (BC, DE, HL, DE+, HL+, DE–, or HL–), designated by A_2-A_0 (1-7), from the accumulator. The next instruction will be skipped if the result is other than zero, (A) ≠ ((rpa)). Does not affect the contents of the accumulator and memory.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:

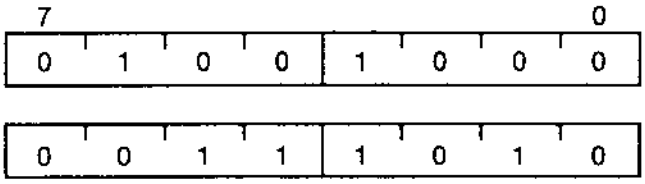
Z
SK
HC
L0 ← 0
L1 ← 0
CY

Example:
```
MVI   A,20H
NEAX  D       ;((D)) = 20H
RET           ;Skip this instruction if (A) ≠ 20H.
```

NEGA

Negate A.

(A) ← $\overline{(A)}$ + 1



Instruction Code: 483AH

Takes two's complement of the contents of the accumulator.

Bytes: 2

T-States: 8 (8)

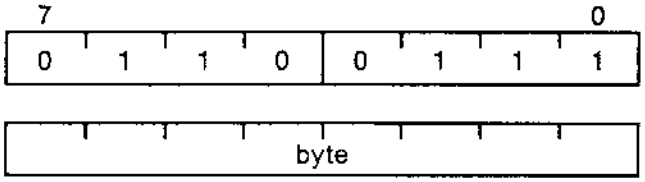
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
NEGA    ;Take two's complement of A.
```

NEI A,byte

Skip next instruction if A not equal to immediate data byte.

(A) − byte, SK/(A) ≠ byte



Instruction Code: 67H, byte

Subtracts the immediate data byte from the contents of the accumulator. If the result is other than zero, (A) ≠ data byte, then the next instruction is skipped. Does not affect the contents of the accumulator.

Bytes: 2

T-States: 7 (7)

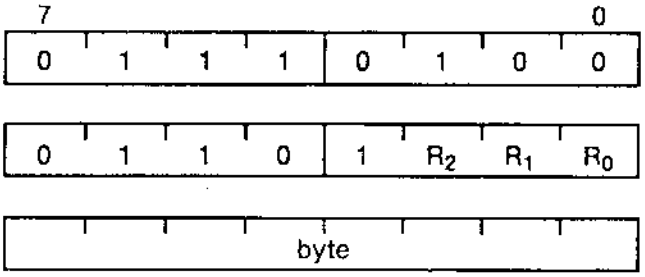
Flag Bits Affected:
Z
HC
SK
L0 ← 0
L1 ← 0
CY

Example:
```
NEI     A,20H
RET             ;Skip this instruction if A ≠ 20H.
```

NEI r,byte

Skip next instruction if register not equal to immediate data byte.

(r) − byte, SK/(r) ≠ byte



Instruction Code: 746(8-F)H, byte

Subtracts the immediate data byte from the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7). If the result is other than zero, (r) ≠ data byte, the next instruction is skipped. Does not affect the contents of the register.

Bytes: 3

T-States: 11 (11)

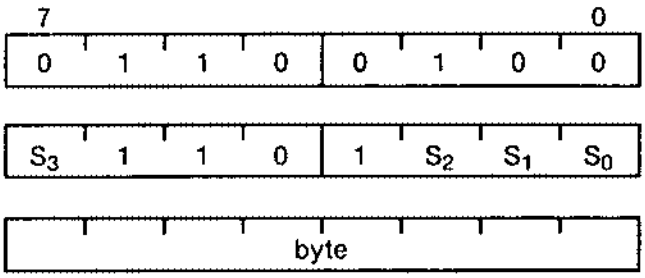
Flag Bits Affected:
Z
HC
SK
L0 ← 0
L1 ← 0
CY

Example:
```
NEI     D,20H
RET             ;Skip this instruction if D ≠ 20H.
```

NEI sr2,byte

Skip next instruction if special register not equal to immediate data byte.

(sr2) − byte, SK/(sr2) ≠ byte



Instruction Code:
646(8-B, D-F)H, byte
64E(8, 9, B, D)H, byte

Subtracts the immediate data byte from the contents of special register sr2 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, or TMM), designated by S_3-S_0 (0-3, 5-9, B, D). If the result is other than zero, (sr2) ≠ data byte, the next instruction is skipped. Does not affect the contents of the register.

Bytes: 3

T-States: 14 (11)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

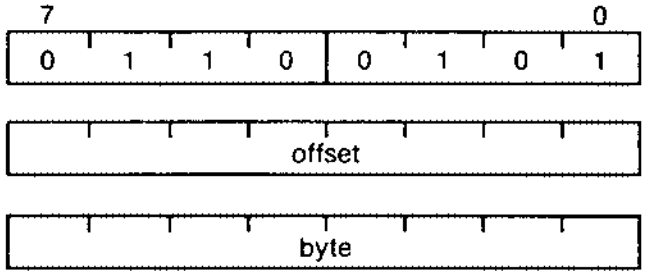
Example:

```
NEI     PD,20H   ;Compare 20H with port D.
RET              ;Skip this instruction if port D
                 ;≠ 20H.
```

NEIW wa,byte

Skip next instruction if working register not equal to immediate data byte.

((V) • wa) − byte, SK/((V) • wa) ≠ byte



Instruction Code: 65H, offset, byte

Subtracts the immediate data byte from the contents of the working register addressed by the V-register (8 high-order address bits) and the offset (8 lower-order address bits). If the result is other than zero, ((V) • wa) ≠ data byte, the next instruction is skipped. Does not affect the contents of the working register.

Bytes: 3

T-States: 13 (10)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

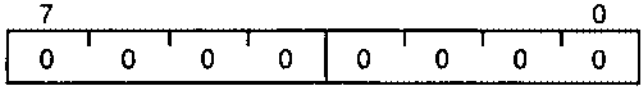
Example:

```
MVI     V,0AAH
NEIW    10H,20H
RET              ;Skip this instruction if
                 ;contents of location AA10H
                 ;≠ 20H.
```

NOP

No operation.

(PC) ← (PC) + 1



Instruction Code: 00H

This instruction does not perform an operation, but uses up four clock cycles.

Bytes: 1

T-States: 4 (4)

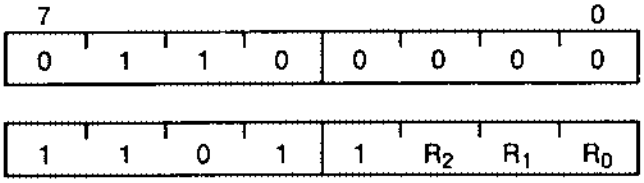
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
NOP     ;No operation.
```

OFFA A,r

Skip next instruction if off-test register with A is zero.

(A) AND (r), SK/Z



Instruction Code: 60D(8-F)H

Performs logical AND of the contents of the accumulator with the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7). If the result is zero, the next instruction is skipped. Does not affect the contents of the accumulator or the register.

Bytes: 2

T-States: 8 (8)

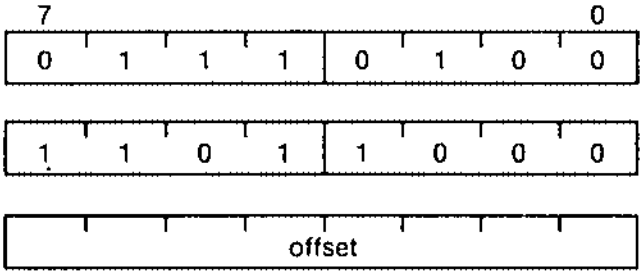
Flag Bits Affected:
Z
SK
L0 ← 0
L1 ← 0

Example:
```
OFFA    A,C     ;AND A with C.
RET             ;This instruction is skipped if
                ;the result is zero.
```

OFFAW wa

Skip next instruction if off-test working register with A is zero.

(A) AND ((V) • wa), SK/Z



Instruction Code: 74D8H, offset

Performs logical AND of the contents of a working register to be addressed by the V-register (8-bit high-order address) and the offset (8-bit low-order address) with the accumulator. If the result is zero, the next instruction is skipped. Does not affect the contents of the accumulator or the working register.

Bytes: 3

T-States: 14 (11)

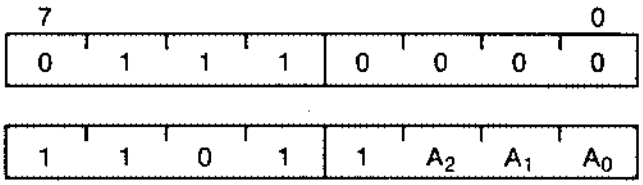
Flag Bits Affected:
Z
SK
L0 ← 0
L1 ← 0

Example:
```
MVI     A,02H
MVI     V,08H
OFFAW   10H
RET             ;Skip this instruction if bit 1 of
                ;location 0810H is not set.
```

OFFAX rpa

Skip next instruction if off-test memory addressed by register pair with A is zero.

(A) AND ((rpa)), SK/Z



Instruction Code: 70D(9-F)H

Performs a logical AND of the contents of the memory location that is addressed by register pair rpa (BC, DE, HL, DE+, HL+, DE−, or HL−), designated by A_2-A_0 (1-7), with the accumulator. The next instruction is skipped if the result is zero. Does not affect the contents of the accumulator or the memory.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
SK
L0 ← 0
L1 ← 0

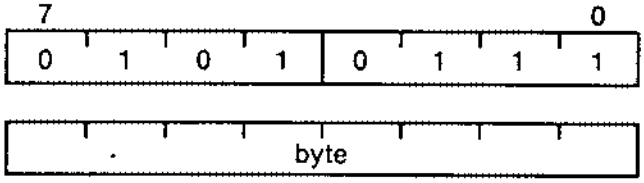
Example:

```
LXI     D,0F000H
MVI     A,80H
OFFAX   D
RET                 ;This instruction skipped if bit
                    ;7 of location F000H is not set.
```

OFFI A,byte

Skip next instruction if off-test immediate data byte with A is zero.

(A) AND byte, SK/Z



Instruction Code: 57H, byte

Performs a logical AND with the accumulator and the immediate data byte. If the result is zero, the next instruction is skipped. Does not affect the contents of the accumulator.

Bytes: 2

T-States: 7 (7)

Flag Bits Affected:
Z
SK
L0 ← 0
L1 ← 0

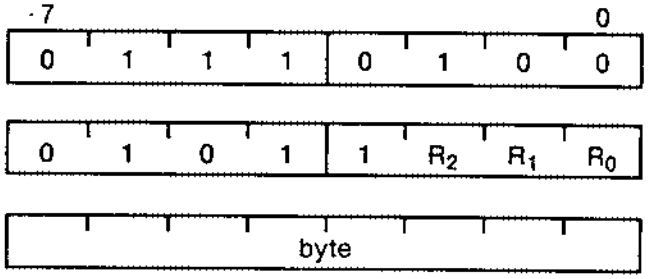
Example:

```
OFFI    A,20H
RET                 ;Skip this instruction if bit 5 of
                    ;the accumulator is not set.
```

OFFI r,byte

Skip next instruction if off-test immediate data byte with register is zero.

(r) AND byte, SK/Z



Instruction Code: 745(8-F)H, byte

Performs a logical AND with the immediate data byte and the contents of register r (V, A, B, C, D, E, H, or L) designated by R_2-R_0 (0-7). If the result is zero, the next instruction will be skipped. Does not affect the contents of the register.

Bytes: 3

T-States: 11 (11)

Flag Bits Affected:
Z
SK
L0 ← 0
L1 ← 0

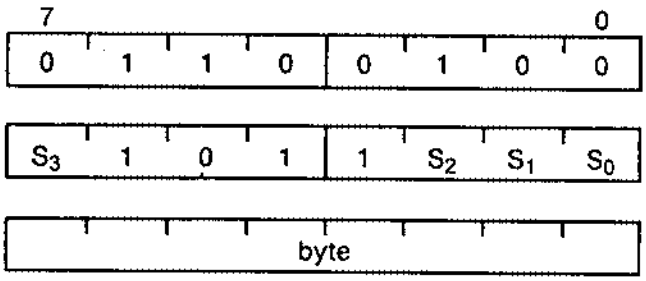
Example:

```
OFFI    D,20H
RET                 ;This instruction is skipped if
                    ;bit 5 in register D is not set.
```

OFFI sr2,byte

Skip next instruction if off-test immediate data byte with special register is zero.

(sr2) AND byte, SK/Z



Instruction Code:
645(8-B, D-F)H, byte
64D(8, 9, B, D)H, byte

Performs a logical AND of the immediate data byte with the contents of special register sr2 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, or TMM) designated by S_3-S_0 (0-3, 5-9, B, D). If the result is zero, the next instruction is skipped. Does not affect the contents of the register.

Bytes: 3

T-States: 14 (11)

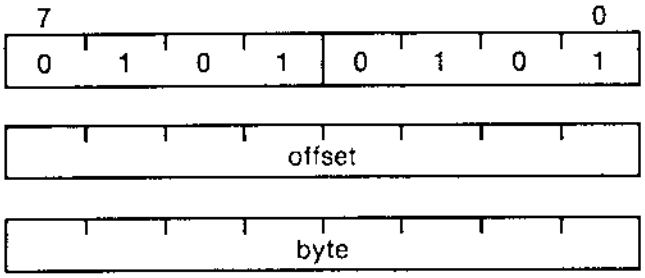
Flag Bits Affected:
Z
SK
L0 ← 0
L1 ← 0

Example:
```
OFFI    PD,20H
RET             ;Skip this instruction if bit 5 of
                ;port D is not set.
```

OFFIW wa,byte

Skip next instruction if off-test immediate data byte with working register is zero.

((V) • wa) AND byte, SK/Z



Instruction Code: 55H, offset, data

Performs a logical AND of the immediate data byte with the contents of the working register addressed by the V-register (8 high-order address bits) and the offset (8 low-order address bits). If the result is zero, the next instruction is skipped. Does not affect the contents of the working register.

Bytes: 3

T-States: 13 (10)

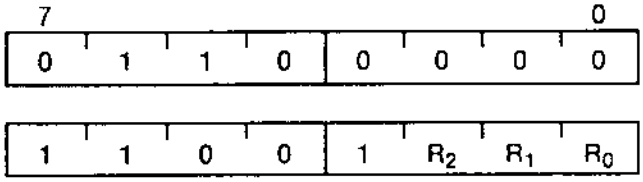
Flag Bits Affected:
Z
SK
L0 ← 0
L1 ← 0

Example:
```
MVI     V,0FH
OFFIW   10H,20H
RET             ;Skip this instruction if bit 5 of
                ;location 0F10H is not set.
```

ONA A,r

Skip next instruction if on-test register with A is not zero.

(A) AND (r), SK/NZ



Instruction Code: 60C(8-F)H

Performs a logical AND between the contents of the accumulator and the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7). If the result is not zero, the next instruction is skipped. Does not affect the contents of the accumulator or the register.

Bytes: 2

T-States: 8 (8)

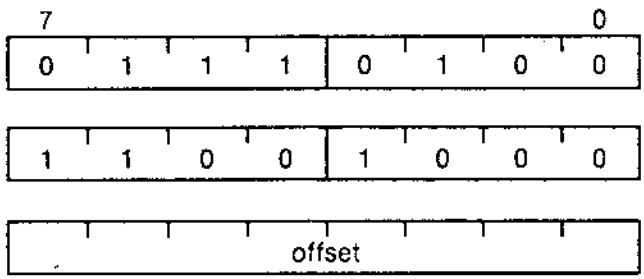
Flag Bits Affected:
Z
SK
L0 ← 0
L1 ← 0

Example:
```
ONA     A,C     ;AND A with C.
RET             ;This instruction is skipped if
                ;the result is not zero.
```

ONAW wa

Skip next instruction if on-test working register with A is not zero.

(A) AND ((V) • wa), SK/NZ



Instruction Code: 74C8H, offset

Performs a logical AND between the contents of a working register to be addressed by the V-register (8-bit high-order address) and the offset (8-bit low order address) with the accumulator. If the result is not zero, the next instruction is skipped. Does not affect the contents of the accumulator or the working register.

Bytes: 3

T-States: 14 (11)

Flag Bits Affected:
Z
SK
L0 ← 0
L1 ← 0

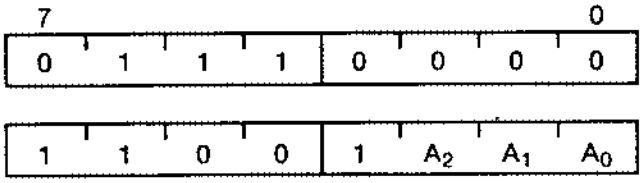
Example:

```
MVI     A,0FFH
MVI     V,0FH
ONAW    10H
RET             ;Skip this instruction if any bit
                ;of working register location
                ;0F10H is set.
```

ONAX rpa

Skip next instruction if off-test memory addressed by register pair with A is not zero.

(A) AND ((rpa)), SK/NZ



Instruction Code: 70C(9-F)H

Performs a logical AND between the contents of memory addressed by register pair rpa (BC, DE, HL, DE+, HL+, DE−, or HL−), designated by A_2-A_0 (1-7), and the accumulator. The next instruction is skipped if the result is not zero. Does not affect the contents of memory or the accumulator.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
SK
L0 ← 0
L1 ← 0

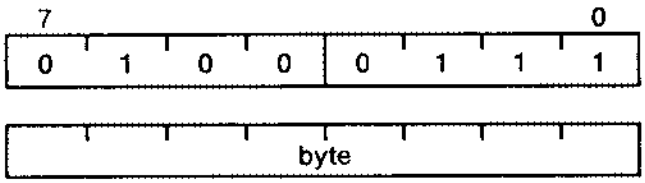
Example:

```
MVI     A,80H
ONAX    D
RET             ;This instruction skipped if bit
                ;7 of the memory location
                ;pointed to by register D is set.
```

ONI A,byte

Skip next instruction if on-test immediate data byte with A is not zero.

(A) AND byte, SK/NZ



Instruction Code: 47H, byte

Performs a logical AND with the accumulator and the immediate data byte. If the result is not zero, the next instruction is skipped. Does not affect the contents of the accumulator.

Bytes: 2

T-States: 7 (7)

Flag Bits Affected:
Z
SK
L0 ← 0
L1 ← 0

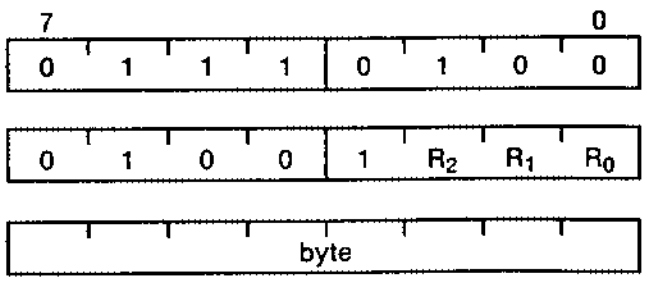
Example:

```
ONI     A,20H   ;AND 20H with the
                ;accumulator.
RET             ;Skip this instruction if the
                ;result is not zero.
```

ONI r,byte

Skip next instruction if on-test immediate data byte with register is not zero.

(r) AND byte, SK/NZ



Instruction Code: 744(8-F)H, byte

Performs a logical AND with the immediate data byte and the contents of register r (V, A, B, C, D, E, H, or L) designated by R_2-R_0 (0-7). If the result is not zero, the next instruction will be skipped. Does not affect the contents of the register.

Bytes: 3

T-States: 11 (11)

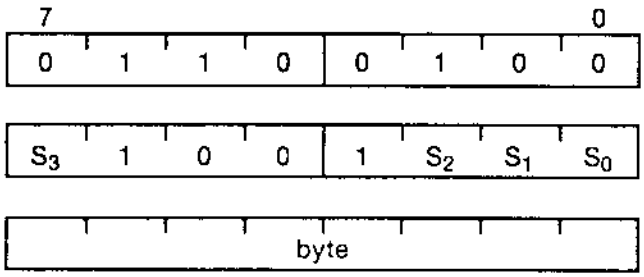
Flag Bits Affected:
Z
SK
L0 ← 0
L1 ← 0

Example:
```
ONI     D,20H
RET             ;Skip this instruction if bit 5 of
                ;register D is set.
```

ONI sr2,byte

Skip next instruction if on-test immediate data byte with special register is not zero.

(sr2) AND byte, SK/NZ



Instruction Code:
644(8-B, D-F)H, byte
64C(8, 9, B, D)H, byte

Performs logical AND of the immediate data byte with the contents of special register sr2 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, or TMM) designated by S_3-S_0 (0-3, 5-9, B, D). If the result is not zero, skip the next instruction. Does not affect the contents of the register.

Bytes: 3

T-States: 14 (11)

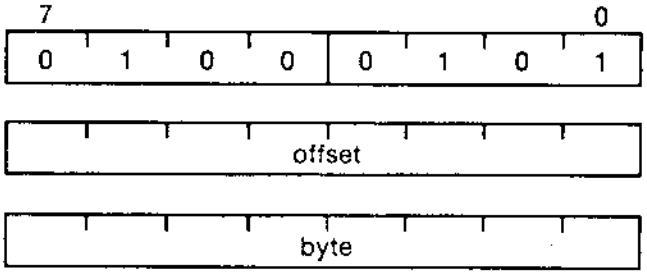
Flag Bits Affected:
Z
SK
L0 ← 0
L1 ← 0

Example:
```
ONI     PD,20H
RET             ;Skip this instruction if bit 5 of
                ;port D is set.
```

ONIW wa,byte

Skip next instruction if on-test immediate data byte with working register is not zero.

((V) • wa) AND byte, SK/NZ



Instruction Code: 45H, offset, byte

Performs a logical AND with the immediate data byte and the contents of the working register addressed by the V-register (8 high-order address bits) and the offset (8 low-order address bits). If the result is not zero, the next instruction is skipped. Does not affect the contents of the working register.

Bytes: 3

T-States: 13 (10)

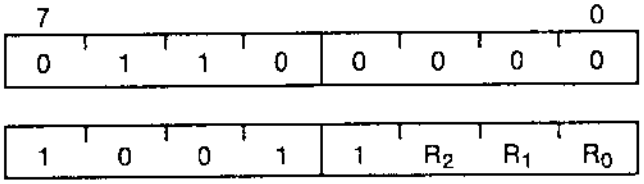
Flag Bits Affected:
Z
SK
L0 ← 0
L1 ← 0

Example:
```
MVI     V,0FH
ONIW    10H,20H
RET             ;Skip this instruction if bit 5 of
                ;the working register location
                ;0F10H is set.
```

ORA A,r

OR A with register.

(A) ← (A) OR (r)



Instruction Code: 609(8-F)H

Performs a logical OR between the contents of the accumulator and the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7). Stores the result in the accumulator.

Bytes: 2

T-States: 8 (8)

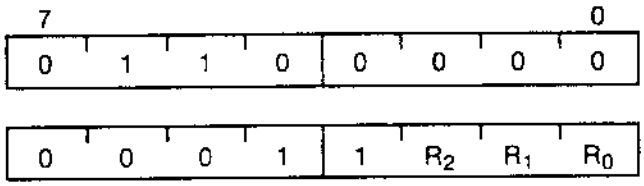
Flag Bits Affected:
- Z
- SK ← 0
- L0 ← 0
- L1 ← 0

Example:
```
ORA     A,C     ;OR C with A.
```

ORA r,A

OR register with A.

(r) ← (r) OR (A)



Instruction Code: 601(8-F)H

Performs a logical OR of the contents of the accumulator with the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7). Stores the result in the designated register.

Bytes: 2

T-States: 8 (8)

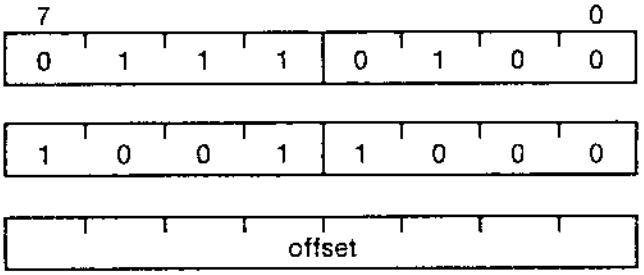
Flag Bits Affected:
- Z
- SK ← 0
- L0 ← 0
- L1 ← 0

Example:
```
ORA     C,A     ;OR A with C.
```

ORAW wa

OR working register with A.

(A) ← (A) OR ((V) • wa)



Instruction Code: 7498H, offset

Performs a logical OR of the contents of a working register to be addressed by the V-register (8-bit high-order address) and the offset (8-bit low-order address) with the accumulator. Stores the result in the accumulator.

Bytes: 3

T-States: 14 (11)

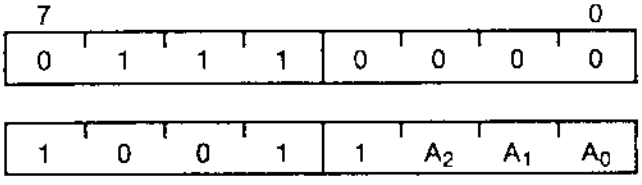
Flag Bits Affected:
- Z
- SK ← 0
- L0 ← 0
- L1 ← 0

Example:
```
MVI     V,00H
ORAW    10H     ;OR the working register
                ;0010H with A.
```

ORAX rpa

OR memory addressed by register pair with A.

(A) ← (A) OR ((rpa))



Instruction Code: 709(9-F)H

Performs a logical OR between the contents of memory addressed by register pair rpa (BC, DE, HL, DE+, HL+, DE−, or HL−), designated by A_2-A_0 (1-7), with the accumulator. The result is stored in the accumulator.

Bytes: 2

T-States: 11 (8)

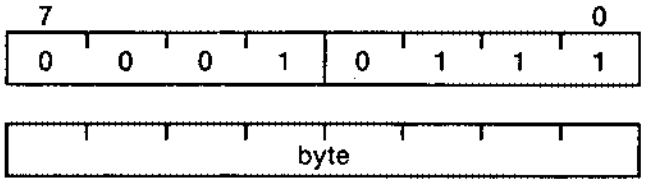
Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
ORAX    D       ;OR memory pointed to by DE
                ;with the accumulator.
```

ORI A,byte

OR immediate data byte with A.

(A) ← (A) OR byte



Instruction Code: 17H, byte

Performs a logical OR with the accumulator and the immediate data byte. The result is stored in the accumulator.

Bytes: 2

T-States: 7 (7)

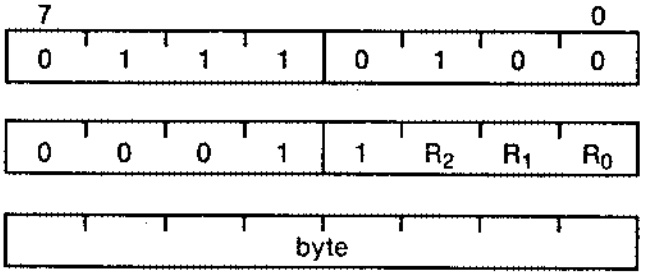
Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
ORI     A,20H   ;OR 20H with the
                ;accumulator.
```

ORI r,byte

OR immediate data byte with register.

(r) ← (r) OR byte



Instruction Code: 741(8-F)H, byte

Performs a logical OR with the immediate data byte and the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7). The result is stored in the designated register.

Bytes: 3

T-States: 11 (11)

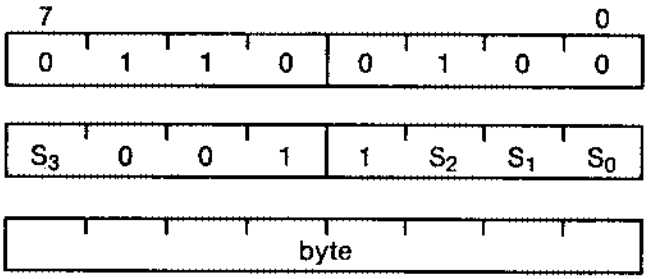
Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
ORI     D,20H   ;OR 20H with D.
```

ORI sr2,byte

OR immediate data byte with special register.

(sr2) ← (sr2) OR byte



Instruction Code:
641(8-B, D-F)H, byte
649(8, 9, B, D)H, byte

Performs a logical OR of the immediate data byte with the contents of special register sr2 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, or TMM) designated by S_3-S_0 (0-3, 5-9, B, D). The result is stored in the designated special register.

Bytes: 3

T-States: 20 (11)

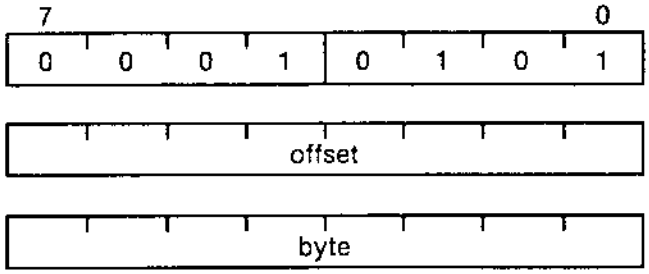
Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
ORI     PD,20H  ;Do a logical OR with port D
                ;and 20H. This sets bit PD5.
```

ORIW wa,byte

OR immediate data byte with working register.

((V) • wa) ← ((V) • wa) OR byte



Instruction Code: 15H, offset, byte

Performs a logical OR between the immediate data byte and the contents of the working register addressed by the V-register (8 high-order address bits) and the offset (8 low-order address bits). The result is stored in the working register.

Bytes: 3

T-States: 19 (10)

Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

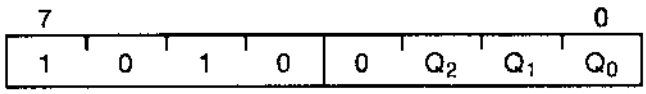
Example:

```
MVI     V,0FFH
ORIW    10H,20H  ;OR 20H with the working
                 ;register at (FF10H).
```

POP rp1

Pop register pair off stack.

(rp1L) ← ((SP)), (rp1H) ← ((SP) + 1), (SP) ← (SP) + 2



Instruction Code: A(0-4)H

The top two bytes of the memory stack are popped into register pair rp1 (VA, BC, DE, HL, or EA) as designated by Q_2-Q_0 (0-4). Then the stack pointer is incremented twice.

Bytes: 1

T-States: 10 (4)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

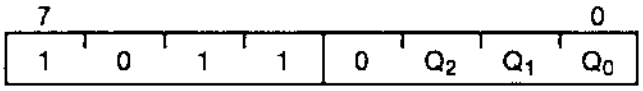
Example:

```
POP     H        ;Get 16-bit value from top of
                 ;stack into HL. SP ← SP + 2.
```

PUSH rp1

Push register pair on stack.

((SP) − 1) ← rp1H, ((SP) − 2) ← rp1L,
(SP) ← (SP) − 2



Instruction Code: B(0-4)H

The 16-bit value of register pair rp1 (VA, BC, DE, HL, or EA), designated by Q_2-Q_0 (0-4), is pushed into the top two bytes of the stack. Then the stack pointer is decremented by 2 to point to the next available location.

Bytes: 1

T-States: 13 (4)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

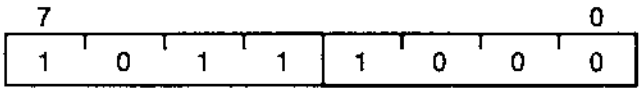
Example:

```
PUSH    H        ;Save 16-bit value of HL on
                 ;top of stack. SP ← SP − 2.
```

RET

Return from subroutine.

(PCL) ← ((SP)), (PCH) ← ((SP) + 1), (SP) ← (SP) + 2



Instruction Code: B8H

The top two bytes of the stack are removed from the stack and loaded into the program counter and the SP is incremented by 2.

Bytes: 1

T-States: 10 (4)

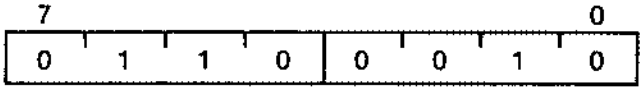
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

Example:
RET ;Return from subroutine.

RETI

Return from interrupt.

(PCL) ← ((SP)), (PCH) ← ((SP) + 1), (PSW) ← ((SP) + 2), (SP) ← (SP) + 3



Instruction Code: 62H

The top two bytes of the stack are removed from the stack and placed into the program counter. The next byte is popped into the PSW and the SP is incremented by 3.

Bytes: 1

T-States: 13 (4)

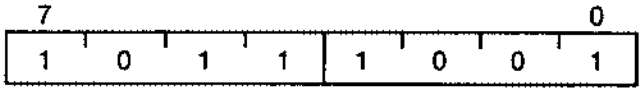
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

Example:
RETI ;Return from interrupt.

RETS

Return from subroutine and skip next instruction.

(PCL) ← ((SP)), (PCH) ← ((SP) + 1), (SP) ← (SP) + 2



Instruction Code: B9H

The top two bytes of the stack are popped into the program counter and the SP is incremented by 2. Then the instruction pointed to by the PC is skipped.

Bytes: 1

T-States: 10 (4)

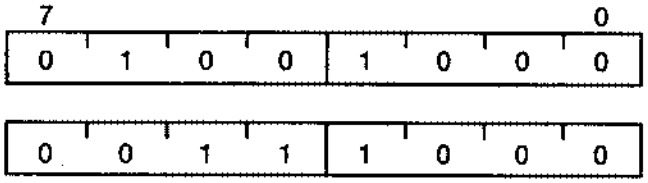
Flag Bits Affected:
SK ← 1
L0 ← 0
L1 ← 0

Example:
RETS ;Return from subroutine and
;skip next instruction.

RLD

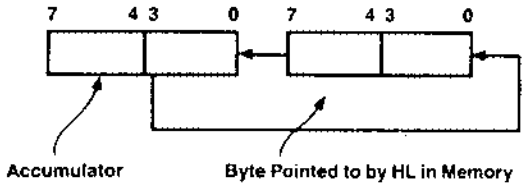
Rotate left digit.

$(A_3\text{-}A_0) \leftarrow ((HL_7\text{-}HL_4))$, $((HL_7\text{-}HL_4)) \leftarrow ((HL_3\text{-}HL_0))$, $((HL_3\text{-}HL_0)) \leftarrow (A_3\text{-}A_0)$



Instruction Code: 4838H

Rotates the four high bits in memory addressed by HL to the four low bits in the accumulator. Rotates the four low bits in memory to the high four bits in memory. Rotates left the low four bits in the accumulator into the four low bits of memory addressed by HL. The four high bits in the accumulator are not affected by this instruction.

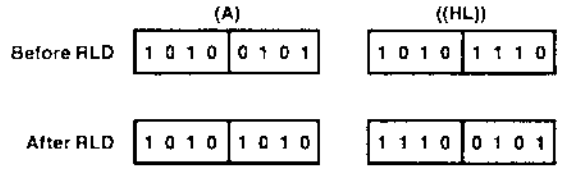


Bytes: 2

T-States: 17 (8)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

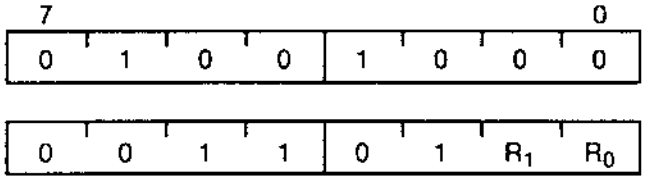
Example:
RLD



RLL r2

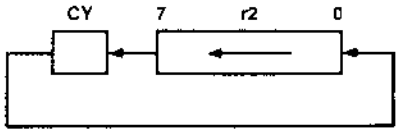
Rotate register left one through carry.

$(r2_{m+1} \leftarrow (r2_m),\ (r2_0) \leftarrow (CY),\ (CY) \leftarrow (r2_7)$



Instruction Code: 483(5-7)H

Performs a 1-bit left rotate through carry with the contents of register r2 (A, B, or C), designated by R_1-R_0 (1-3).



Bytes: 2

T-States: 8 (8)

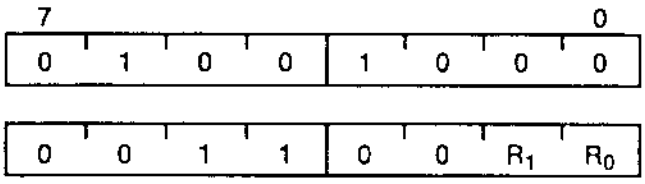
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0
CY

Example:
RLL C

RLR r2

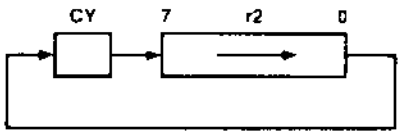
Rotate register right one through carry.

$(r2_{m-1}) \leftarrow (r2_m),\ (r2_7) \leftarrow (CY),\ (CY) \leftarrow (r2_0)$



Instruction Code: 483(1-3)H

This instruction does a 1-bit right rotate through carry with the contents of register r2 (A, B, or C), designated by R_1-R_0 (1-3).



Bytes: 2

T-States: 8 (8)

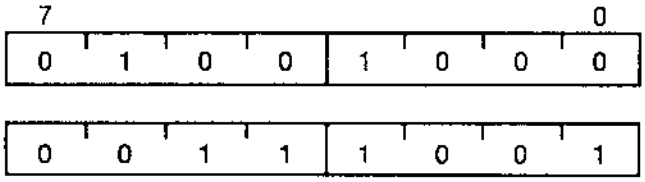
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0
CY

Example:
RLR C ;Rotate right C through carry.

RRD

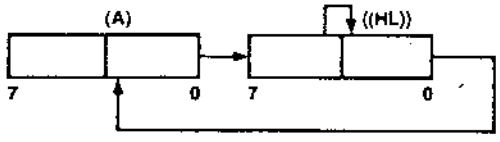
Rotate right digit.

$((HL_7\text{-}HL_4)) \leftarrow (A_3\text{-}A_0),\ ((HL_3\text{-}HL_0)) \leftarrow ((HL_7\text{-}HL_4)),\ (A_3\text{-}A_0) \leftarrow ((HL_3\text{-}HL_0))$



Instruction Code: 4839H

Rotates right the low four bits in the accumulator into the high four bits of memory addressed by HL, and rotates the four high bits in memory to the low four bits in memory. Rotates the four low bits in memory to the four low bits in the accumulator. The four high bits in the accumulator are not affected by this instruction.

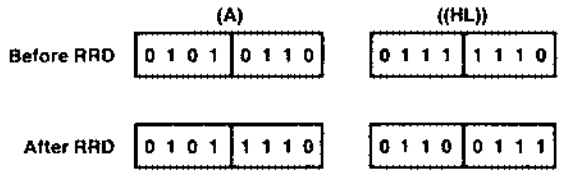


Bytes: 2

T-States: 17 (8)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

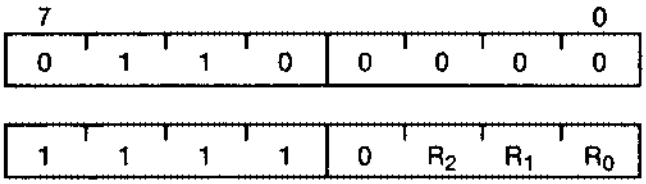
Example:
RRD



SBB A,r

Subtract register from A with borrow.

(A) ← (A) − (r) − (CY)



Instruction Code: 60F(0-7)H

Subtracts the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), from the accumulator, including the carry flag, and stores the result in the accumulator.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

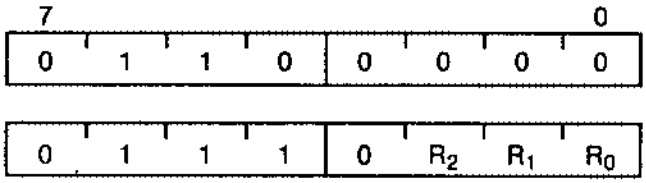
Example:

```
SBB     A,C     ;Subtract C from A with
                ;borrow.
```

SBB r,A

Subtract A from register with borrow.

(r) ← (r) − (A) − (CY)



Instruction Code: 607(0-7)H

Subtracts the contents of the accumulator, including the carry flag, from the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), and stores the result in the designated register.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

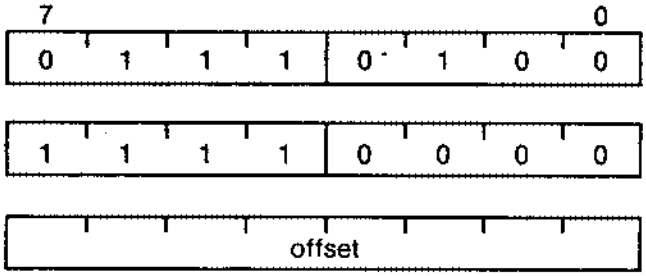
Example:

```
SBB     C,A     ;Subtract A from C with carry.
```

SBBW wa

Subtract working register from A with borrow.

(A) ← (A) − ((V) • wa) − (CY)



Instruction Code: 74F0H, offset

Subtracts the CY bit and the contents of a working register to be addressed by the V-register (8-bit high-order address) and the offset (8-bit low-order address) from the accumulator and stores the result in the accumulator.

Bytes: 3

T-States: 14 (11)

Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

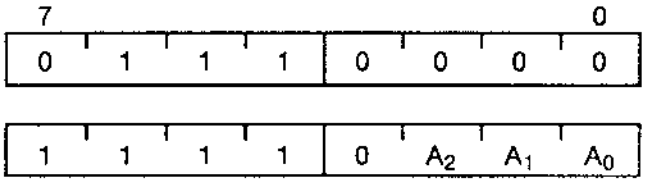
Example:

```
MVI     V,0FFH
SBBW    10H     ;Subtract working register
                ;0FF10H from accumulator.
```

SBBX rpa

Subtract memory addressed by register pair from A with borrow.

(A) ← (A) − ((rpa)) − (CY)



Instruction Code: 70F(1-7)H

Subtracts the contents of the memory location addressed by register pair rpa (BC, DE, HL, DE+, HL+, DE−, or HL−), designated by A_2-A_0 (1-7), and the CY flag from the accumulator. The result is stored in the accumulator.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:

Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

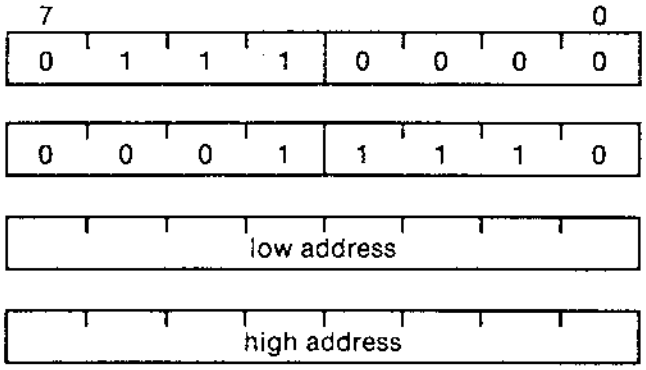
Example:

```
SBBX    D       ;Subtract memory pointed to
                ;by DE and the CY flag from
                ;accumulator.
```

SBCD word

Store register pair BC direct.

(word) ← (C), (word + 1) ← (B)



Instruction Code: 701EH, low, high

The contents of C are stored at location word, and the contents of B are stored at location word + 1.

Bytes: 4

T-States: 20 (14)

Flag Bits Affected:

SK ← 0
L0 ← 0
L1 ← 0

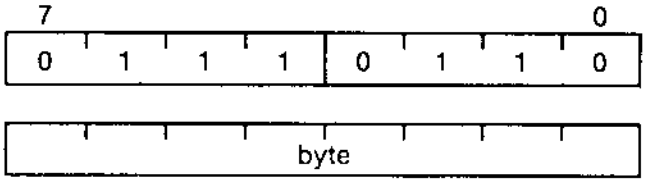
Example:

```
SBCD    1000H   ;C is stored at 1000H and B
                ;at 1001H.
```

SBI A,byte

Subtract immediate data byte from A with borrow.

(A) ← (A) − byte − (CY)



Instruction Code: 76H, byte

Subtracts the immediate data byte, including the carry, from the contents of the accumulator and stores the result in the accumulator.

Bytes: 2

T-States: 7 (7)

Flag Bits Affected:

Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

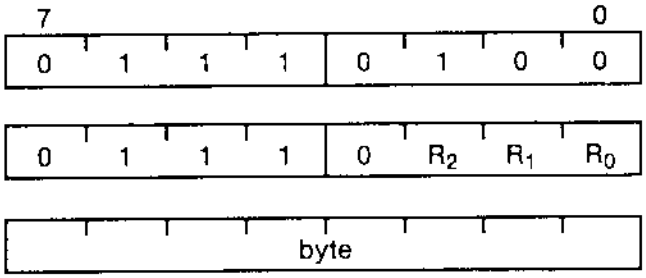
Example:

```
SBI     A,20H   ;Subtract 20H plus carry
                ;from A.
```

SBI r,byte

Subtract immediate data byte from register with borrow.

(r) ← (r) − byte − (CY)



Instruction Code: 747(0-7)H, byte

Subtracts the immediate data byte plus carry from the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7). The result is stored in the register.

Bytes: 3

T-States: 11 (11)

Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

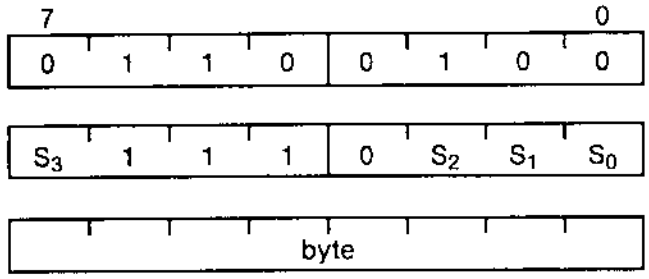
Example:

```
SBI     D,20H   ;Subtract 20H plus carry
                ;from D.
```

SBI sr2,byte

Subtract immediate data byte from special register with borrow.

(sr2) ← (sr2) − byte − (CY)



Instruction Code:
647(0-3, 5-7)H, byte
64F(0, 1, 3, 5)H, byte

Subtracts the immediate data byte plus carry from the contents of special register sr2 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, or TMM). The register is designated by S_3-S_0 (0-3, 5-9, B, D). The result is stored in the designated special register.

Bytes: 3

T-States: 20 (11)

Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

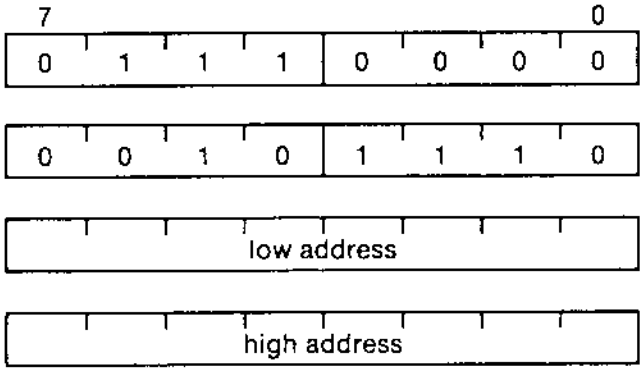
Example:

```
SBI     PD,20H  ;Subtract 20H plus carry from
                ;port D.
```

SDED word

Store register pair DE direct.

(word) ← (E), (word + 1) ← (D)



Instruction Code: 702EH, low, high

The contents of E are stored in memory at location word, and the contents of D are stored in memory at location word + 1.

Bytes: 4

T-States: 20 (14)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

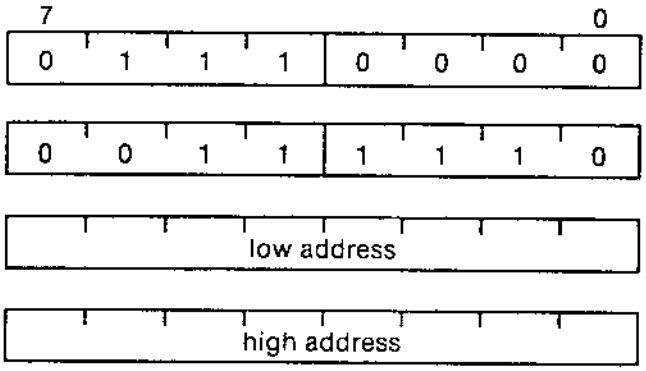
Example:

```
SDED    1000H   ;E is stored at 1000H and D at
                ;1001H.
```

SHLD word

Store register pair HL direct.

(word) ← (L), (word + 1) ← (H)



Instruction Code: 703EH, low, high

The contents of L are stored in memory at location word, and the contents of H are stored in memory at location word + 1.

Bytes: 4

T-States: 20 (14)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

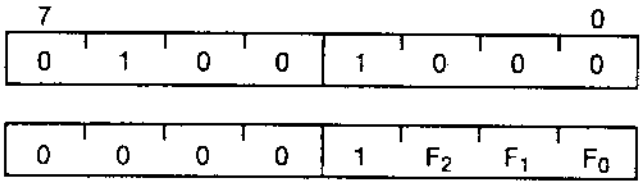
Example:

```
SHLD    1000H    ;L is stored at 1000H and H
                 ;at 1001H.
```

SK f

Skip next instruction if flag is set.

skip if f = 1



Instruction Code: 480(A-C)H

Skips the next instruction if flag f (CY, HC, or Z), designated by F_2-F_0 (2-4), is set to 1.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
SK
L0 ← 0
L1 ← 0

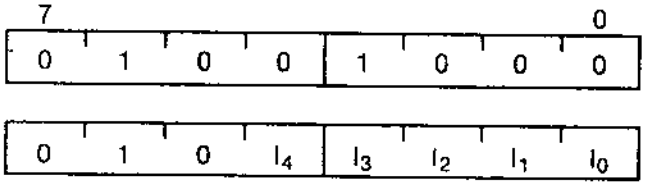
Example:

```
SK   Z      ;Skip if zero.
RET         ;This instruction skipped if Z flag set.
```

SKIT irf

Skip next instruction if interrupt flag is set.

skip if irf = 1; irf is reset.



Instruction Code:
484(0-C)H,
485(0-4)H

Skips the next instruction if interrupt request flag irf (INTFNMI, INTFT0, INTFT1, INTF1, INTF2, INTFE0, INTFE1, INTFEIN, INTFAD, INTFSR, INTFST, ER, OV, AN4, AN5, AN6, AN7, or SB) as designated by I_4-I_0 (0-C, 10-14), is set to 1. Reset the checked interrupt request flag. If the tested flag is 0, the next instruction is executed.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
SK
L0 ← 0
L1 ← 0

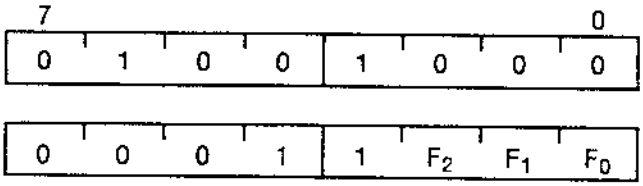
Example:

```
SKIT    AN4     ;Skip and reset interrupt flag
                ;AN4 if analog 4 flag is set (1).
. RET           ;This instruction skipped if
                ;AN4 flag set.
```

SKN f

Skip next instruction if flag is not set.

skip if f = 0



Instruction Code: 481(A-C)H

Skips the next instruction if flag f (CY, HC, or Z) designated by F_2-F_0 (2-4) is reset to 0.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
SK
L0 ← 0
L1 ← 0

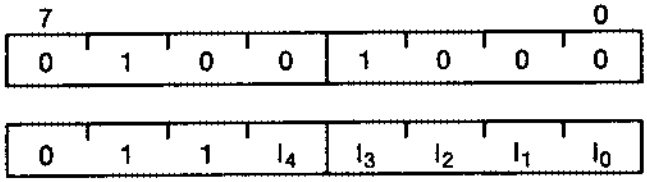
Example:

```
SKN     Z       ;Skip if not zero.
RET             ;This instruction skipped if Z
                ;flag cleared.
```

SKNIT irf

Skip next instruction if interrupt flag is not set.

skip if irf = 0; irf is reset.



Instruction Code:
486(0-C)H,
487(0-4)H

Skips the next instruction if interrupt request flag irf (INTFNMI, INTFT0, INTFT1, INTF1, INTF2, INTFE0, INTFE1, INTFEIN, INTFAD, INTFSR, INTFST, ER, OV, AN4, AN5, AN6, AN7, or SB) as designated by I_4-I_0 (0-C, 10-14), is reset to 0. If the flag is set to 1, it will be reset to 0 and not skip the next instruction.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
SK
L0 ← 0
L1 ← 0

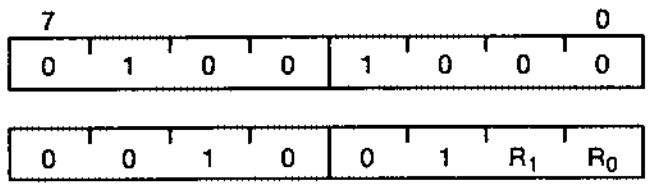
Example:

```
SKNIT   AN4     ;Skip if analog 4 interrupt flag
                ;AN4 is reset (0).
RET             ;This instruction skipped if
                ;AN4 is reset.
```

SLL r2

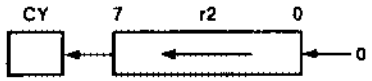
Shift register left one through carry.

$(r2_{m+1}) \leftarrow (r2_m)$, $(r2_0) \leftarrow 0$, $(CY) \leftarrow (r2_7)$



Instruction Code: 482 (5-7)H

Performs a logical left shift through carry with the contents of register r2 (A, B, or C) designated by R_1-R_0 (1-3). Register bit 0 is loaded with 0.



Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0
CY

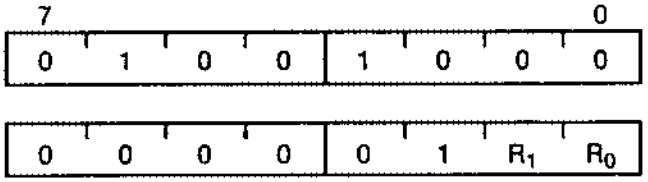
Example:

```
SLL     C       ;Shift left C through carry.
```

SLLC r2

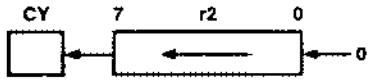
Shift register left one; skip next instruction if carry.

$(r2_{m+1}) \leftarrow (r2_m)$, $(r2_0) \leftarrow 0$, $(CY) \leftarrow (r2_7)$, SK/C



Instruction Code: 480(5-7)H

Performs a logical left shift through carry with the contents of register r2 (A, B, or C), designated by R_1-R_0 (1-3). If a carry is generated, the next instruction is skipped.



Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
SK
L0 ← 0
L1 ← 0
CY

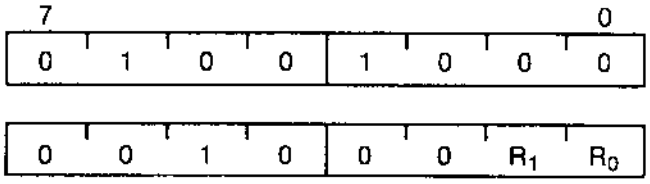
Example:

```
SLLC    C       ;Shift left C through carry.
RET             ;Skip this instruction if
                ;carry = 1 is generated.
```

SLR r2

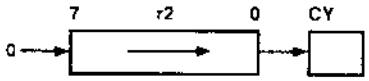
Shift register right one through carry.

$(r2_{m-1}) \leftarrow (r2_m)$, $(r2_7) \leftarrow 0$, $(CY) \leftarrow (r2_0)$



Instruction Code: 482(1-3)H

Performs a logical right shift through carry with the contents of register r2 (A, B, or C), designated by R_1-R_0 (1-3). Register bit 7 is loaded with zero.



Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
- SK ← 0
- L0 ← 0
- L1 ← 0
- CY

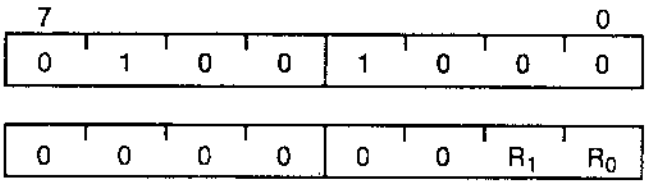
Example:

```
SLR     C       ;Rotate right C through carry.
```

SLRC r2

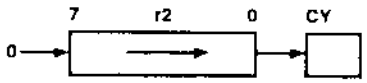
Shift register right one; skip next instruction if carry.

$(r2_{m-1}) \leftarrow (r2_m)$, $(r2_7) \leftarrow 0$, $(CY) \leftarrow (r2_0)$, SK/C



Instruction Code: 480(1-3)H

Performs a logical right shift through carry with the contents of register r2 (A, B, or C), designated by R_1-R_0 (1-3). If a carry is generated, the next instruction is skipped.



Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
- SK
- L0 ← 0
- L1 ← 0
- CY

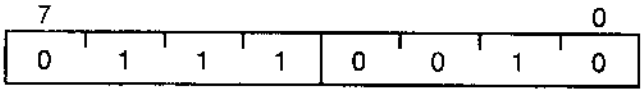
Example:

```
SLRC    C       ;Rotate right C through carry.
RET             ;Skip this instruction if carry
                ;true.
```

SOFTI

Software interrupt.

$(PC) \leftarrow (PC) + 1$, $(SP-1) \leftarrow (PSW)$, $(SP-2) \leftarrow (PCH)$, $(SP-3) \leftarrow (PCL)$, $(SP) \leftarrow (SP)-3$, $(PC) \leftarrow$ 0060H



Instruction Code: 72H

Performs a software interrupt. Stores the PSW and program counter on the stack. It then vectors to the address at location 60H in memory.

Bytes: 1

T-States: 16 (4)

Flag Bits Affected:
- SK ← 0
- L0 ← 0
- L1 ← 0

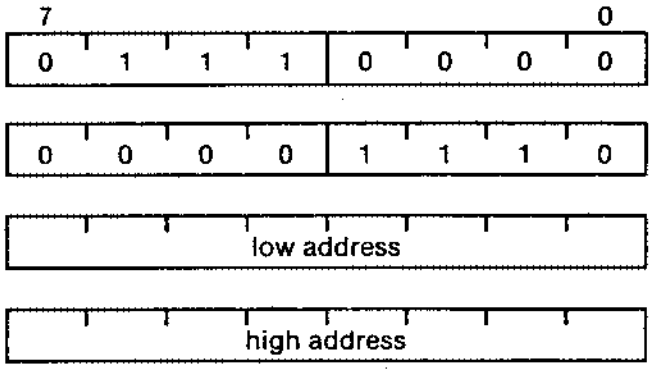
Example:

```
SOFTI           ;Vector to interrupt routine at
                ;location 0060H.
```

SSPD word

Store stack pointer direct.

(word) ← (SPL), (word + 1) ← (SPH)



Instruction Code: 700EH, low, high

The contents of SPL are stored in memory at location word, and SPH is stored in memory at location word + 1.

Bytes: 1

T-States: 20 (14)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

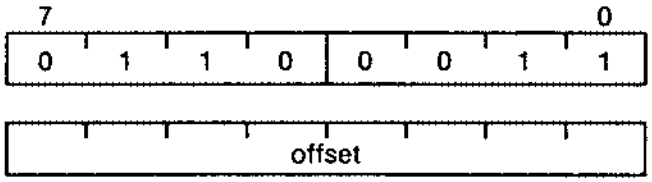
Example:

```
SSPD    1000H    ;SPL is stored at 1000H and
                 ;SPH at 1001H.
```

STAW wa

Store A to working register.

((V) • wa) ← (A)



Instruction Code: 63H, offset

Stores the contents of the accumulator in the working register as addressed by the V-register (8-bit high-order address) and the offset (8-bit low-order address).

Bytes: 2

T-States: 10 (7)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

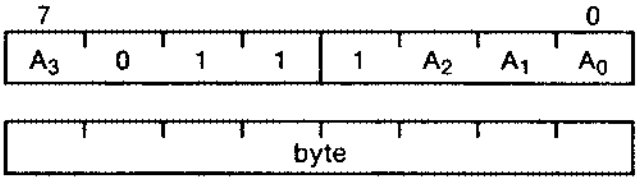
Example:

```
MVI     V,00H
STAW    10H      ;Store the accumulator in
                 ;working register 0010H.
```

STAX rpa2

Store A to memory addressed by register pair.

((rpa2)) ← (A)



Instruction Code:
3(9-F)H, byte
B(B-F)H, byte

Stores the contents of the accumulator in memory addressed by register pair rpa2 (BC, DE, HL, DE+, HL+, DE−, HL−, DE+byte, HL+A, HL+B, HL+EA, or HL+byte), as designated by A_3-A_0 (1-7, B-F).

If the autoincrement or decrement is designated, the contents of register pair (DE or HL) are automatically incremented or decremented by 1 after storing the accumulator.

If rpa2 is designated with DE+byte or HL+byte, the memory is addressed by the sum of the contents of DE or HL and the data byte of the instruction. Similarly the contents of HL and that of register A, B, or EA are added if HL+A, HL+B, or HL+EA is designated.

The number of bytes and T-states change with the designation of rpa2:

rpa2	B, D, H, D+, H+, D−, H−	D+byte	H+A, H+B, H+EA	H+byte
Bytes	1	2	1	2
States	7 (4)	13 (7)	13 (7)	13 (7)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

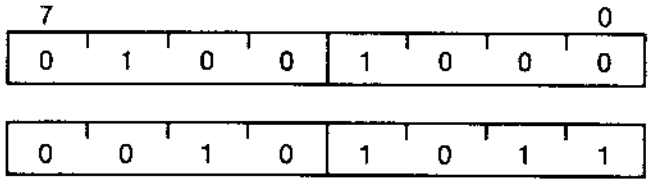
Example: To store A at memory locations 1000H and 1011H.

```
LXI     D,1000H  ;(DE) ← 1000H
STAX    D+       ;(1000H) ← (A),
                 ;(DE) ← 1001H
STAX    D+10H    ;(1011H) ← (A),
                 ;(DE) ← 1001H
```

STC

Set carry.

(CY) ← 1



Instruction Code: 482BH

Sets the carry flag to 1.

Bytes: 2

T-States: 8 (8)

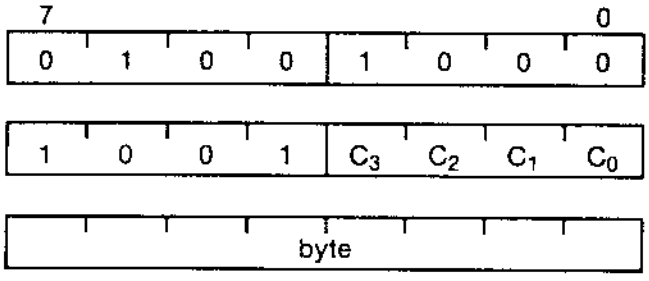
Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0
CY ← 1

Example:
STC

STEAX rpa3

Store EA to memory addressed by register pair.

((rpa3)) ← (EAL), ((rpa3) + 1) ← (EAH)



Instruction Code: 489(2-5, B-F)H, byte

Stores the low-order byte (EAL) of the extended accumulator in memory addressed by register pair rpa3 (DE, HL, DE++, HL++, DE+byte, HL+A, HL+B, HL+EA, or HL+byte), as designated by C_3-C_0 (2-5, B-F), and EAH in memory addressed by rpa3 + 1.

If DE+byte or HL+byte is designated as rpa3, the memory is addressed by the sum of the contents of the register pair and the data byte. If the destination memory is addressed by HL+A, HL+B, or HL+EA, it is addressed by the sum of the contents of HL and register (A, B, or EA).

The number of bytes and states change with the designation of rpa3:

rpa3	D, H, D++, H++	D+byte	H+A, H+B, H+EA	H+byte
Bytes	2	3	2	3
States	14 (8)	20 (11)	20 (11)	20 (11)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

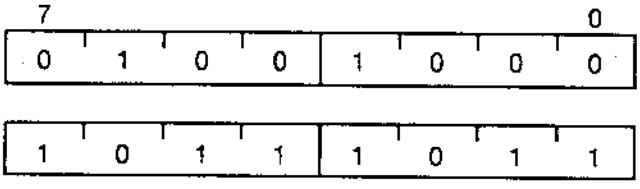
Example: To store EAL at 100H and 112H and EAH at 101H and 113H.

```
LXI     D,100H   ;(DE) ← 100H
STEAX   D++      ;(100H) ← (EAL),
                 ;(101H) ← (EAH),
                 ;(DE) ← (102)H
STEAX   D+10H    ;(112H) ← (EAL),
                 ;(113H) ← (EAH),
                 ;(DE) ← 102H
```

STOP

Stop oscillator (78C10, 78C11, and 78C14 only).

Stop oscillator



Instruction Code: 48BBH

The oscillator stops, the CPU stops executing instructions, and all on-chip peripherals stop. Only internal RAM retains its values and only the $\overline{NMI}$ and $\overline{RESET}$ inputs are recognized.

Bytes: 2

T-States: 12 (8)

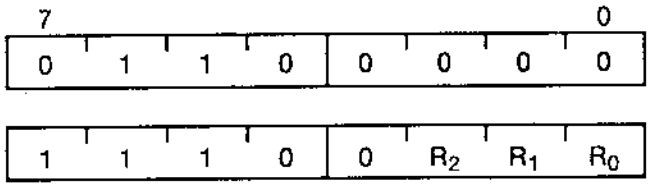
Flag Bits Affected: Undefined

Example:
```
STOP
```

SUB A,r

Subtract register from A.

(A) ← (A) − (r)



Instruction Code: 60E(0-7)H

Subtracts the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), from the accumulator and leaves the result in the accumulator.

Bytes: 2

T-States: 8 (8)

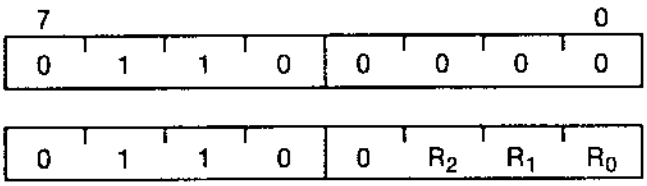
Flag Bits Affected:
- Z
- SK ← 0
- HC
- L0 ← 0
- L1 ← 0
- CY

Example:
```
SUB     A,C     ;Subtract C from A.
```

SUB r,A

Subtract A from register.

(r) ← (r) − (A)



Instruction Code: 606(0-7)H

Subtracts the contents of the accumulator from the contents of register r (V, A, B, C, D, E, H, or L) designated by R_2-R_0 (0-7). Leaves the result in the designated register.

Bytes: 2

T-States: 8 (8)

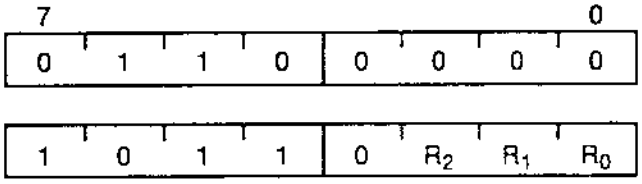
Flag Bits Affected:
- Z
- SK ← 0
- HC
- L0 ← 0
- L1 ← 0
- CY

Example:
```
MOV     A,C
SUB     C,A     ;C − A = 0 which clears CY
                ;and HC. Set the Z flag.
```

SUBNB A,r

Subtract register from A; skip next instruction if A greater than or equal to register.

(A) ← (A) − (r), SK/(A) ≥ (r)



Instruction Code: 60B(0-7)H

Subtracts the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), from the accumulator. Leaves the result in the accumulator. If no borrow is generated, (A) ≥ (r), the next instruction is skipped.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
- Z
- SK
- HC
- L0 ← 0
- L1 ← 0
- CY

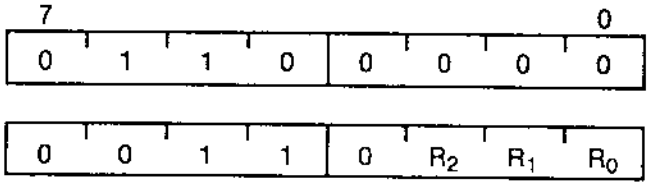
Example:

```
SUBNB   A,D
MOV     A,C     ;MOV instruction is skipped if
                ;no borrow is generated.
```

SUBNB r,A

Subtract A from register; skip next instruction if register greater than or equal to A.

(r) ← (r) − (A), SK/(r) ≥ (A)



Instruction Code: 603(0-7)H

Subtracts the contents of the accumulator from the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7), and leaves the result in the designated register. If no borrow is generated, (r) ≥ (A), the next instruction is skipped.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:

Z
SK
HC
L0 ← 0
L1 ← 0
CY

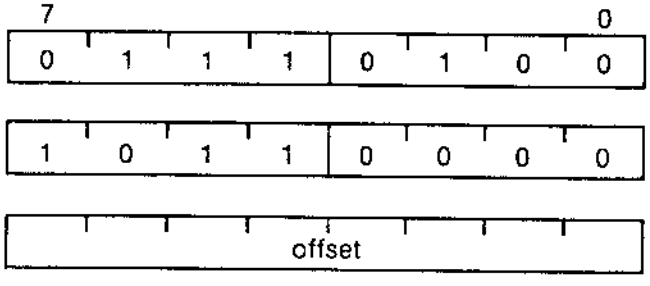
Example:

```
SUBNB   L,A     ;Sequence subtracts A from HL.
SUI     H,1
```

SUBNBW wa

Subtract working register from A; skip next instruction if A is greater than or equal to working register.

(A) ← (A) − ((V) • wa), SK/(A) ≥ ((V) • wa)



Instruction Code: 74B0H, offset

Subtracts the contents of a working register from the accumulator, and stores the result in the accumulator. If no borrow is generated, (A) ≥ ((V) • wa), the next instruction is skipped. The working register is addressed by the V-register (8 high-order address bits) and the offset (8 low-order address bits).

Bytes: 3

T-States: 14 (11)

Flag Bits Affected:

Z
SK
HC
L0 ← 0
L1 ← 0
CY

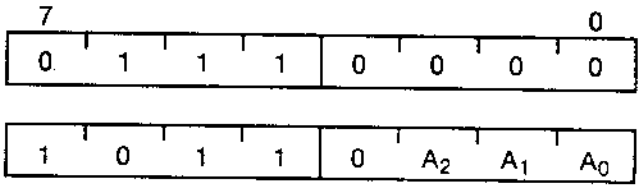
Example:

```
WORK    EQU     0E0H    ;WORK = 0E0H
LOCA    EQU     0       ;LOCA = 0
MVI     V,WORK          ;V = 0E0H
SUBNBW  LOCA            ;A = A − (0E000H)
RET                     ;This instruction is
                        ;skipped if (A)
                        ;≥ (E000H).
```

SUBNBX rpa

Subtract memory addressed by register pair from A; skip next instruction if A is greater than or equal to memory.

(A) ← (A) − ((rpa)), SK/(A) ≥ ((rpa))



Instruction Code: 70B(1-7)H

Subtracts the contents of memory from the accumulator and stores the result in the accumulator. If no borrow is generated, (A) ≥ ((rpa)), the next instruction is skipped. Memory is addressed by register pair rpa (BC, DE, HL, DE+, HL+, DE−, or HL−), designated by A_2-A_0 (1-7).

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

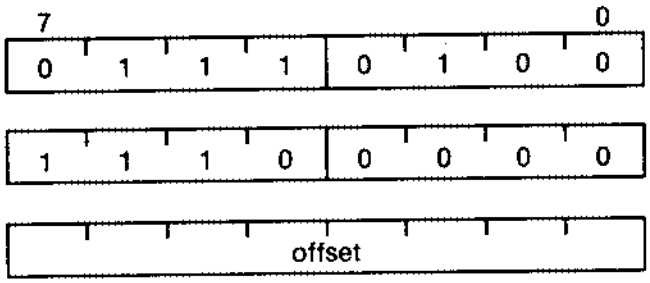
Example:

```
SUBNBX B      ;Subtract memory addressed
              ;by BC from accumulator.
RET           ;This instruction is skipped if
              ;(A) ≥ ((BC))
```

SUBW wa

Subtract working register from A.

(A) ← (A) − ((V) • wa)



Instruction Code: 74E0H, offset

Subtracts the contents of a working register from the accumulator and leaves the result in the accumulator. The working register is addressed by the V-register (8 high-order address bits) and offset (8 low-order address bits).

Bytes: 3

T-States: 14 (11)

Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

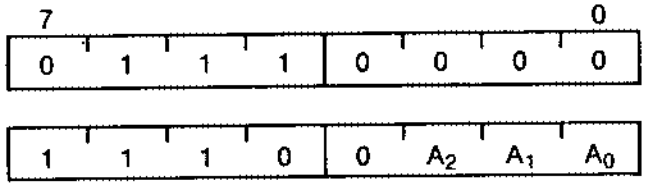
Example:

```
MVI     V,80H
SUBW    10H     ;Subtract the working register
                ;8010H from the accumulator.
```

SUBX rpa

Subtract memory addressed by register pair from A.

(A) ← (A) − ((rpa))



Instruction Code: 70E(1-7)H

Subtracts the contents of memory from the accumulator and leaves the result in the accumulator. The memory is addressed by register pair rpa (BC, DE, HL, DE+, HL+, DE−, or HL−), as designated by A_2-A_0 (1-7).

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

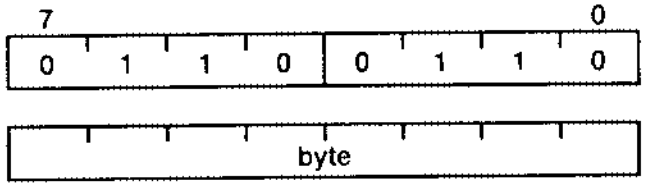
Example:

```
SUBX    D       ;Subtract memory pointed to
                ;by DE from the accumulator.
```

SUI A,byte

Subtract immediate data byte from A.

(A) ← (A) − byte



Instruction Code: 66H, byte

Subtracts the immediate data byte from the accumulator and leaves the result in the accumulator.

Bytes: 2

T-States: 7 (7)

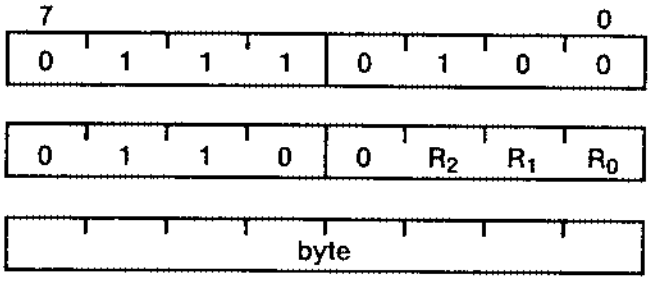
Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

Example:
```
SUI     A,20H    ;Subtract 20H from the
                 ;accumulator.
```

SUI r,byte

Subtract immediate data byte from register.

(r) ← (r) − byte



Instruction Code: 746(0-7)H, byte

Subtracts the immediate data byte from the contents of register r (V, A, B, C, D, E, H, or L) and leaves the result stored in the register. The register is designated by R_2-R_0 (0-7).

Bytes: 3

T-States: 11 (11)

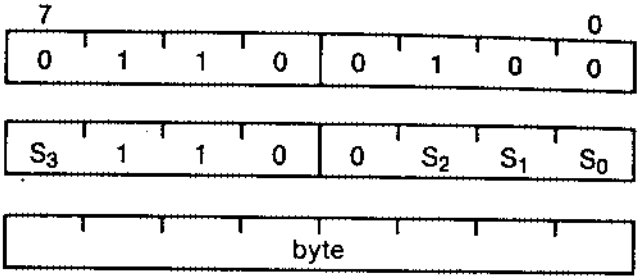
Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

Example:
```
SUI     D,20H    ;Subtract 20H from register D.
```

SUI sr2,byte

Subtract immediate data byte from special register.

(sr2) ← (sr2) − byte



Instruction Code:
646(0-3, 5-7)H, byte
64E(0, 1, 3, 5)H, byte

Subtracts the immediate data byte from the contents of special register sr2 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, or TMM). The register is designated by S_3-S_0 (0-3, 5-9, B, D). The result is stored in the designated special register.

Bytes: 3

T-States: 20 (11)

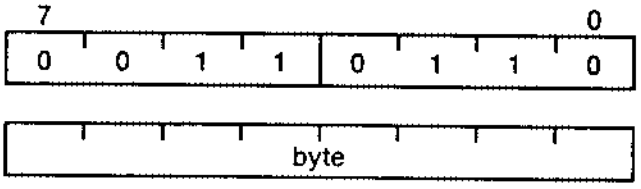
Flag Bits Affected:
Z
SK ← 0
HC
L0 ← 0
L1 ← 0
CY

Example:
```
SUI     PD,20H   ;Subtract 20H from port D and
                 ;leave result in port D.
```

SUINB A,byte

Subtract immediate data byte from A; skip next instruction if A is greater than or equal to data byte.

(A) ← (A) − byte, SK/(A) ≥ byte



Instruction Code: 36H, byte

Subtracts the immediate data byte from the contents of the accumulator and stores the result in the accumulator. If no borrow is generated, (A) ≥ byte, the next instruction is skipped.

Bytes: 2

T-States: 7 (7)

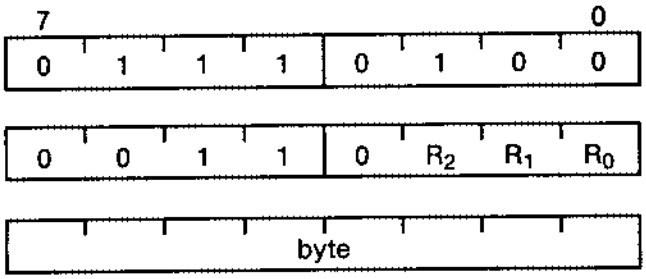
Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

Example:
```
SUINB   A,20H   ;Subtract 20H from A.
RET             ;Skip this instruction if no
                ;borrow is generated.
```

SUINB r,byte

Subtract immediate data byte from register; skip next instruction if register is greater than or equal to byte.

(r) ← (r) − byte, SK/(r) ≥ byte



Instruction Code: 743(0-7)H, byte

Subtracts the immediate data byte from the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7). The result is stored in the register. If no borrow is generated, (r) ≥ byte, the next instruction is skipped.

Bytes: 3

T-States: 11 (11)

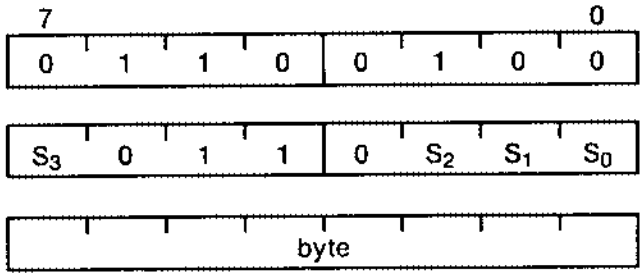
Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

Example:
```
SUINB   L,20H   ;Subtract 20H from L.
SUI     H,1     ;Subtract 1 from H if borrow
                ;from previous instruction.
```

SUINB sr2,byte

Subtract immediate data byte from special register; skip next instruction if special register greater than or equal to byte.

(sr2) ← (sr2) − byte, SK/(sr2) ≥ byte



Instruction Code:
643(0-3, 5-7)H, byte
64B(0, 1, 3, 5)H, byte

Subtracts the immediate data byte from the contents of special register sr2 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, or TMM), designated by S_3-S_0 (0-3, 5-9, B, D). Stores the result in the designated special register. Skips the next instruction if no borrow is generated: (sr2) ≥ byte.

Bytes: 3

T-States: 20 (11)

Flag Bits Affected:
Z
SK
HC
L0 ← 0
L1 ← 0
CY

Example:
```
SUINB   PD,20H  ;Subtract 20H from port D.
RET             ;Skip this instruction if no
                ;borrow is generated.
```

TABLE

Table look-up.

(C) ← ((PC) + 3 + (A)), (B) ← ((PC) + 3 + (A) + 1)

7							0
0	1	0	0	1	0	0	0
1	0	1	0	1	0	0	0

Instruction Code: 48A8H

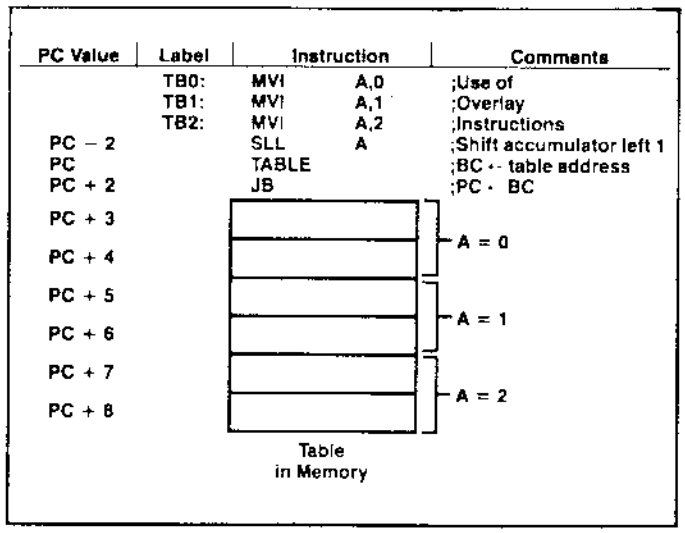
Loads the contents of a table addressed by ((PC) + 3 + (A)) into the C-register and ((PC) + 3 + (A) + 1) into the B-register.

Bytes: 2

T-States: 17 (8)

Flag Bits Affected:
SK ← 0
L0 ← 0
L1 ← 0

Example: Table location depends on value in A.



XRA A,r

Exclusive-OR register with A.

(A) ← (A) XOR (r)

7							0
0	1	1	0	0	0	0	0
1	0	0	1	0	R_2	R_1	R_0

Instruction Code: 609(0-7)H

Performs an exclusive-OR of the contents of the accumulator with the contents of register r (V, A, B, C, D, E, H, or L). The register is designated by R_2-R_0 (0-7). Stores the result in the accumulator.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

Example:

```
XRA     A,C     ;Exclusive-OR C with A.
```

XRA r,A

Exclusive-OR A with register.

(r) ← (r) XOR (A)

7							0
0	1	1	0	0	0	0	0
0	0	0	1	0	R_2	R_1	R_0

Instruction Code: 601(0-7)H

Performs an exclusive-OR between the contents of the accumulator and the contents of register r (V, A, B, C, D, E, H, or L). Register is designated by R_2-R_0 (0-7). Stores the result in the designated register.

Bytes: 2

T-States: 8 (8)

Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

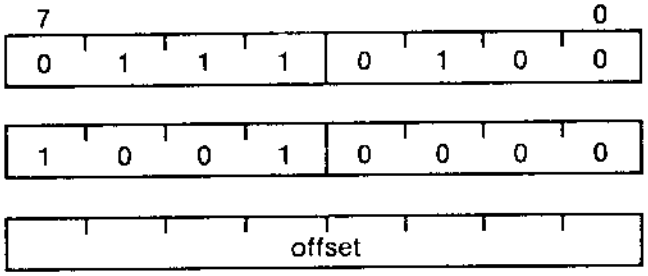
Example:

```
XRA     C,A     ;Exclusive-OR A with C.
```

XRAW wa

Exclusive-OR working register with A.

(A) ← (A) XOR ((V) • wa)



Instruction Code: 7490H, offset

Performs an exclusive-OR between the contents of a working register with the accumulator. Stores the result in the accumulator. The working register is addressed by the V-register (high-order 8 bits of address) and the offset (low-order 8 bits).

Bytes: 3

T-States: 14 (11)

Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

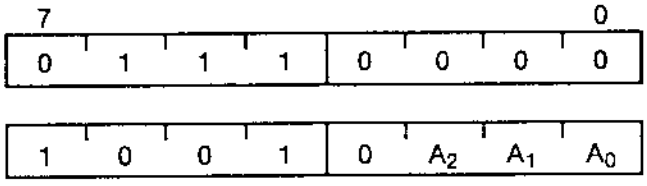
Example:

```
MVI     V,0FFH
XRAW    10H     ;Exclusive-OR working
                ;register located at FF10H with
                ;the accumulator.
```

XRAX rpa

Exclusive-OR memory addressed by register pair with A.

(A) ← (A) XOR ((rpa))



Instruction Code: 709(1-7)H

Performs logical exclusive-OR between the contents of memory that is addressed by register pair rpa (BC, DE, HL, DE+, HL+, DE−, or HL−), designated by A_2-A_0 (1-7), and the accumulator. The result is stored in the accumulator.

Bytes: 2

T-States: 11 (8)

Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

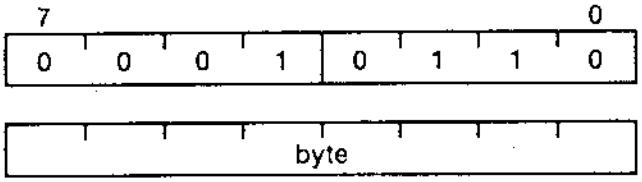
Example:

```
XRAX    D       ;Exclusive-OR memory
                ;pointed to by DE with the
                ;accumulator.
```

XRI A,byte

Exclusive-OR immediate data byte with A.

(A) ← (A) XOR byte



Instruction Code: 16H, byte

Performs an exclusive-OR with the accumulator and the immediate data byte. The result is stored in the accumulator.

Bytes: 2

T-States: 7 (7)

Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

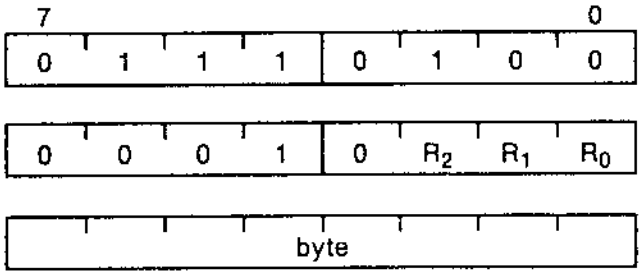
Example:

```
XRI     A,80H   ;Complements bit 7 of
                ;accumulator.
```

XRI r,byte

Exclusive-OR immediate data byte with register.

(r) ← (r) XOR byte



Instruction Code: 741(0-7)H, byte

Performs an exclusive-OR with the immediate data byte and the contents of register r (V, A, B, C, D, E, H, or L), designated by R_2-R_0 (0-7). The result is stored in the register.

Bytes: 3

T-States: 11 (11)

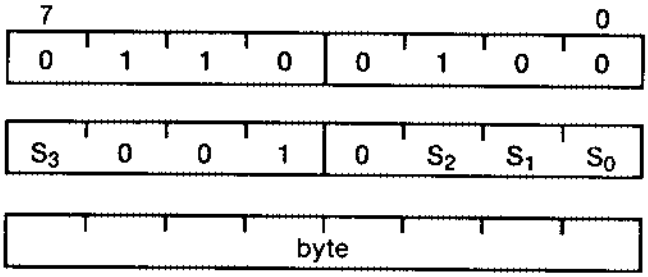
Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
XRI     D, 20H ;Complements bit 5 of register D.
```

XRI sr2,byte

Exclusive-OR immediate data byte with special register.

(sr2) ← (sr2) XOR byte



Instruction Code:
641(0-3, 5-7)H, byte
649(0, 1, 3, 5)H, byte

Performs an exclusive-OR of the immediate data byte with the contents of special register sr2 (PA, PB, PC, PD, PF, MKH, MKL, ANM, SMH, EOM, or TMM), designated by S_3-S_0 (0-3, 5-9, B, D). The result is stored in the designated special register.

Bytes: 3

T-States: 20 (11)

Flag Bits Affected:
Z
SK ← 0
L0 ← 0
L1 ← 0

Example:
```
XRI     PD,0FFH ;Result is the complement of
                ;value in port D.
```

Instruction		Description	Skip Condition
ACI	A,byte	Add immediate data byte to A with carry.	
ACI	r,byte	Add immediate data byte to register with carry.	
ACI	sr2,byte	Add immediate data byte to special register with carry.	
ADC	A,r	Add register to A with carry.	
ADC	r,A	Add A to register with carry.	
ADCW	wa	Add working register to A with carry.	
ADCX	rpa	Add memory addressed by register pair to A with carry.	
ADD	A,r	Add register to A.	
ADD	r,A	Add A to register.	
ADDNC	A,r	Add register to A and skip next instruction if no carry.	SK/NC
ADDNC	r,A	Add A to register and skip next instruction if no carry.	SK/NC
ADDNCW	wa	Add working register to A and skip next instruction if no carry.	SK/NC
ADDNCX	rpa	Add memory addressed by register pair to A and skip next instruction if no carry.	SK/NC
ADDW	wa	Add working register to A.	
ADDX	rpa	Add memory to A.	
ADI	A,byte	Add immediate data byte to A.	
ADI	r,byte	Add immediate data byte to register.	
ADI	sr2,byte	Add immediate data byte to special register.	
ADINC	A,byte	Add immediate data byte to A and skip next instruction if no carry.	SK/NC
ADINC	r,byte	Add immediate data byte to register and skip next instruction if no carry.	SK/NC
ADINC	sr2,byte	Add immediate data byte to special register and skip next instruction if no carry.	SK/NC
ANA	A,r	AND register with A.	
ANA	r,A	AND A with register.	
ANAW	wa	AND working register with A.	
ANAX	rpa	AND memory addressed by register pair with A.	
ANI	A,byte	AND immediate data byte with A.	
ANI	r,byte	AND immediate data byte with register.	
ANI	sr2,byte	AND immediate data byte with special register.	
ANIW	wa,byte	AND immediate data byte with working register.	
BIT	bit,wa	Bit test working register and skip next instruction if set.	SK = 1
BLOCK		Block data transfer.	
CALB		Call subroutine using BC indirect.	
CALF	word	Call subroutine in fixed area.	
CALL	word	Call subroutine direct.	
CALT	byte	Call table address.	
CLC		Clear carry.	
DAA		Decimal adjust A.	
DADC	EA,rp3	Add contents of register pair to EA with carry.	
DADD	EA,rp3	Add register pair to EA.	
DADDNC	EA,rp3	Add register pair to EA and skip next instruction if no carry.	SK/NC

Instruction		Description	Skip Condition
DAN	EA,rp3	AND register pair with EA.	
DCR	r2	Decrement register and skip next instruction if borrow.	SK/B
DCRW	wa	Decrement working register and skip next instruction if borrow.	SK/B
DCX	EA	Decrement EA.	
DCX	rp	Decrement register pair.	
DEQ	EA,rp3	Skip next instruction if register pair equals EA.	SK/EQ
DGT	EA,rp3	Skip the next instruction if EA greater than register pair.	SK/(EA) > (rp3)
DI		Disable interrupt.	
DIV	r2	Divide EA by register.	
DLT	EA,rp3	Skip next instruction if register pair greater than EA.	SK/(rp3) > (EA)
DMOV	EA,rp3	Move register pair to EA.	
DMOV	EA,sr4	Move special register to EA.	
DMOV	rp3,EA	Move EA to register pair.	
DMOV	sr3,EA	Move EA to special register.	
DNE	EA,rp3	Skip next instruction if register pair not equal to EA.	SK/(EA) ≠ (rp3)
DOFF	EA,rp3	Skip next instruction if off-test register pair with EA is zero.	SK/Z
DON	EA,rp3	Skip next instruction if on-test (AND) of register pair with EA is not zero.	SK/NZ
DOR	EA,rp3	OR register pair with EA.	
DRLL	EA	Rotate EA logical left one with carry.	
DRLR	EA	Rotate EA logical right one with carry.	
DSBB	EA,rp3	Subtract register pair from EA with borrow.	
DSLL	EA	Shift EA logical left one into carry.	
DSLR	EA	Shift EA logical right one into carry.	
DSUB	EA,rp3	Subtract register pair from EA.	
DSUBNB	EA,rp3	Subtract register pair from EA and skip next instruction if EA is greater than or equal to register pair.	SK/(EA) ≥ (rp3)
DXR	EA,rp3	Exclusive-OR register pair with EA.	
EADD	EA,r2	Add register to EA.	
EI		Enable interrupt.	
EQA	A,r	Skip next instruction if register equal to accumulator.	SK/EQ
EQA	r,A	Skip next instruction if accumulator equal to register.	SK/EQ
EQAW	wa	Skip next instruction if working register equal to accumulator.	SK/EQ
EQAX	rpa	Skip next instruction if memory addressed by register pair equal to accumulator.	SK/EQ
EQI	A,byte	Skip next instruction if immediate data byte equal to accumulator.	SK/EQ
EQI	r,byte	Skip next instuction if immediate data byte equal to register.	SK/EQ
EQI	sr2,byte	Skip next instruction if immediate data byte equal to special register.	SK/EQ
EQIW	wa,byte	Skip next instruction if immediate data byte equal to working register.	SK/EQ
ESUB	EA,r2	Subtract register from EA.	
EXA		Exchange V, A, and EA with V', A', and EA'.	
EXH		Exchange HL with H'L'.	
EXX		Exchange register sets BC, DE, HL with B'C', D'E', H'L'	

Instruction		Description	Skip Condition
GTA	A,r	Skip next instruction if A greater than register.	SK/(A) > (r)
GTA	r,A	Skip next instruction if register greater than A.	SK/(r) > (A)
GTAW	wa	Skip next instruction if A greater than working register.	SK/A > ((V)•wa)
GTAX	rpa	Skip next instruction if accumulator greater than memory addressed by register pair.	SK/(A) > MEM
GTI	A,byte	Skip next instruction if accumulator greater than immediate data byte.	SK/(A) > byte
GTI	r,byte	Skip next instruction if register greater than immediate data byte.	SK/(r) > byte
GTI	sr2,byte	Skip next instruction if special register greater than immediate data byte.	SK/(sr2) > byte
GTIW	wa,byte	Skip next instruction if working register greater than immediate data byte.	SK/((V)•wa) > byte
HLT		Halt.	
INR	r2	Increment register and skip next instruction if carry.	SK/CY
INRW	wa	Increment working register and skip next instruction if carry.	SK/CY
INX	EA	Increment EA.	
INX	rp	Increment register pair.	
JB		Jump BC indirect.	
JEA		Jump EA indirect.	
JMP	word	Jump direct.	
JR	word	Jump relative.	
JRE	word	Jump relative extended.	
LBCD	word	Load register pair BC direct.	
LDAW	wa	Load A with working register.	
LDAX	rpa2	Load A with memory addressed by register pair.	
LDEAX	rpa3	Load EA with contents of memory addressed by register pair.	
LDED	word	Load register pair DE direct.	
LHLD	word	Load register pair HL direct.	
LSPD	word	Load SP register direct.	
LTA	A,r	Skip next instruction if register greater than A.	SK/(r) > (A)
LTA	r,A	Skip next instruction if A greater than register.	SK/(A) > (r)
LTAW	wa	Skip next instruction if working register greater than A.	SK/((V)•wa) > (A)
LTAX	rpa	Skip next instruction if memory addressed by register pair greater than accumulator.	SK/((rpa)) > (A)
LTI	A,byte	Skip next instruction if immediate data byte greater than A.	SK/byte > (A)
LTI	r,byte	Skip next instruction if immediate data byte greater than register.	SK/byte > (r)
LTI	sr2,byte	Skip next instruction if immediate data byte greater than special register.	SK/byte > (sr2)
LTIW	wa,byte	Skip next instruction if immediate data byte greater than working register.	SK/byte > ((V)•wa)
LXI	rp2,word	Load register pair with immediate data	
MOV	r1,A	Move A to register.	
MOV	A,r1	Move register to A.	

Instruction		Description	Skip Condition
MOV	sr,A	Move A to special register.	
MOV	A,sr1	Move special register to A.	
MOV	r,word	Move memory to register.	
MOV	word,r	Move register to memory.	
MUL	r2	Multiply A by register.	
MVI	sr2,byte	Move immediate data byte to special register.	
MVI	r,byte	Move immediate data byte to register.	
MVIW	wa,byte	Move immediate data byte to working register.	
MVIX	rpa1,byte	Move immediate data byte to memory.	
NEA	A,r	Skip next instruction if register not equal to A.	SK/(r) ≠ (A)
NEA	r,A	Skip next instruction if A not equal to register.	SK/(A) ≠ (r)
NEAW	wa	Skip next instruction if A not equal to working register.	SK/(A) ≠ ((V)•wa)
NEAX	rpa	Skip next instruction if A not equal to memory addressed by register pair.	SK/(A) ≠ ((rpa))
NEGA		Negate A.	
NEI	A,byte	Skip next instruction if A not equal to immediate data byte.	SK/(A) ≠ byte
NEI	r,byte	Skip next instruction if register not equal to immediate data byte.	SK/(r) ≠ byte
NEI	sr2,byte	Skip next instruction if special register not equal to immediate data byte.	SK/(sr2) ≠ byte
NEIW	wa,byte	Skip next instruction if working register not equal to immediate data byte.	SK/((V)•wa) ≠ byte
NOP		No operation.	
OFFA	A,r	Skip next instruction if off-test (AND) register with A is zero.	SK/Z
OFFAW	wa	Skip next instruction if off-test (AND) working register with A is zero.	SK/Z
OFFAX	rpa	Skip next instruction if off-test (AND) memory addressed by register pair with A is zero.	SK/Z
OFFI	A,byte	Skip next instruction if off-test (AND) immediate data byte with A is zero.	SK/Z
OFFI	r,byte	Skip next instruction if off-test (AND) immediate data byte with register is zero.	SK/Z
OFFI	sr2,byte	Skip next instruction if off-test (AND) immediate data byte with special register is zero.	SK/Z
OFFIW	wa,byte	Skip next instruction if off-test (AND) immediate data byte with working register is zero.	SK/Z
ONA	A,r	Skip next instruction if on-test (AND) register with A is not zero.	SK/NZ
ONAW	wa	Skip next instruction if on-test (AND) working register with A is not zero.	SK/NZ
ONAX	rpa	Skip next instruction if off-test (AND) memory addressed by register pair with A is not zero.	SK/NZ
ONI	A,byte	Skip next instruction if on-test (AND) immediate data byte with A is not zero.	SK/NZ
ONI	r,byte	Skip next instruction if on-test (AND) immediate data byte with register is not zero.	SK/NZ
ONI	sr2,byte	Skip next instruction if on-test (AND) immediate data byte with special register is not zero.	SK/NZ

Instruction		Description	Skip Condition
ONIW	wa,byte	Skip next instruction if on-test (AND) immediate data byte with working register is not zero.	SK/NZ
ORA	A,r	OR A with register.	
ORA	r,A	OR register with A.	
ORAW	wa	OR working register with A.	
ORAX	rpa	OR memory addressed by register pair with A.	
ORI	A,byte	OR immediate data byte with A.	
ORI	r,byte	OR immediate data byte with register.	
ORI	sr2,byte	OR immediate data byte with special register.	
ORIW	wa,byte	OR immediate data byte with working register.	
POP	rp1	Pop register pair off stack.	
PUSH	rp1	Push register pair on stack.	
RET		Return from subroutine.	
RETI		Return from interrupt.	
RETS		Return from subroutine and skip next instruction.	
RLD		Rotate left digit.	
RLL	r2	Rotate register left one through carry.	
RLR	r2	Rotate register right one through carry.	
RRD		Rotate right digit.	
SBB	A,r	Subtract register from A with borrow.	
SBB	r,A	Subtract A from register with borrow.	
SBBW	wa	Subtract working register from A with borrow.	
SBBX	rpa	Subtract memory addressed by register pair from A with borrow.	
SBCD	word	Store register pair BC direct.	
SBI	A,byte	Subtract immediate data byte from A with borrow.	
SBI	r,byte	Subtract immediate data byte from register with borrow.	
SBI	sr2,byte	Subtract immediate data byte from special register with borrow.	
SDED	word	Store register pair DE direct.	
SHLD	word	Store register pair HL direct.	
SK	f	Skip next instruction if flag is set.	SK/f = 1
SKIT	irf	Skip next instruction if interrupt flag is set.	SK/irf = 1
SKN	f	Skip next instruction if no flag is set.	SK/f = 0
SKNIT	irf	Skip next instruction if no interrupt flag is set.	SK/irf = 0
SLL	r2	Shift register left one through carry.	
SLLC	r2	Shift register left one; skip next instruction if carry.	SK/C
SLR	r2	Shift register right one through carry.	
SLRC	r2	Shift register right one; skip next instruction if carry.	SK/C
SOFTI		Software interrupt.	
SSPD	word	Store stack pointer direct.	
STAW	wa	Store A to working register.	
STAX	rpa2	Store A to memory addressed by register pair.	

Instruction		Description	Skip Condition
STC		Set carry.	
STEAX	rpa3	Store EA to memory addressed by register pair.	
STOP		Stop oscillator.	
SUB	A,r	Subtract register from A.	
SUB	r,A	Subtract A from register.	
SUBNB	A,r	Subtract register from A; skip next instruction if A is greater than or equal to register.	SK/(A) ≥ (r)
SUBNB	r,A,	Subtract A from register; skip next instruction if register is greater than or equal to A.	SK/(r) ≥ (A)
SUBNBW	wa	Subtract working register from A; skip next instruction if A is greater than or equal to working register.	SK/(A) ≥ ((V)•wa)
SUBNBX	rpa	Subtract memory addressed by register pair from A; skip next instruction if A is greater than or equal to memory.	SK/(A) ≥ ((rpa))
SUBW	wa	Subtract working register from A.	
SUBX	rpa	Subtract memory addressed by register pair from A.	
SUI	A,byte	Subtract immediate data byte from A.	
SUI	r,byte	Subtract immediate data byte from register.	
SUI	sr2,byte	Subtract immediate data byte from special register.	
SUINB	A,byte	Subtract immediate data byte from A; skip next instruction if A is greater than or equal to byte.	SK/(A) ≥ byte
SUINB	r,byte	Subtract immediate data byte from register; skip next instruction if register is greater than or equal to byte.	SK/(r) ≥ byte
SUINB	sr2,byte,	Subtract immediate data byte from special register; skip next instruction if special register is greater than or equal to byte.	SK/(sr2) ≥ byte
TABLE		Table look-up.	
XRA	A,r	Exclusive-OR register with A.	
XRA	r,A	Exclusive-OR A with register.	
XRAW	wa	Exclusive-OR working register with A.	
XRAX	rpa	Exclusive-OR memory addressed by register pair with A.	
XRI	A,byte	Exclusive-OR immediate data byte with A.	
XRI	r,byte	Exclusive-OR immediate data byte with register.	
XRI	sr2,byte	Exclusive-OR immediate data byte with special register.	

Instruction Set

Mnemonic	Operand	Operation	Operation Code B1 / B3	Operation Code B2 / B4	State (Note 1)	Bytes	Skip Condition
8-Bit Data Transfer							
MOV	r1,A	(r1) ← (A)	0 0 0 1 1 T_2 T_1 T_0		4	1	
	A, r1	(A) ← (r1)	0 0 0 0 1 T_2 T_1 T_0		4	1	
	*sr,A	(sr) ← (A)	0 1 0 0 1 1 0 1	1 1 S_5 S_4 S_3 S_2 S_1 S_0	10	2	
	*A,sr1	(A) ← (sr1)	0 1 0 0 1 1 0 0	1 1 S_5 S_4 S_3 S_2 S_1 S_0	10	2	
	r,word	(r) ← (word)	0 1 1 1 0 0 0 0	0 1 1 0 1 R_2 R_1 R_0	17	4	
			Low addr	High addr			
	word,r	(word) ← (r)	0 1 1 1 0 0 0 0	0 1 1 1 1 R_2 R_1 R_0	17	4	
			Low addr	High addr			
MVI	*r,byte	(r) ← byte	0 1 1 0 1 R_2 R_1 R_0	Data	7	2	
	sr2,byte	(sr2) ← byte	0 1 1 0 0 1 0 0	S_3 0 0 0 0 S_2 S_1 S_0	14	3	
			Data				
MVIW	*wa, byte	((V)•(wa)) ← byte	0 1 1 1 0 0 0 1	Offset	13	3	
			Data				
MVIX	*rpa1,byte	(rpa1) ← byte	0 1 0 0 1 0 A_1 A_0	Data	10	2	
STAW	*wa	((V)•(wa)) ← (A)	0 1 1 0 0 0 1 1	Offset	10	2	
LDAW	*wa	(A) ← ((V)•(wa))	0 0 0 0 0 0 0 1	Offset	10	2	
STAX	*rpa2	((rpa2)) ← (A)	A_3 0 1 1 1 A_2 A_1 A_0	Data (Note 2)	7/13 (Note 3)	2	
LDAX	*rpa2	(A) ← ((rpa2))	A_3 0 1 0 1 A_2 A_1 A_0	Data (Note 2)	7/13 (Note 3)	2	
EXX		(B) ↔ (B'), (C) ↔ (C'), (D) ↔ (D') (E) ↔ (E'), (H) ↔ (H'), (L) ↔ (L')	0 0 0 1 0 0 0 1		4	1	
EXA		(V) ↔ (V'), (A) ↔ (A'), (EA) ↔ (EA')	0 0 0 1 0 0 0 0		4	1	
EXH		(H) ↔ (H'), (L) ↔ (L')	0 1 0 1 0 0 0 0		4	1	
BLOCK		((DE)) ← ((HL)), (DE) ← (DE) + 1, (HL) ← (HL) + 1, (C) ← (C) − 1 End if borrow	0 0 1 1 0 0 0 1		13 x (C + 1)	1	
16-Bit Data Transfer							
DMOV	rp3, EA	($rp3_L$) ← (EAL), ($rp3_H$) ← (EAH)	1 0 1 1 0 1 P_1 P_0		4	1	
	EA,rp3	(EAL) ← ($rp3_L$), (EAH) ← ($rp3_H$)	1 0 1 0 0 1 P_1 P_0		4	1	

Notes:

(1) For the skip condition, the idle states are as follows:
- 1-byte instruction: 4 states
- 2-byte instruction: 8 states
- 3-byte instruction: 11 states
- 2-byte instruction (with *): 7 states
- 3-byte instruction (with *): 10 states
- 4-byte instruction: 14 states

(2) B2 (Data): rpa2 = D+byte or H+byte.

(3) Right side of slash (/) in states indicates case rpa2 or rpa3 = D+byte, H+A, H+B, H+EA, or H+byte.

(4) B3 (Data): rpa3 = D+byte or H+byte.

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code (7 6 5 4 3 2 1 0)								Operation Code (7 6 5 4 3 2 1 0)								State (Note 1)	Bytes	Skip Condition
			B1 / B3								B2 / B4										
16-Bit Data Transfer (cont)																					
DMOV	sr3, EA	(sr3) ← (EA)	0	1	0	0	1	0	0	0	1	1	0	1	0	0	1	U_0	14	2	
	EA,sr4	(EA) ← (sr4)	0	1	0	0	1	0	0	0	1	1	0	0	0	0	0	V_0	14	2	
SBCD	word	(word) ← (C), (word + 1) ← (B)	0	1	1	1	0	0	0	0	0	0	0	1	1	1	1	0	20	4	
			Low addr								High addr										
SDED	word	(word) ← (E), (word + 1) ← (D)	0	1	1	1	0	0	0	0	0	0	1	0	1	1	1	0	20	4	
			Low addr								High addr										
SHLD	word	(word) ← (L), (word + 1) ← (H)	0	1	1	1	0	0	0	0	0	0	1	1	1	1	1	0	20	4	
			Low addr								High addr										
SSPD	word	(word) ← (SP_L), (word + 1) ← (SP_H)	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	20	4	
			Low addr								High addr										
STEAX	rpa3	((rpa3)) ← (EAL), (((rpa3)) + 1)) ← (EAH)	0	1	0	0	1	0	0	0	1	0	0	1	C_3	C_2	C_1	C_0	14/20 (Note 3)	3	
			Data (Note 4)																		
LBCD	word	(C) ← (word), (B) ← (word + 1)	0	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	20	4	
			Low addr								High addr										
LDED	word	(E) ← (word), (D) ← (word + 1)	0	1	1	1	0	0	0	0	0	0	1	0	1	1	1	1	20	4	
			Low addr								High addr										
LHLD	word	(L) ← (word), (H) ← (word + 1)	0	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	20	4	
			Low addr								High addr										
LSPD	word	(SP_L) ← (word), (SP_H) ← (word + 1)	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	20	4	
			Low addr								High addr										
LDEAX	rpa3	(EAL) ← ((rpa3)), (EAH) ← (((rpa3) + 1))	0	1	0	0	1	0	0	0	1	0	0	0	C_3	C_2	C_1	C_0	14/20 (Note 3)	3	
			Data (Note 4)																		
PUSH	rp1	(((SP) − 1)) ← ($rp1_H$), (((SP) − 2)) ← ($rp1_L$), (SP) ← (SP) − 2	1	0	1	1	0	Q_2	Q_1	Q_0									13	1	
POP	rp1	($rp1_L$) ← ((SP)), ($rp1_H$) ← (((SP) + 1)), (SP) ← (SP) + 2	1	0	1	0	0	Q_2	Q_1	Q_0									10	1	
LXI	*rp2,word	(rp2) ← (word)	0	P_2	P_1	P_0	0	1	0	0	Low byte								10	3	
			High byte																		
TABLE		(C) ← (((PC) + 3 + (A))), (B) ← (((PC) + 3 + (A) + 1))	0	1	0	0	1	0	0	0	1	0	1	0	1	0	0	0	17	2	
8-Bit Arithmetic (Register)																					
ADD	A,r	(A) ← (A) + (r)	0	1	1	0	0	0	0	0	1	1	0	0	0	R_2	R_1	R_0	8	2	
	r,A	(r) ← (r) + (A)	0	1	1	0	0	0	0	0	0	1	0	0	0	R_2	R_1	R_0	8	2	
ADC	A,r	(A) ← (A) + (r) + (CY)	0	1	1	0	0	0	0	0	1	1	0	1	0	R_2	R_1	R_0	8	2	
	r,A	(r) ← (r) + (A) + (CY)	0	1	1	0	0	0	0	0	0	1	0	1	0	R_2	R_1	R_0	8	2	

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code B1 / B3 7	6	5	4	3	2	1	0	B2 / B4 7	6	5	4	3	2	1	0	State [Note 1]	Bytes	Skip Condition
8-Bit Arithmetic [Register] (cont)																					
ADDNC	A,r	(A) ← (A) + (r)	0	1	1	0	0	0	0	0	1	0	1	0	0	R_2	R_1	R_0	8	2	No carry
	r,A	(r) ← (r) + (A)	0	1	1	0	0	0	0	0	0	0	1	0	0	R_2	R_1	R_0	8	2	No carry
SUB	A,r	(A) ← (A) − (r)	0	1	1	0	0	0	0	0	1	1	1	0	0	R_2	R_1	R_0	8	2	
	r,A	(r) ← (r) − (A)	0	1	1	0	0	0	0	0	0	1	1	0	0	R_2	R_1	R_0	8	2	
SBB	A,r	(A) ← (A) − (r) − (CY)	0	1	1	0	0	0	0	0	1	1	1	1	0	R_2	R_1	R_0	8	2	
	r,A	(r) ← (r) − (A) − (CY)	0	1	1	0	0	0	0	0	0	1	1	1	0	R_2	R_1	R_0	8	2	
SUBNB	A,r	(A) ← (A) − (r)	0	1	1	0	0	0	0	0	1	0	1	1	0	R_2	R_1	R_0	8	2	No borrow
	r,A	(r) ← (r) − (A)	0	1	1	0	0	0	0	0	0	0	1	1	0	R_2	R_1	R_0	8	2	No borrow
ANA	A,r	(A) ← (A) ∧ (r)	0	1	1	0	0	0	0	0	1	0	0	0	1	R_2	R_1	R_0	8	2	
	r,A	(r) ← (r) ∧ (A)	0	1	1	0	0	0	0	0	0	0	0	0	1	R_2	R_1	R_0	8	2	
ORA	A,r	(A) ← (A) ∨ (r)	0	1	1	0	0	0	0	0	1	0	0	1	1	R_2	R_1	R_0	8	2	
	r,A	(r) ← (r) ∨ (A)	0	1	1	0	0	0	0	0	0	0	0	1	1	R_2	R_1	R_0	8	2	
XRA	A,r	(A) ← (A)∀(r)	0	1	1	0	0	0	0	0	1	0	0	1	0	R_2	R_1	R_0	8	2	
	r,A	(r) ← (r)∀(A)	0	1	1	0	0	0	0	0	0	0	0	1	0	R_2	R_1	R_0	8	2	
GTA	A,r	(A) − (r) − 1	0	1	1	0	0	0	0	0	1	0	1	0	1	R_2	R_1	R_0	8	2	No borrow
	r,A	(r) − (A) − 1	0	1	1	0	0	0	0	0	0	0	1	0	1	R_2	R_1	R_0	8	2	No borrow
LTA	A,r	(A) − (r)	0	1	1	0	0	0	0	0	1	0	1	1	1	R_2	R_1	R_0	8	2	Borrow
	r,A	(r) − (A)	0	1	1	0	0	0	0	0	0	0	1	1	1	R_2	R_1	R_0	8	2	Borrow
NEA	A,r	(A) − (r)	0	1	1	0	0	0	0	0	1	1	1	0	1	R_2	R_1	R_0	8	2	No zero
	r,A	(r) − (A)	0	1	1	0	0	0	0	0	0	1	1	0	1	R_2	R_1	R_0	8	2	No zero
EQA	A,r	(A) − (r)	0	1	1	0	0	0	0	0	1	1	1	1	1	R_2	R_1	R_0	8	2	Zero
	r,A	(r) − (A)	0	1	1	0	0	0	0	0	0	1	1	1	1	R_2	R_1	R_0	8	2	Zero
ONA	A,r	(A) ∧ (r)	0	1	1	0	0	0	0	0	1	1	0	0	1	R_2	R_1	R_0	8	2	No zero
OFFA	A,r	(A) ∧ (r)	0	1	1	0	0	0	0	0	1	1	0	1	1	R_2	R_1	R_0	8	2	Zero
8-Bit Arithmetic (Memory)																					
ADDX	rpa	(A) ← (A) + ((rpa))	0	1	1	1	0	0	0	0	1	1	0	0	0	A_2	A_1	A_0	11	2	
ADCX	rpa	(A) ← (A) + ((rpa)) + (CY)	0	1	1	1	0	0	0	0	1	1	0	1	0	A_2	A_1	A_0	11	2	
ADDNCX	rpa	(A) ← (A) + ((rpa))	0	1	1	1	0	0	0	0	1	0	1	0	0	A_2	A_1	A_0	11	2	No carry
SUBX	rpa	(A) ← (A) − ((rpa))	0	1	1	1	0	0	0	0	1	1	1	0	0	A_2	A_1	A_0	11	2	
SBBX	rpa	(A) ← (A) − ((rpa)) − (CY)	0	1	1	1	0	0	0	0	1	1	1	1	0	A_2	A_1	A_0	11	2	
SUBNBX	rpa	(A) ← (A) − ((rpa))	0	1	1	1	0	0	0	0	1	0	1	1	0	A_2	A_1	A_0	11	2	No borrow
ANAX	rpa	(A) ← (A) ∧ ((rpa))	0	1	1	1	0	0	0	0	1	0	0	0	1	A_2	A_1	A_0	11	2	
ORAX	rpa	(A) ← (A) ∨ ((rpa))	0	1	1	1	0	0	0	0	1	0	0	1	1	A_2	A_1	A_0	11	2	

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code B1 / B3								B2 / B4								State (Note 1)	Bytes	Skip Condition
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			
8-Bit Arithmetic (Memory) (cont)																					
XRAX	rpa	(A) ← (A) ∀ ((rpa))	0	1	1	1	0	0	0	0	1	0	0	1	0	A_2	A_1	A_0	11	2	
GTAX	rpa	(A) – ((rpa)) – 1	0	1	1	1	0	0	0	0	1	0	1	0	1	A_2	A_1	A_0	11	2	No borrow
LTAX	rpa	(A) – ((rpa))	0	1	1	1	0	0	0	0	1	0	1	1	1	A_2	A_1	A_0	11	2	Borrow
NEAX	rpa	(A) – ((rpa))	0	1	1	1	0	0	0	0	1	1	1	0	1	A_2	A_1	A_0	11	2	No zero
EQAX	rpa	(A) – ((rpa))	0	1	1	1	0	0	0	0	1	1	1	1	1	A_2	A_1	A_0	11	2	Zero
ONAX	rpa	(A) ∧ ((rpa))	0	1	1	1	0	0	0	0	1	1	0	0	1	A_2	A_1	A_0	11	2	No zero
OFFAX	rpa	(A) ∧ ((rpa))	0	1	1	1	0	0	0	0	1	1	0	1	1	A_2	A_1	A_0	11	2	Zero
Immediate Data																					
ADI	*A,byte	(A) ← (A) + byte	0	1	0	0	0	1	1	0	Data								7	2	
	r,byte	(r) ← (r) + byte	0	1	1	1	0	1	0	0	0	1	0	0	0	R_2	R_1	R_0	11	3	
			Data																		
	sr2, byte	(sr2) ← (sr2) + byte	0	1	1	0	0	1	0	0	S_3	1	0	0	0	S_2	S_1	S_0	20	3	
			Data																		
ACI	*A,byte	(A) ← (A) + byte + (CY)	0	1	0	1	0	1	1	0	Data								7	2	
	r,byte	(r) ← (r) + byte + (CY)	0	1	1	1	0	1	0	0	0	1	0	1	0	R_2	R_1	R_0	11	3	
			Data																		
	sr2,byte	(sr2) ← (sr2) + byte + (CY)	0	1	1	0	0	1	0	0	S_3	1	0	1	0	S_2	S_1	S_0	20	3	
			Data																		
ADINC	*A,byte	(A) ← (A) + byte	0	0	1	0	0	1	1	0	Data								7	2	No carry
	r,byte	(r) ← (r) + byte	0	1	1	1	0	1	0	0	0	0	1	0	0	R_2	R_1	R_0	11	3	No carry
			Data																		
	sr2,byte	(sr2) ← (sr2) + byte	0	1	1	0	0	1	0	0	S_3	0	1	0	0	S_2	S_1	S_0	20	3	No carry
			Data																		
SUI	*A,byte	(A) ← (A) – byte	0	1	1	0	0	1	1	0	Data								7	2	
	r,byte	(r) ← (r) – byte	0	1	1	1	0	1	0	0	0	1	1	0	0	R_2	R_1	R_0	11	3	
			Data																		
	sr2,byte	(sr2) ← (sr2) – byte	0	1	1	0	0	1	0	0	S_3	1	1	0	0	S_2	S_1	S_0	20	3	
			Data																		
SBI	*A,byte	(A) ← (A) – byte – (CY)	0	1	1	1	0	1	1	0	Data								7	2	
	r,byte	(r) ← (r) – byte – (CY)	0	1	1	1	0	1	0	0	0	1	1	1	0	R_2	R_1	R_0	11	3	
			Data																		
	sr2,byte	(sr2) ← (sr2) – byte – (CY)	0	1	1	0	0	1	0	0	S_3	1	1	1	0	S_2	S_1	S_0	20	3	
			Data																		

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code B1 / B3	Operation Code B2 / B4	State (Note 1)	Bytes	Skip Condition
Immediate Data (cont)							
SUINB	*A,byte	(A) ← (A) – byte	0 0 1 1 0 1 1 0	Data	7	2	No borrow
	r,byte	(r) ← (r) – byte	0 1 1 1 0 1 0 0	0 0 1 1 0 R_2 R_1 R_0	11	3	No borrow
			Data				
	sr2,byte	(sr2) ← (sr2) – byte	0 1 1 0 0 1 0 0	S_3 0 1 1 0 S_2 S_1 S_0	20	3	No borrow
			Data				
ANI	*A,byte	(A) ← (A) ∧ byte	0 0 0 0 0 1 1 1	Data	7	2	
	r,byte	(r) ← (r) ∧ byte	0 1 1 1 0 1 0 0	0 0 0 0 1 R_2 R_1 R_0	11	3	
			Data				
	sr2,byte	(sr2) ← (sr2) ∧ byte	0 1 1 0 0 1 0 0	S_3 0 0 0 1 S_2 S_1 S_0	20	3	
			Data				
ORI	*A,byte	(A) ← (A) ∨ byte	0 0 0 1 0 1 1 1	Data	7	2	
	r,byte	(r) ← (r) ∨ byte	0 1 1 1 0 1 0 0	0 0 0 1 1 R_2 R_1 R_0	11	3	
			Data				
	sr2,byte	(sr2) ← (sr2) ∨ byte	0 1 1 0 0 1 0 0	S_3 0 0 1 1 S_2 S_1 S_0	20	3	
			Data				
XRI	*A,byte	(A) ← (A) ⊻ byte	0 0 0 1 0 1 1 0	Data	7	2	
	r,byte	(r) ← (r) ⊻ byte	0 1 1 1 0 1 0 0	0 0 0 1 0 R_2 R_1 R_0	11	3	
			Data				
	sr2,byte	(sr2) ← (sr2) ⊻ byte	0 1 1 0 0 1 0 0	S_3 0 0 1 0 S_2 S_1 S_0	20	3	
			Data				
GTI	*A,byte	(A) – byte – 1	0 0 1 0 0 1 1 1	Data	7	2	No borrow
	r,byte	(r) – byte – 1	0 1 1 1 0 1 0 0	0 0 1 0 1 R_2 R_1 R_0	11	3	No borrow
			Data				
	sr2,byte	(sr2) – byte – 1	0 1 1 0 0 1 0 0	S_3 0 1 0 1 S_2 S_1 S_0	14	3	No borrow
			Data				
LTI	*A,byte	(A) – byte	0 0 1 1 0 1 1 1	Data	7	2	Borrow
	r,byte	(r) – byte	0 1 1 1 0 1 0 0	0 0 1 1 1 R_2 R_1 R_0	11	3	Borrow
			Data				
	sr2,byte	(sr2) – byte	0 1 1 0 0 1 0 0	S_3 0 1 1 1 S_2 S_1 S_0	14	3	Borrow
			Data				
NEI	*A,byte	(A) – byte	0 1 1 0 0 1 1 1	Data	7	2	No zero
	r,byte	(r) – byte	0 1 1 1 0 1 0 0	0 1 1 0 1 R_2 R_1 R_0	11	3	No zero
			Data				

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code B1 / B3								Operation Code B2 / B4								State (Note 1)	Bytes	Skip Condition
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			
Immediate Data (cont)																					
NEI	sr2,byte	(sr2) – byte	0	1	1	0	0	1	0	0	S_3	1	1	0	1	S_2	S_1	S_0	14	3	No zero
			Data																		
EQI	*A,byte	(A) – byte	0	1	1	1	0	1	1	1	Data								7	2	Zero
	r,byte	(r) – byte	0	1	1	1	0	1	0	0	0	1	1	1	1	R_2	R_1	R_0	11	3	Zero
			Data																		
	sr2,byte	(sr2) – byte	0	1	1	0	0	1	0	0	S_3	1	1	1	1	S_2	S_1	S_0	14	3	Zero
			Data																		
ONI	*A,byte	(A) ∧ byte	0	1	0	0	0	1	1	1	Data								7	2	No zero
	r,byte	(r) ∧ byte	0	1	1	1	0	1	0	0	0	1	0	0	1	R_2	R_1	R_0	11	3	No zero
			Data																		
	sr2,byte	(sr2) ∧ byte	0	1	1	0	0	1	0	0	S_3	1	0	0	1	S_2	S_1	S_0	14	3	No zero
			Data																		
OFFI	*A,byte	(A) ∧ byte	0	1	0	1	0	1	1	1	Data								7	2	Zero
	r,byte	(r) ∧ byte	0	1	1	1	0	1	0	0	0	1	0	1	1	R_2	R_1	R_0	11	3	Zero
			Data																		
	sr2,byte	(sr2) ∧ byte	0	1	1	0	0	1	0	0	S_3	1	0	1	1	S_2	S_1	S_0	14	3	Zero
			Data																		
Working Register																					
ADDW	wa	(A) ← (A) + ((V)•(wa))	0	1	1	1	0	1	0	0	1	1	0	0	0	0	0	0	14	3	
			Offset																		
ADCW	wa	(A) ← (A) + ((V)•(wa)) + (CY)	0	1	1	1	0	1	0	0	1	1	0	1	0	0	0	0	14	3	
			Offset																		
ADDNCW	wa	(A) ← (A) + ((V)•(wa))	0	1	1	1	0	1	0	0	1	0	1	0	0	0	0	0	14	3	No carry
			Offset																		
SUBW	wa	(A) ← (A) – ((V)•(wa))	0	1	1	1	0	1	0	0	1	1	1	0	0	0	0	0	14	3	
			Offset																		
SBBW	wa	(A) ← (A) – ((V)•(wa)) – (CY)	0	1	1	1	0	1	0	0	1	1	1	1	0	0	0	0	14	3	
			Offset																		
SUBNBW	wa	(A) ← (A) – ((V)•(wa))	0	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	14	3	No borrow
			Offset																		
ANAW	wa	(A) ← (A) ∧ ((V)•(wa))	0	1	1	1	0	1	0	0	1	0	0	0	1	0	0	0	14	3	
			Offset																		

NEC

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code B1 / B3								Operation Code B2 / B4								State (Note 1)	Bytes	Skip Condition
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			
Working Register (cont)																					
ORAW	wa	(A) ← (A) V ((V)•(wa))	0	1	1	1	0	1	0	0	1	0	0	1	1	0	0	0	14	3	
			Offset																		
XRAW	wa	(A) ← (A)⊻((V)•(wa))	0	1	1	1	0	1	0	0	1	0	0	1	0	0	0	0	14	3	
			Offset																		
GTAW	wa	(A) – ((V)•(wa)) – 1	0	1	1	1	0	1	0	0	1	0	1	0	1	0	0	0	14	3	No borrow
			Offset																		
LTAW	wa	(A) – ((V)•(wa))	0	1	1	1	0	1	0	0	1	0	1	1	1	0	0	0	14	3	Borrow
			Offset																		
NEAW	wa	(A) – ((V)•(wa))	0	1	1	1	0	1	0	0	1	1	1	0	1	0	0	0	14	3	No zero
			Offset																		
EQAW	wa	(A) – ((V)•(wa))	0	1	1	1	1	1	0	0	1	1	1	1	1	0	0	0	14	3	Zero
			Offset																		
ONAW	wa	(A) ∧ ((V)•(wa))	0	1	1	1	0	1	0	0	1	1	0	0	1	0	0	0	14	3	No zero
			Offset																		
OFFAW	wa	(A) ∧ ((V)•(wa))	0	1	1	1	0	1	0	0	1	1	0	1	1	0	0	0	14	3	Zero
			Offset																		
ANIW	*wa,byte	((V)•(wa)) ← ((V)•(wa)) ∧ byte	0	0	0	0	0	1	0	1	Offset								19	3	
			Data																		
ORIW	*wa,byte	((V)•(wa)) ← ((V)•(wa)) V byte	0	0	0	1	0	1	0	1	Offset								19	3	
			Data																		
GTIW	*wa,byte	((V)•(wa)) – byte – 1	0	0	1	0	0	1	0	1	Offset								13	3	No borrow
			Data																		
LTIW	*wa,byte	((V)•(wa)) – byte	0	0	1	1	0	1	0	1	Offset								13	3	Borrow
			Data																		
NEIW	*wa,byte	((V)•(wa)) – byte	0	1	1	0	0	1	0	1	Offset								13	3	No zero
			Data																		
EQIW	*wa,byte	((V)•(wa)) – byte	0	1	1	1	0	1	0	1	Offset								13	3	Zero
			Data																		
ONIW	*wa,byte	((V)•(wa)) ∧ byte	0	1	0	0	0	1	0	1	Offset								13	3	No zero
			Data																		
OFFIW	*wa,byte	((V)•(wa)) ∧ byte	0	1	0	1	0	1	0	1	Offset								13	3	Zero
			Data																		

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code B1/B3 7	6	5	4	3	2	1	0	Operation Code B2/B4 7	6	5	4	3	2	1	0	State (Note 1)	Bytes	Skip Condition
16-Bit Arithmetic																					
EADD	EA,r2	(EA) ← (EA) + (r2)	0	1	1	1	0	0	0	0	0	1	0	0	0	0	R_1	R_0	11	2	
DADD	EA,rp3	(EA) ← (EA) + (rp3)	0	1	1	1	0	1	0	0	1	1	0	0	0	1	P_1	P_0	11	2	
DADC	EA,rp3	(EA) ← (EA) + (rp3) + (CY)	0	1	1	1	0	1	0	0	1	1	0	1	0	1	P_1	P_0	11	2	
DADDNC	EA,rp3	(EA) ← (EA) + (rp3)	0	1	1	1	0	1	0	0	1	0	1	0	0	1	P_1	P_0	11	2	No carry
ESUB	EA,r2	(EA) ← (EA) − (r2)	0	1	1	1	0	0	0	0	0	1	1	0	0	0	R_1	R_0	11	2	
DSUB	EA,rp3	(EA) ← (EA) − (rp3)	0	1	1	1	0	1	0	0	1	1	1	0	0	1	P_1	P_0	11	2	
DSBB	EA,rp3	(EA) ← (EA) − (rp3) − (CY)	0	1	1	1	0	1	0	0	1	1	1	1	0	1	P_1	P_0	11	2	
DSUBNB	EA,rp3	(EA) ← (EA) − (rp3)	0	1	1	1	0	1	0	0	1	0	1	1	0	1	P_1	P_0	11	2	No borrow
DAN	EA,rp3	(EA) ← (EA) ∧ (rp3)	0	1	1	1	0	1	0	0	1	0	0	0	1	1	P_1	P_0	11	2	
DOR	EA,rp3	(EA) ← (EA) ∨ (rp3)	0	1	1	1	0	1	0	0	1	0	0	1	1	1	P_1	P_0	11	2	
DXR	EA,rp3	(EA) ← (EA) ⊻ (rp3)	0	1	1	1	0	1	0	0	1	0	0	1	0	1	P_1	P_0	11	2	
DGT	EA,rp3	(EA) − (rp3) − 1	0	1	1	1	0	1	0	0	1	0	1	0	1	1	P_1	P_0	11	2	No borrow
DLT	EA,rp3	(EA) − (rp3)	0	1	1	1	0	1	0	0	1	0	1	1	1	1	P_1	P_0	11	2	Borrow
DNE	EA,rp3	(EA) − (rp3)	0	1	1	1	0	1	0	0	1	1	1	0	1	1	P_1	P_0	11	2	No zero
DEQ	EA,rp3	(EA) − (rp3)	0	1	1	1	0	1	0	0	1	1	1	1	1	1	P_1	P_0	11	2	Zero
DON	EA,rp3	(EA) ∧ (rp3)	0	1	1	1	0	1	0	0	1	1	0	0	1	1	P_1	P_0	11	2	No zero
DOFF	EA,rp3	(EA) ∧ (rp3)	0	1	1	1	0	1	0	0	1	1	0	1	1	1	P_1	P_0	11	2	Zero
Multiply/Divide																					
MUL	r2	(EA) ← (A) x (r2)	0	1	0	0	1	0	0	0	0	0	1	0	1	1	R_1	R_0	32	2	
DIV	r2	(EA) ← (EA) ÷ (r2), (r2) ← Remainder	0	1	0	0	1	0	0	0	0	0	1	1	1	1	R_1	R_0	59	2	
Increment/Decrement																					
INR	r2	(r2) ← (r2) + 1	0	1	0	0	0	0	R_1	R_0									4	1	Carry
INRW	*wa	((V)•(wa)) ← ((V)•(wa)) + 1	0	0	1	0	0	0	0	0	Offset								16	2	Carry
INX	rp	(rp) ← (rp) + 1	0	0	P_1	P_0	0	0	1	0									7	1	
	EA	(EA) ← (EA) + 1	1	0	1	0	1	0	0	0									7	1	
DCR	r2	(r2) ← (r2) − 1	0	1	0	1	0	0	R_1	R_0									4	1	Borrow
DCRW	*wa	((V)•(wa)) ← ((V)•(wa)) − 1	0	0	1	1	0	0	0	0	Offset								16	2	Borrow
DCX	rp	(rp) ← (rp) − 1	0	0	P_1	P_0	0	0	1	1									7	1	
	EA	(EA) ← (EA) − 1	1	0	1	0	1	0	0	1									7	1	
Others																					
DAA		Decimal Adjust Accumulator	0	1	1	0	0	0	0	1									4	1	
STC		(CY) ← 1	0	1	0	0	1	0	0	0	0	0	1	0	1	0	1	1	8	2	
CLC		(CY) ← 0	0	1	0	0	1	0	0	0	0	0	1	0	1	0	1	0	8	2	

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code B1 / B3								Operation Code B2 / B4								State (Note 1)	Bytes	Skip Condition
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			
Others (cont)																					
NEGA		$(A) \leftarrow \overline{(A)} + 1$	0	1	0	0	1	0	0	0	0	0	1	1	1	0	1	0	8	2	
Rotate and Shift																					
RLD		Rotate left digit $(A_{3-0}) \leftarrow ((HL))_{7-4}$, $((HL))_{7-4} \leftarrow ((HL))_{3-0}$, $((HL))_{3-0} \leftarrow (A_{3-0})$	0	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	17	2	
RRD		Rotate right digit $(A_{3-0}) \leftarrow ((HL))_{3-0}$, $((HL))_{7-4} \leftarrow ((HL))_{3-0}$, $((HL))_{7-4} \leftarrow (A_{3-0})$	0	1	0	0	1	0	0	0	0	0	1	1	1	0	0	1	17	2	
RLL	r2	$(r2_{m+1}) \leftarrow (r2_m)$, $(r2_0) \leftarrow (CY)$, $(CY) \leftarrow (r2_7)$	0	1	0	0	1	0	0	0	0	0	1	1	0	1	R_1	R_0	8	2	
RLR	r2	$(r2_{m-1}) \leftarrow (r2_m)$, $(r2_7) \leftarrow (CY)$, $(CY) \leftarrow (r2_0)$	0	1	0	0	1	0	0	0	0	0	1	1	0	0	R_1	R_0	8	2	
SLL	r2	$(r2_{m+1}) \leftarrow (r2_m)$, $(r2_0) \leftarrow 0$, $(CY) \leftarrow (r2_7)$	0	1	0	0	1	0	0	0	0	0	1	0	0	1	R_1	R_0	8	2	
SLR	r2	$(r2_{m-1}) \leftarrow (r2_m)$, $(r2_7) \leftarrow 0$, $(CY) \leftarrow (r2_0)$	0	1	0	0	1	0	0	0	0	0	1	0	0	0	R_1	R_0	8	2	
SLLC	r2	$(r2_{m+1}) \leftarrow (r2_m)$, $(r2_0) \leftarrow 0$, $(CY) \leftarrow (r2_7)$	0	1	0	0	1	0	0	0	0	0	0	0	0	1	R_1	R_0	8	2	Carry
SLRC	r2	$(r2_{m-1}) \leftarrow (r2_m)$, $(r2_7) \leftarrow 0$, $(CY) \leftarrow (r2_0)$	0	1	0	0	1	0	0	0	0	0	0	0	0	0	R_1	R_0	8	2	Carry
DRLL	EA	$(EA_{n+1}) \leftarrow (EA_n)$, $(EA_0) \leftarrow (CY)$, $(CY) \leftarrow (EA_{15})$	0	1	0	0	1	0	0	0	1	0	1	1	0	1	0	0	8	2	
DRLR	EA	$(EA_{n-1}) \leftarrow (EA_n)$, $(EA_{15}) \leftarrow (CY)$, $(CY) \leftarrow (EA_0)$	0	1	0	0	1	0	0	0	1	0	1	1	0	0	0	0	8	2	
DSLL	EA	$(EA_{n+1}) \leftarrow (EA_n)$, $(EA_0) \leftarrow 0$, $(CY) \leftarrow (EA_{15})$	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	8	2	
DSLR	EA	$(EA_{n-1}) \leftarrow (EA_n)$, $(EA_{15}) \leftarrow 0$, $(CY) \leftarrow (EA_0)$	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	0	8	2	
Jump																					
JMP	*word	(PC) ← word	0	1	0	1	0	1	0	0	Low addr								10	3	
			High addr																		
JB		$(PC_H) \leftarrow (B)$, $(PC_L) \leftarrow (C)$	0	0	1	0	0	0	0	1									4	1	
JR	word	(PC) ← (PC) + 1 + jdisp 1	1	1	← jdisp1 →														10	1	
JRE	*word	(PC) ← (PC) + 2 + jdisp	0	1	0	0	1	1	1	← jdisp →									10	2	
JEA		(PC) ← (EA)	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0	8	2	
Call																					
CALL	*word	$((SP) - 1) \leftarrow ((PC) + 3)_H$, $((SP) - 2) \leftarrow ((PC) + 3)_L$, (PC) ← word, (SP) ← (SP) − 2	0	1	0	0	0	0	0	0	Low addr								16	3	
			High addr																		
CALB		$((SP) - 1) \leftarrow ((PC) + 2)_H$, $((SP) - 2) \leftarrow ((PC) + 2)_L$, $(PC_H) \leftarrow (B)$, $(PC_L) \leftarrow (C)$, (SP) ← (SP) − 2	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	1	17	2	

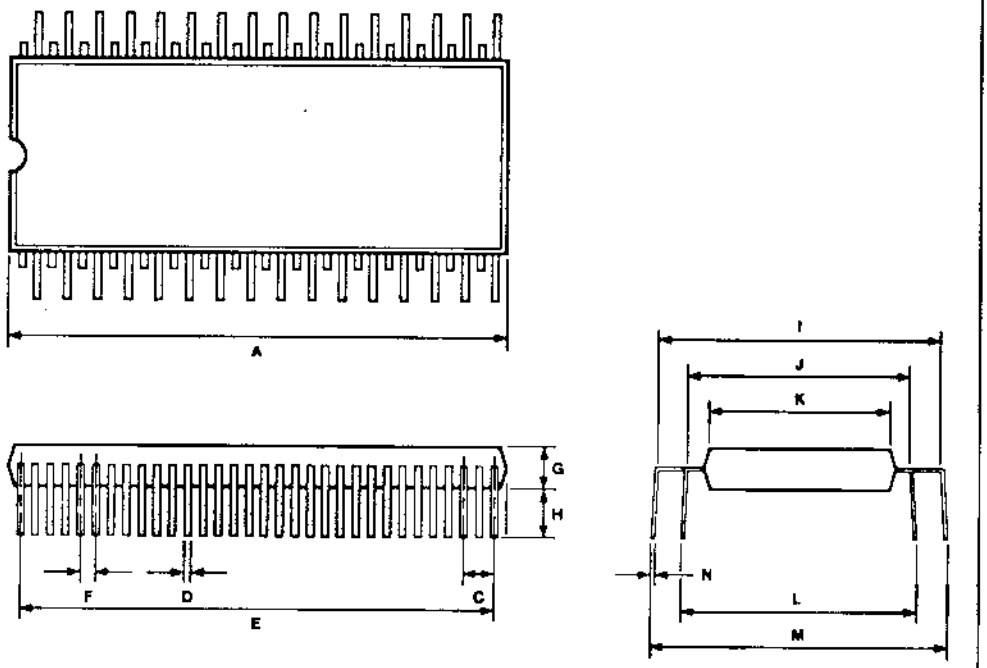
Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code B1 / B3								Operation Code B2 / B4								State (Note 1)	Bytes	Skip Condition
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			
Call (cont)																					
CALF	*word	$((SP)-1) \leftarrow ((PC)+2)_H$, $((SP)-2) \leftarrow ((PC)+2)_L$, $(PC_{15-11}) \leftarrow 00001$, $(PC_{10-0}) \leftarrow fa$, $(SP) \leftarrow (SP)-2$	0	1	1	1	1	←			fa							→	13	2	
CALT	word	$((SP)-1) \leftarrow ((PC)+1)_H$, $((SP)-2) \leftarrow ((PC)+1)_L$, $(PC_L) \leftarrow (128+2ta)$, $(PC_H) \leftarrow (129+2ta)$, $(SP) \leftarrow (SP)-2$	1	0	0	←		ta		→									16	1	
SOFTI		$((SP)-1) \leftarrow (PSW)$, $((SP)-2) \leftarrow ((PC)+1)_H$, $((SP)-3) \leftarrow ((PC)+1)_L$, $(PC) \leftarrow 0060H$, $(SP) \leftarrow (SP)-3$	0	1	1	1	0	0	1	0									16	1	
Return																					
RET		$(PC_L) \leftarrow ((SP))$, $(PC_H) \leftarrow ((SP)+1)$ $(SP) \leftarrow (SP)+2$	1	0	1	1	1	0	0	0									10	1	
RETS		$(PC_L) \leftarrow ((SP))$, $(PC_H) \leftarrow ((SP)+1)$ $(SP) \leftarrow (SP)+2$, $(PC) \leftarrow (PC)+n$	1	0	1	1	1	0	0	1									10	1	Unconditional Skip
RETI		$(PC_L) \leftarrow ((SP))$, $(PC_H) \leftarrow ((SP)+1)$ $(PSW) \leftarrow ((SP)+2)$, $(SP) \leftarrow (SP)+3$	0	1	1	0	0	0	1	0									13	1	
Skip																					
BIT	*bit, wa	Skip if ((V)•(wa)) bit = 1	0	1	0	1	1	B_2	B_1	B_0	←				Offset			→	10	2	Bit Test
SK	f	Skip if f = 1	0	1	0	0	1	0	0	0	0	0	0	0	1	F_2	F_1	F_0	8	2	f = 1
SKN	f	Skip if f = 0	0	1	0	0	1	0	0	0	0	0	0	1	1	F_2	F_1	F_0	8	2	f = 0
SKIT	irf	Skip if irf = 1, then reset irf	0	1	0	0	1	0	0	0	0	1	0	I_4	I_3	I_2	I_1	I_0	8	2	irf = 1
SKNIT	irf	Skip if irf = 0 Reset irf if irf = 1 and don't skip	0	1	0	0	1	0	0	0	0	1	1	I_4	I_3	I_2	I_1	I_0	8	2	irf = 0
CPU Control																					
NOP		No operation	0	0	0	0	0	0	0	0									4	1	
EI		Enable interrupt	1	0	1	0	1	0	1	0									4	1	
DI		Disable interrupt	1	0	1	1	1	0	1	0									4	1	
HLT		Set HALT mode	0	1	0	0	1	0	0	0	0	0	1	1	1	0	1	1	12	2	
STOP		Set STOP mode	0	1	0	0	1	0	0	0	1	0	1	1	1	0	1	1	12	2	

NEC

Figure C-1. 64-Pin Plastic QUIP

Item	Millimeters	Inches
A	41.8 max	1.65 max
C	2.54	.100
D	.5 ±.1	.020 ±.004
E	39.37	1.55
F	1.27 min	.050 min
G	3.6	.142
H	3.2 min	.126 min
I	24.13 ±1.05	.950 ±.041
J	19.05 ±1.05	.750 ±.041
K	16.5	.650
N	.25 +.10 −.05	.010 +.004 −.002



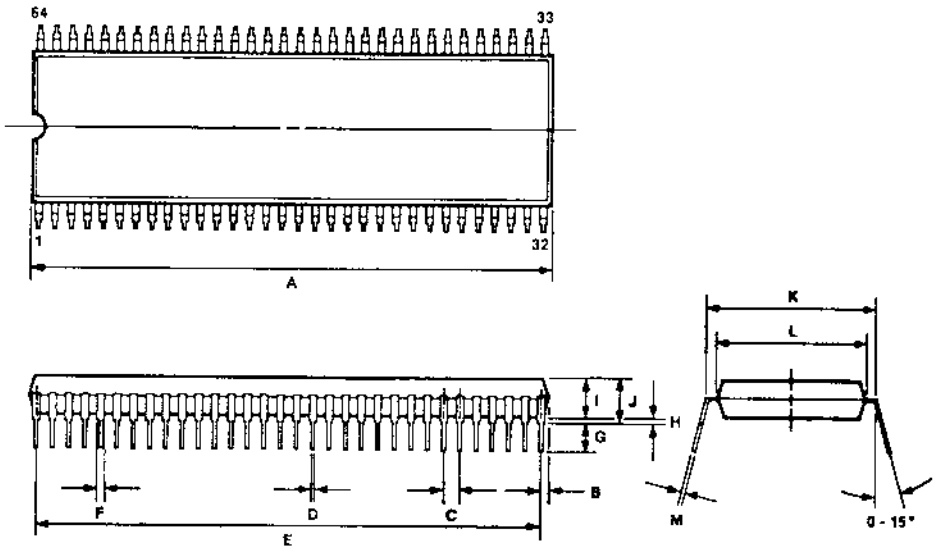
83-001508B

Figure C-2. 64-Pin Plastic Shrink DIP (750 mil)

Item	Millimeters	Inches
A	58.68 max	2.311 max
B	1.78 max	.07 max
C	1.778 [TP]	.07 [TP]
D	.5 ±.10	.02 +.004 −.005
E	55.12	2.17
F	.9 min	.035 min
G	3.2 ±.3	.126 ±.012
H	.51 min	.02 min
I	4.31 max	.17 max
J	5.08 max	.2 max
K	19.05 [TP]	.75 [TP]
L	17	.669
M	.25 +.10 −.05	.01 +.004 −.003

Notes:

1. Each lead centerline is located within .17 mm [.0007 inch] of its true position [TP] at maximum material condition.
2. Item "K" to center of leads when formed parallel.



83-001494B

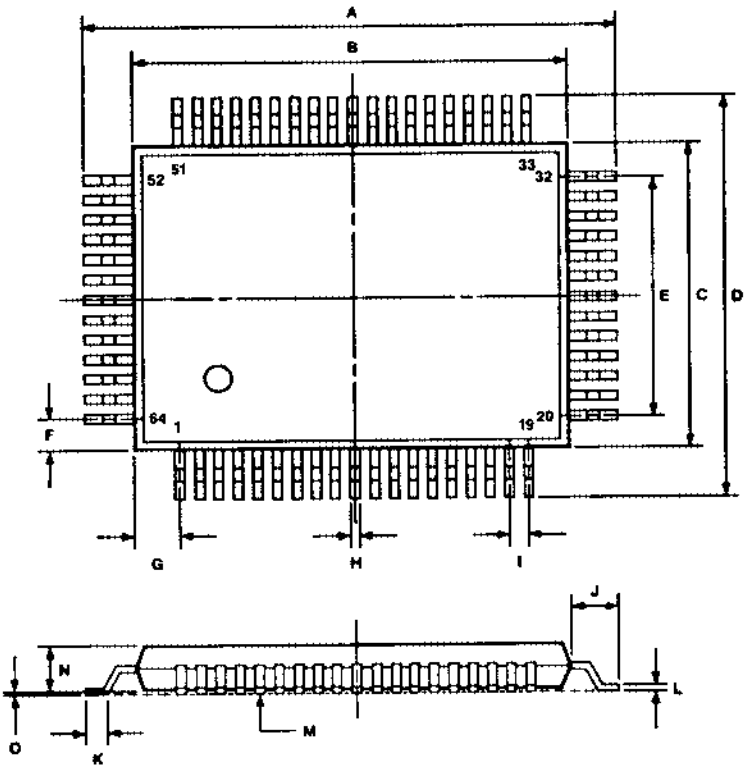
Figure C-3. 64-Pin Plastic Miniflat

Item	Millimeters	Inches
A	24.7 ±.4	.972 $^{+.017}_{-.016}$
B	20 ±.2	.795 $^{+.009}_{-.008}$
C	14 ±.2	.551 $^{+.009}_{-.008}$
D	18.7 ±.4	0.736 ±.016
E	12.0	.472
F	1.0	.039
G	1.0	.039
H	.40 ±.10	.016 $^{+.004}_{-.005}$
I	1.0 [TP] Note 1	.039 [TP]
J	2.35 ±.2	.093 $^{+.008}_{-.009}$
K	1.2 ±.2	.047 $^{+.009}_{-.008}$
L	.15 $^{+.10}_{-.05}$	.006 $^{+.004}_{-.003}$
M	.15 Note 2	.006
N	2.05 $^{+.2}_{-.1}$	.081 $^{+.008}_{-.005}$
O	0.1 ±.1	0.004 ±.004

Note:

[1] Each lead centerline is located within .20 mm [.008 inch] of its true position [TP] at maximum material condition.

[2] Flat within .15 mm [.006 inch] total.



83-009338

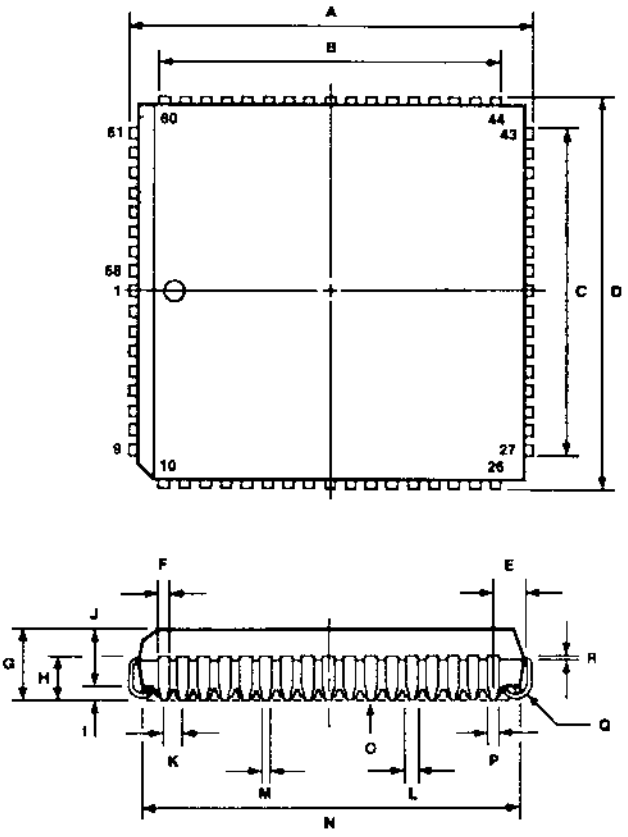
Figure C-4. 68-Pin Plastic Leaded Chip Carrier (PLCC)

Item	Millimeters	Inches
A	25.2 ±.2	.992 ±.008
B	24.20	.953
C	24.20	.953
D	25.2 ±.2	.992 ±.008
E	1.94 ±.15	.076 $^{+.007}_{-.006}$
F	.8	.024
G	4.4 ±.2	.173 $^{+.009}_{-.008}$
H	2.8 ±.2	.110 $^{+.009}_{-.008}$
I	.7 min	.028 min
J	3.6	.142
K	1.27 [TP] Note 1	.050 [TP]
L	.7	.028
M	.40 ±.10	.016 $^{+.004}_{-.005}$
N	23.12 ±.20	.910 $^{+.009}_{-.008}$
O	.15 Note 2	.006
P	1.0	.040
Q	R .8	R .031
R	.20 $^{+.10}_{-.06}$	.008 $^{+.004}_{-.002}$

Note:

[1] Each lead centerline is located within .12 mm [.005 inch] of its true position [TP] at maximum material condition.

[2] Flat within .15 mm [.006 inch] total.



NEC
NEC Electronics Inc.
CORPORATE HEADQUARTERS
401 Ellis Street
P.O. Box 7241
Mountain View, CA 94039
TEL 415-960-6000
TWX 910-379-6985

For Literature Call Toll Free: 1-800-632-3531
1-800-632-3532 (In California)

NECEL-000108
STOCK NO. 500375