

**MCS-48™**  
**MICROCOMPUTER**  
**USER'S MANUAL**

Intellec and MCS are Registered  
Trademarks of Intel Corporation

Copyright ©1978 by Intel Corporation. All rights reserved, no part of this publication including any mnemonics contained herein may be reproduced without the prior written permission of Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051

## INTRODUCTION

1.0 Introduction to MCS-48™ .....	1-1
1.1 Functions of a Computer .....	1-5
1.2 Programming a Microcomputer .....	1-10
1.3 Developing An MCS-48™ Based Product .....	1-13

- **The 8049/8039 is now 80% faster! See page 6-9.**
- **The 8022 is an 8021 with more Memory, I/O, and an A/D converter. See page 6-21.**

# INTRODUCTION

## 1.0 Introduction to MCS-48™

Recent advances in NMOS technology have allowed Intel for the first time to place enough capability on a single silicon die to create a true single-chip microcomputer containing all the functions required in a digital processing system. A set of such microcomputers on single chips, their variations, and optional peripherals are collectively called the MCS-48 microcomputer family. These products are fully described in this manual.

The head of the family is the 8048 microcomputer which contains the following functions in a single 40 pin package:

- 8-Bit CPU
- 1K x 8 ROM Program Memory
- 64 x 8 RAM Data Memory
- 27 I/O Lines
- 8-Bit Timer/Event Counter

A 2.5 or 5.0 microsecond cycle time and a repertoire of over 90 instructions each consisting of either one or two cycles makes the single chip 8048 the equal in performance of most presently available multi-chip NMOS

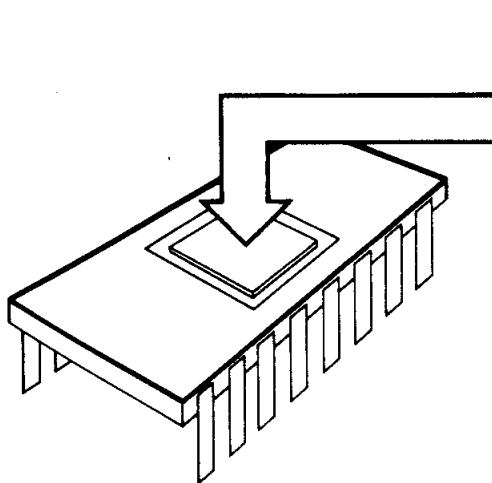
microprocessors. The 8048 is, however, a true "low-cost" microcomputer. A single 5V supply requirement for all MCS-48 components assures that "low cost" also applies to the power supply in your system.

### New Family Members

The MCS-48 family of microcomputers which began with the 8048 and 8748 has now been expanded with new members which provide either more capability or lower cost than the original family members. While broadening the applications possible with a single chip microcomputer, these new microcomputers share both the instruction set and development support of the 8048.

The 8049 is a single-chip microcomputer which is completely interchangeable with the 8048, but contains twice the program memory and twice the data memory of the 8048. The 8035 and 8039 are compatible processors without internal program memory. The 8039 contains twice the data memory of the 8035.

The 8021 is a new very low cost MCS-48 family member which contains a subset of



8049	8048	8021	FEATURES
✓	✓	✓	8 BIT CPU
2K x 8	1K x 8	1K x 8	PROGRAM MEMORY
128 x 8	64 x 8	64 x 8	DATA RAM
27	27	21	I/O LINES
✓	✓	✓	TIMER COUNTER
✓	✓	✓	OSCILLATOR AND CLOCK
✓	✓	✓	RESET CIRCUIT
✓	✓		INTERRUPT

## ON CHIP FEATURES

the 8048's instruction set and incorporates several new features critical in low cost applications.

Even with low component costs; however, a project may be jeopardized by high development and rework costs resulting from an inflexible production design. Intel has solved this problem by creating two pin-compatible versions of the 8048 microcomputer: the 8048 with mask Programmable ROM program memory for low cost production and the 8748 with user programmable and erasable EPROM program memory for prototype development. The 8748 is essentially a single chip microcomputer "breadboard" which can be modified over and over again during development and pre-production then replaced by the low cost 8021\*, 8048, or 8049 ROM for volume production. The 8748 provides a very easy transition from development to production and also provides an easy vehicle for temporary field updates while new ROMs are being made.

---

### **SPECIAL FEATURES**

- **SINGLE 5V SUPPLY**
- **40 PIN DIP OR 28 PIN DIP**
- **PIN COMPATIBLE ROM AND EPROM**
- **2.5, 5.0 AND 10.0  $\mu$ sec CYCLE VERSIONS**
- **ALL INSTRUCTIONS 1 OR 2 CYCLES**
- **SINGLE STEP**
- **8 LEVEL STACK**
- **2 WORKING REGISTER BANKS**
- **RC, LC, XTAL, OR EXTERNAL FREQUENCY SOURCE**
- **OPTIONAL CLOCK OUTPUT**
- **POWER DOWN STANDBY MODE**

---

To allow the MCS-48 to solve a wide range of problems and to provide for future expansion, all 8048 and 8049 functions have been made externally expandable using either special expanders or standard memories and peripherals. An efficient low cost means of I/O expansion is provided by either the 8243 I/O Expander or standard TTL or CMOS circuits. The 8243 provides 16 I/O lines in a 24 pin package. For systems with large I/O requirements, multiple 8243s can be used.

\*The 8021 is code compatible but not pin compatible with the 8748.

For such applications as Keyboards, Displays, Serial communication lines, etc. standard MCS-80/85 peripheral circuits may be added. Program and data memory may be expanded using standard memories or the 8355 and 8155 memories that also include programmable I/O lines and timing functions.

For applications which require a more custom tailored interface, the 8041 or 8741 Universal Peripheral Interface (UPI-41) devices can be used. The UPI-41 devices are available in both ROM and EPROM versions and are essentially slave versions of the 8048/8748 which are designed to interface directly with expandable MCS-48 processors and provide flexible intelligent I/O capability. The 8041/8741 share the instruction set of the MCS-48 family of processors.

The 8035 and 8039 are an 8048 or 8049 respectively without internal program memory that allows the user to match his program memory requirements exactly by using a wide variety of external memories. The 8035 and 8039 allow the user to select a minimum cost system no matter what his program memory requirements. The 8035L is an 8035 with the powerdown mode of the 8048.

The MCS-48 processors are designed to be efficient control processors as well as arithmetic processors. They provide an instruction set which allows the user to directly set and reset individual lines within its I/O ports as well as test individual bits within the accumulator. A large variety of branch and table look-up instructions make these processors very efficient in implementing standard logic functions. Also, special attention has been given to code efficiency. Over 70% of the instructions are a single byte long and all others are only two bytes long. This means many functions requiring 1.5K to 2.0K bytes in other computers may very well be compressed into the 1K words resident in the 8048 or up to 3K to 4K equivalent bytes may be compressed into the 8049.

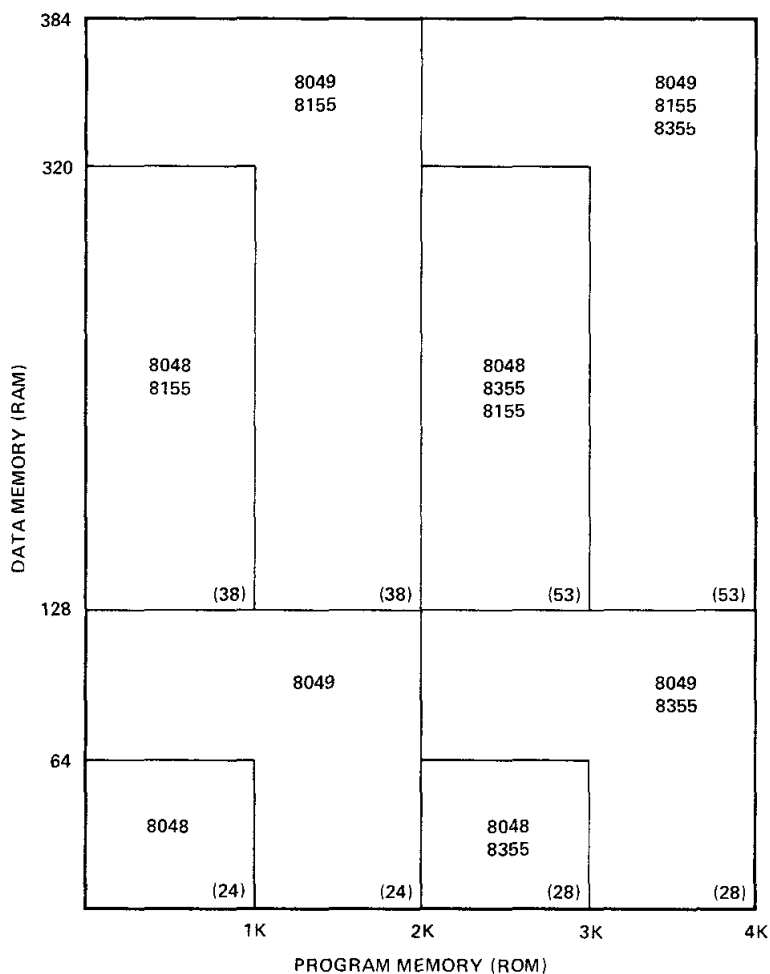
# INTRODUCTION

	FUNCTION	PART NUMBER	DESCRIPTION	COMMENTS
MCS-48™	Microcomputers	8021 8048 8049 8035 8035L 8039 8048-8 8748-8 8035-8	1K ROM Program Memory 1K ROM Program Memory 2K ROM Program Memory No Program Memory 64 x 8 RAM 8035 with Power Down Mode No Program Memory 128 x 8 RAM 1K ROM Program Memory 1K EPROM Program Memory No Program Memory	Compatible versions of the single chip microcomputers provide mask programmed, light erasable, or no internal program memory.
	Memory and I/O Expanders	8355 8755A 8155/56	2K x 8 ROM with 16 I/O Lines 2K x 8 EPROM with 16 I/O Lines 256 x 8 RAM with 22 I/O Lines and Timer	Compatible devices allow direct expansion of MCS-48 functions with no additional external components.
	I/O Expander	8243	16 Line I/O Expander	Low Cost I/O Expander
Compatible MCS-80/85™ Components	Standard ROMs	8308 2316E	1K x 8 450 ns 2K x 8 450 ns	Allow low cost external expansion of Program Memory. The 8308 is interchangeable with 8708 and the 2316E with the 2716.
	Standard EPROM	8708 2716	1K x 8 450 ns Light Erasable 2K x 8 450 ns Light Erasable	User programmable and erasable.
	Standard RAMs	8111A-4 8101A-4 5101	256 x 4 450 ns Common I/O 256 x 4 450 ns Separate I/O 256 x 4 650 ns CMOS	Data memory can be easily expanded using standard NMOS RAMs. The 5101 CMOS equivalent reduces standby power to 75 nW/bit
	Standard I/O	8212 8255A 8251A	8-Bit I/O Port Programmable Peripheral Interface Programmable Communicating Interface	Serves as Address Latch or I/O port. Three 8-bit programmable I/O ports. Serial Communications Receiver/Transmitter
	Standard Peripherals	8205 8214 8216 8226 8253 8259 8279 8278	1 of 8 Binary Decoder Priority Interrupt Controller Bi-directional Bus Driver Bi-directional Bus Driver (Inverting) Programmable Interval Timer Programmable Interrupt Controller Programmable Keyboard/Display Interface (64 Keys) Programmable Keyboard/Display Interface (128 Keys)	MCS-80 peripheral devices are compatible with the MCS-48 allowing easy addition of such specialized interfaces as the 8279 Keyboard/Display Interface. Future MCS-80/85 devices will also be compatible.
	Universal Peripheral Interface	8041 8741	ROM Program Memory EPROM Program Memory	User programmable to perform any custom I/O and control functions.

## MCS-48™ MICROCOMPUTER COMPONENTS

# INTRODUCTION

---



( ) NUMBER OF AVAILABLE I/O LINES

## THE EXPANDED MCS-48™ SYSTEM

---

The chart above shows the expansion possibilities using the 8048 and 8049 in various combinations with the Intel® 8355/8755 Program Memory and I/O Expander and the 8155 Data Memory and I/O Expander. Data Memory can be expanded beyond the resident words in blocks of 256

by adding 8155's. Program Memory can be expanded beyond the resident 1K or 2K in blocks of 2K by using the 8355/8755 in combination with the 8048 or 8049. If all external memory is desired, the 8035 or 8039 can be substituted for the 8048 and 8049.

## 1.1 The Function of a Computer

This chapter introduces certain basic computer concepts. It provides background information and definitions which will be useful in later chapters of this manual. Those already familiar with computers may skip this material, at their option.

### 1.1.1 A Typical Computer System

A typical digital computer consists of:

- A central processor unit (CPU)
- Program Memory
- Data Memory
- Input/output (I/O) ports

The processor memory serves as a place to store Instructions, the coded pieces of information that direct the activities of the CPU, while Memory stores the Data, the coded pieces of information that are processed by the CPU. A group of logically related instructions stored in memory is referred to as a Program. The CPU "reads" each instruction from memory in a logically determined sequence, and uses it to initiate processing actions. If the program sequence is coherent and logical, processing the program will produce intelligible and useful results. The program must be organized such that the CPU does not read a non-instruction word when it expects to see an instruction.

The CPU can rapidly access any data stored in memory; but often the memory is not large enough to store the entire data bank required for a particular application. The problem can be resolved by providing the computer with one or more Input Ports. The CPU can address these ports and input the data contained there. The addition of input ports enables the computer to receive information from external equipment (such as a paper tape reader or floppy disk) at high rates of speed and in large volumes.

A computer also requires one or more Output Ports that permit the CPU to communicate the result of its processing to the outside world. The output may go to a display, for use by a human operator, to a peripheral device that produces "hard-copy", such as a line-

printer, to a peripheral storage device, such as a floppy disk unit, or the output may constitute process control signals that direct the operations of another system, such as an automated assembly line. Like input ports, output ports are addressable. The input and output ports together permit the processor to communicate with the outside world.

The CPU unifies the system. It controls the functions performed by the other components. The CPU must be able to fetch instructions from memory, decode their binary contents and execute them. It must also be able to reference memory and I/O ports as necessary in the execution of instructions. In addition, the CPU should be able to recognize and respond to certain external control signals, such as INTERRUPT requests. The functional units within a CPU that enable it to perform these functions are described below.

### 1.1.2 The Architecture of a CPU

A typical central processor unit (CPU) consists of the following interconnected functional units:

- Registers
- Arithmetic/Logic Unit (ALU)
- Control Circuitry

Registers are temporary storage units within the CPU. Some registers, such as the program counter and instruction register, have dedicated uses. Other registers, such as the accumulator, are for more general purpose use.

#### Accumulator

The accumulator usually stores one of the operands to be manipulated by the ALU. A typical instruction might direct the ALU to add the contents of some other register to the contents of the accumulator and store the result in the accumulator itself. In general, the accumulator is both a source (operand) and a destination (result) register. Often a CPU will include a number of additional general purpose registers that can be used to store operands or intermediate data. The availability of general purpose registers

eliminates the need to "shuffle" intermediate results back and forth between memory and the accumulator, thus improving processing speed and efficiency.

### **Program Counter (Jumps, Subroutines and the Stack):**

The instructions that make up a program are stored in the system's memory. The central processor references the contents of memory in order to determine what action is appropriate. This means that the processor must know which location contains the next instruction.

Each of the locations in memory is numbered, to distinguish it from all other locations in memory. The number which identifies a memory location is called its Address. The processor maintains a counter which contains the address of the next program instruction. This register is called the Program Counter. The processor updates the program counter by adding "1" to the counter each time it fetches an instruction, so that the program counter is always current (pointing to the next instruction).

The programmer therefore stores his instructions in numerically adjacent addresses, so that the lower addresses contain the first instructions to be executed and the higher addresses contain later instructions. The only time the programmer may violate this sequential rule is when an instruction in one section of memory is a Jump instruction to another section of memory.

A jump instruction contains the address of the instruction which is to follow it. The next instruction may be stored in any memory location, as long as the programmed jump specifies the correct address. During the execution of a jump instruction, the processor replaces the contents of its program counter with the address embodied in the Jump. Thus, the logical continuity of the program is maintained.

A special kind of program jump occurs when the stored program "Calls" a subroutine. In

this kind of jump, the processor is required to "remember" the contents of the program counter at the time that the jump occurs. This enables the processor to resume execution of the main program when it is finished with the last instruction of the subroutine.

A Subroutine is a program within a program. Usually it is a general-purpose set of instructions that must be executed repeatedly in the course of a main program. Routines which calculate the square, the sine, or the logarithm of a program variable are good examples of functions often written as subroutines. Other examples might be programs designed for inputting data to a particular peripheral device.

The processor has a special way of handling subroutines, in order to insure an orderly return to the main program. When the processor receives a Call instruction, it increments the Program Counter and stores the counter's contents in a reserved memory area known as the Stack. The Stack thus saves the address of the instruction to be executed after the subroutine is completed. Then the processor loads the address specified in the Call into its Program Counter. The next instruction fetched will therefore be the first step of the subroutine.

The last instruction in any subroutine is a Return. Such an instruction need specify no address. When the processor fetches a Return instruction, it simply replaces the current contents of the Program Counter with the address on the top of the stack. This causes the processor to resume execution of the calling program at the point immediately following the original Call instruction.

Subroutines are often Nested; that is, one subroutine will sometimes call a second subroutine. The second may call a third, and so on. This is perfectly acceptable, as long as the processor has enough capacity to store the necessary return addresses, and the logical provision for doing so. In other words, the maximum depth of nesting is determined by the depth of the stack itself. If the stack has space for storing three return addresses, then



three levels of subroutines may be accommodated.

### **Instruction Register and Decoder**

Every computer has a Word Length that is characteristic of that machine. A computer's word length is usually determined by the size of its internal storage elements and interconnecting paths (referred to as Buses); for example, a computer whose registers and buses can store and transfer 8-bits of information has a characteristic word length of 8-bits and is referred to as an 8-bit parallel processor. An 8-bit parallel processor generally finds it most efficient to deal with 8-bit binary fields, and the memory associated with such a processor is therefore organized to store 8-bits in each addressable memory location. Data and instructions are stored in memory as 8-bit binary numbers, or as numbers that are integral multiples of 8-bits: 16-bits, 24-bits, and so on. This characteristic 8-bit field is often referred to as a Byte. If however, efficient handling of 4 or even 1-bit data is necessary special processor instructions can provide this capability.

Each operation that the processor can perform is identified by a unique byte of data known as an Instruction Code or Operation Code. An 8-bit word used as an instruction code can distinguish between 256 alternative actions, more than adequate for most processors.

The processor fetches an instruction in two distinct operations. First, the processor transmits the address in its Program Counter to the program memory. Then the program memory returns the addressed byte to the processor. The CPU stores this instruction byte in a register known as the Instruction Register, and uses it to direct activities during the remainder of the instruction execution.

The 8-bits stored in the instruction register can be decoded and used to selectively activate one of a number of output lines. Each line represents a set of activities associated with execution of a particular instruction code. The enabled line can be combined with selected timing pulses, to develop electrical

signals that can then be used to initiate specific actions. This translation of code into action is performed by the Instruction Decoder and by the associated control circuitry.

An 8-bit instruction code is often sufficient to specify a particular processing action. There are times, however, when execution of the instruction requires more information than 8-bits can convey.

One example of this is when the instruction references a memory location. The basic instruction code identifies the operation to be performed, but cannot specify the object address as well. In a case like this, a two byte instruction must be used. Successive instruction bytes are stored in sequentially adjacent memory locations, and the processor performs two fetches in succession to obtain the full instruction. The first byte retrieved from memory is placed in the processor's instruction register, and subsequent byte is placed in temporary storage; the processor then proceeds with the execution phase.

### **Address Register(s)**

A CPU may use a register to hold the address of a memory location that is to be accessed for data. If the address register is Programmable, (i.e., if there are instructions that allow the programmer to alter the contents of the register) the program can "build" an address in the address register prior to executing a Memory Reference instruction (i.e., an instruction that reads data from memory, writes data to memory or operates on data stored in memory).

### **Arithmetic/Logic Unit (ALU)**

All processors contain an arithmetic/logic unit, which is often referred to simply as the ALU. The ALU, as its name implies, is that portion of the CPU hardware which performs the arithmetic and logical operations on the binary data.

The ALU must contain an Adder which is capable of combining the contents of two registers in accordance with the logic of binary arithmetic. This provision permits the

processor to perform arithmetic manipulations on the data it obtains from memory and from its other inputs.

Using only the basic adder a capable programmer can write routines which will subtract, multiply and divide, giving the machine complete arithmetic capabilities. In practice, however, most ALUs provide other built-in functions, including boolean logic operations, and shift capabilities.

The ALU contains Flag Bits which specify certain conditions that arise in the course of arithmetic and logical manipulations. It is possible to program jumps which are conditionally dependent on the status of one or more flags. Thus, for example, the program may be designed to jump to a special routine if the carry bit is set following an additional instruction.

## **Control Circuitry**

The control circuitry is the primary functional unit within a CPU. Using clock inputs, the control circuitry maintains the proper sequence of events required for any processing task. After an instruction is fetched and decoded, the control circuitry issues the appropriate signals (to units both internal and external to the CPU) for initiating the proper processing action. Often the control circuitry will be capable of responding to external signals, such as an interrupt. An Interrupt request will cause the control circuitry to temporarily interrupt main program execution, jump to a special routine to service the interrupting device, then automatically return to the main program.

### **1.1.3 Computer Operations**

There are certain operations that are basic to almost any computer. A sound understanding of these basic operations is a necessary prerequisite to examining the specific operations of a particular computer.

#### **Timing**

The activities of the central processor are cyclical. The processor fetches an instruction, performs the operations required,

fetches the next instruction, and so on. This orderly sequence of events requires precise timing, and the CPU therefore requires a free running oscillator clock which furnishes the reference for all processor actions. The combined fetch and execution of a single instruction is referred to as an Instruction Cycle. The portion of a cycle identified with a clearly defined activity is called a State. And the interval between pulses of the timing oscillator is referred to as a Clock Period. As a general rule, one or more clock periods are necessary for the completion of a state, and there are several states in a cycle.

#### **Instruction Fetch**

The first state(s) of any instruction cycle will be dedicated to fetching the next instruction. The CPU issues a read signal and the contents of the program counter are sent to program memory, which responds by returning the next instruction word. The first byte of the instruction is placed in the instruction register. If the instruction consists of more than one byte, additional states are required to fetch the second byte of the instruction. When the entire instruction is present in the CPU, the program counter is incremented (in preparation for the next instruction fetch) and the instruction is decoded. The operation specified in the instruction will be executed in the remaining states of the instruction cycle. The instruction may call for a data memory read or write, an input or output and/or an internal CPU operation, such as a register-to-register transfer or an add operation.

#### **Memory Read**

An instruction fetch is merely a special program memory read operation that brings the instruction to the CPU's instruction register. The instruction fetched may then call for data to be read from data memory into the CPU. The CPU again issues a read signal and sends the proper memory address; memory responds by returning the requested word. The data received is placed in the accumulator or one of the other general purpose registers (not the instruction register).

### **Memory Write**

A memory write operation is similar to a read except for the direction of data flow. The CPU issues a write signal, sends the proper memory address, then sends the data word to be written into the addressed data memory location.

### **Input/Output**

Input and Output operations are similar to memory read and write operations with the exception that an I/O port is addressed instead of a memory location. The CPU issues the appropriate input or output control signal, sends the proper address and either receives the data being input or sends the data to be output.

Data can be input/output in either parallel or serial form. All data within a digital computer is represented in binary coded form. A binary data word consists of a group of bits; each bit is either a one or a zero. Parallel I/O consists of transferring all bits in the word at the same time, one bit per line. Serial I/O consists of transferring one bit at a time on a single line. Naturally serial I/O is much slower, but it requires considerable less hardware than does parallel I/O.

### **Interrupts**

Interrupt provisions are included on many central processors, as a means of improving

the processor's efficiency. Consider the case of a computer that is processing a large volume of data, portions of which are to be output to a printer. The CPU can output a byte of data within a single machine cycle but it may take the printer the equivalent of many machine cycles to actually print the character specified by the data byte. The CPU could then remain idle waiting until the printer can accept the next data byte. If an interrupt capability is implemented on the computer, the CPU can output a data byte then return to data processing. When the printer is ready to accept the next data byte, it can request an interrupt. When the CPU acknowledges the interrupt, it suspends main program execution and automatically branches to a routine that will output the next data byte. After the byte is output, the CPU continues with main program execution. Note that this is, in principle, quite similar to a subroutine call, except that the jump is initiated externally rather than by the program.

More complex interrupt structures are possible, in which several interrupting devices share the same processor but have different priority levels. Interruptive processing is an important feature that enables maximum utilization of a processor's capacity for high system throughput.

## **1.2 Programming a Microcomputer**

### **1.2.1 Machine Language Programming**

A microprocessor is instructed what to do by programming it with a series of instructions stored in Program Memory. The processor fetches these instructions one at a time and performs the operation indicated. These instructions must be stored in a form that the processor can understand. This format is referred to as Machine Language. For most microprocessors this instruction is a group of 8 binary bits (1's and 0's) called a word (also called a byte if the word is 8-bits). Some instructions require more than one location in Program Memory. To execute a multi-byte instruction, the processor must execute multiple fetches of program memory before performing the instruction. Because multi-byte instructions take more Program Memory and take longer to execute than single byte instructions their use is usually kept to a minimum.

A processor may be programmed by writing a sequence of instructions in the binary code (ones and zeros) which the machine can interpret directly. This is machine language programming and it is very useful where the program to be written is small and the application requires that the designer have an intimate knowledge of the microprocessor. Machine language programming allows the user, because of his detailed knowledge, to use many programming "tricks" to produce the most compact and efficient code possible.

The following is an example of a machine language program: This program reads 5 sequential 8-bit words in from an I/O port and stores them sequentially in data memory. The program starts by initializing two registers, one which determines where the data is to be stored and another which

counts the number of words to be stored. When finished the processor continues on to the next instructions.

<b>Step Number</b>	<b>Machine Code</b>	<b>Explanation</b>
0	1011 1000	Load decimal 32 in
1	0010 0000	register R0
2	1011 1010	Load decimal 5 in
3	0000 0101	register R2
4	0000 1001	Load Port 1 to accumulator
5	1111 0000	Transfer contents of accumulator to register addressed by register 0
6	0001 1000	Increment R0 by 1
7	1110 1010	Decrement register 2
8	0000 0100	by 1, if result is zero continue to step 9, if not go to step 4
9	—	
10	—	

As you can see, writing machine instructions in ones and zeros can be very laborious and subject to error. It is almost always more efficient to represent each 8-bits if machine language code in a shorthand format called Hexadecimal. The term hexadecimal results from the character set used in hexadecimal notation. Hexadecimal is merely an extension of the normal decimal numbers by the addition of the first six letters of the alphabet. This gives a total of 16 different characters. Each hexadecimal "digit" can represent 16 values or the equivalent of four binary bits; therefore, each 8-bit machine language word can be represented by 2 hexadecimal (hex for short) digits. The correspondence among the decimal, binary, and hex number systems is given below:

# INTRODUCTION

Decimal	Hex	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Our machine language program then becomes:

Step	Hex Code
0	B8
1	20
2	BA
3	05
4	09
5	F0
6	18
7	EA
8	04

This coding is now quite efficient to write and read and coding errors are much easier to detect. Hex coding is usually very efficient for small programs (a few hundred lines of code) however, it does have two major limitations in larger programs:

1. Hex coding is not self-documenting, that is, the code itself does not give any indication in human terms of the operation to be performed. The user must learn each code or constantly use a Program Reference Card to convert.
2. Hex coding is absolute, that is, the program will work only when stored in a specific location in program memory. This is because the branch or jump instructions in the program reference specific addresses elsewhere in the program. In the example above steps 7 and 8 reference step (or address) 4. If the program were to be moved,

step 8 would have to be changed to refer to the new address of step 4.

## 1.2.2 Assembly Language Programming

Assembly language overcomes the disadvantages of machine language by allowing the use of alphanumeric symbols to represent machine operation codes, branch addresses, and other operands. For example, the instruction to increment the contents of register 0 becomes `INC R0` instead of the hex 18, giving the user at a glance the meaning of the instruction. Our example program can be written in assembly language as follows:

Step No.	Hex Code	Assembly Code
0	B8	MOV R0, #32
1	20	
2	BA	MOV R2, #05
3	05	
4	09	INP: IN A, P1
5	F0	MOV @R0, A
6	18	INC R0
7	EA	DJNZ R2, INP
8	04	

The first statement can be verbalized as follows: Move to Register 0 the decimal number 32. Move instructions are always structured such that the destination is first and the source is second. The pound sign “#” indicates that the source is “immediate” data (data contained in the following byte of program memory). In this case data was specified as a decimal 32, however, this could have been written as a hex 20H or a binary 0010 0000B since the assembler will accept either form. Notice also that in this instance two lines of hex code are represented by one line of assembly code.

The input instruction `IN A, P1` has the same form as a `MOV` instruction indicating that the contents of Port 1 are to be transferred to the accumulator. In front of the input instruction is an address lable which is delineated by a colon. This lable allows the program to be written in a form independent of its final location in program memory since the branch instruction at the end of the program can refer to this lable rather than a specific address. This is a very important advantage of assembly language programs since it

allows instructions to be added or deleted throughout the program during debugging without requiring that any jump addresses be changed.

The next instruction `MOV @R0, A` can be verbalized as, Move to the data memory location addressed by R0, the contents of the accumulator. The `@` sign indicates an indirect operation whereby the contents of either register 0 or register 1 acts as a pointer to the data memory location to be operated on.

The last instruction is a Decrement and Jump if Not Zero instruction which acts in combination with the specified register as a loop counter. In this case register 2 is loaded with 5 initially and then decremented by one each time the loop is executed. If the result of the decrement is not zero, the program jumps to INP and executes another input operation. The fifth time thru the loop the result is zero and execution falls through to whatever routine follows the `DJNZ` instruction.

In addition to the normal features provided by assemblers, more advanced assemblers such as that for the MCS-48 offer such things as evaluation of expressions at assembly time, conditional assembly, and macro capability.

1. Evaluation of Expressions - Certain assemblers allow the use of arithmetic expressions and multiple symbols in the operand portion of instructions. For instance the MCS-48 assembler accepts instructions such as:

```
ADD A, # ALFA*BETA/2
```

ALFA and BETA are two previously defined symbols. At assembly time the expression `ALFA*BETA/2` will be evaluated and the resulting number (which is the average of ALFA and BETA) will be treated as immediate data and designated as the second byte of the `ADD` immediate instruction. This expression has allowed the immediate data of this instruction to be defined in a single statement and eliminated the need for a third symbol equal to `ALFA*BETA/2`.

2. Conditional Assembly - Conditional assembly allows the programmer to select only certain portions of his assembly language (source) program for conversion to machine (object) code at assembly time. This allows for instance, the inclusion of various "debug" routines to be included in the program during development. Using conditional assembly, they can then be left out when the final assembly is done.

Conditional assembly also allows several versions of one basic program to be generated by selecting various portions of a larger program at assembly time.

3. Macro's - A macro instruction is essentially a symbol which is recognized by the assembler to represent a specific sequence of several standard instructions. A macro is a shorthand way of generating the same sequence of instructions at several locations in a program without having to rewrite the sequence each time it is used. For example, a typical macro instruction might be one which performs a subtract operation. The 8048 does not have a subtract instruction as such but the operation can be performed easily with three instructions:

```
CPL A
ADD A, REG
CPL A
```

This routine subtracts a register from the accumulator and leaves the result in the accumulator. This sequence can be defined as a macro with the name `SUB` and an operand which can be R0 to R7. To subtract R7 from the accumulator then, the programmer merely has to write:

```
SUB R7
```

and the assembler will automatically insert the three instructions above with R7 substituted for REG.

Once the assembly language source code is written it can be converted to machine executable object code by passing it through an assembler program. The MCS-48 assembler is a program which runs on the 8080-based Intellec MDS system explained in the next section.

## 1.3 Developing An MCS-48™ Based Product

Although the development of a microcomputer based product may differ in detail from the development cycle of a product based on TTL logic or relays, the basic procedures are the same — only the tools are different.

### 1.3.1 Education

The first step of course is to become familiar with what the microcomputer is and what it can do. The first step in this education is this document, the MCS-48™ User's Manual. The user's manual gives a detailed description of the MCS-48 family of components and how they may be used in various system configurations. Also included is a description of the 8048 instruction set and examples of how the instructions may be used. For a more complete discussion of the instruction set and programming techniques the MCS-48 Assembly Language Manual is also available.

If time is critical in getting started in microcomputers, individuals can attend one of many Intel sponsored 3-day training courses which give basic instruction in the MCS-48 as well as hands-on experience with MCS-48 development systems. These courses are a convenient means of getting started with the MCS-48, particularly for those not familiar with microprocessors.

After general familiarization is complete, either through self-instruction or a training course, the next step is to gain a better "feel" for what a microprocessor can do in your own applications by writing several exercise programs which perform basic functions. You may require such things as I/O routines, delays, counting functions, look-up tables, arithmetic functions, and logical operations which can serve as a set of building blocks for future applications programs. Several basic programming examples are included in the MCS-48 Assembly Language Manual while the Intel User's Library is a source of more specific applications routines.

### 1.3.2 Function Definition

After a thorough understanding of the

microprocessor is achieved, the functions to be implemented can be defined using a flowchart method to describe each basic system function and the sequence in which the processor executes these functions. Once the system is flowcharted, critical time-related functions can be identified and sample programs written to verify that performance requirements can be met.

### 1.3.3 Hardware Configuration

The next step involves the definition of the microcomputer hardware required to implement the function. Input/Output capability must be defined in terms of number of inputs, number of outputs, bi-directional lines, latching or non-latching I/O, output drive capability, and input impedance. The number of words of RAM storage required for intermediate results and data storage must then be determined. The type of system will dictate whether battery backup is needed to maintain data RAM during power failure.

Probably the most difficult parameter to define initially is the amount of program memory needed to store the applications program. Although previously written exercise programs will make this estimate more accurate, a generous amount of "breathing room" should be allowed in program memory until coding is complete and the exact requirements are known. Many special functions such as serial communications (TTY) or keyboard/display interfaces may be implemented in software (programs); however, in cases where these functions place a severe load on the processor in terms of time or program memory, special peripheral interface circuits such as the 8251, Universal Synchronous or Asynchronous Receiver/Transmitter (USART) or 8279 Keyboard/Display interface may be used.

### 1.3.4 Code Generation

The writing of the final program code for the application can begin once the system function and hardware have been defined and can be generated in parallel with the detailed hardware design (PC card layout, power supply, etc.)

# INTRODUCTION

At this point, there are two paths available to the designer/programmer and two types of design development aids provided by Intel to simplify the procedures. One system, called PROMPT 48, is a low cost development system which supports machine language programming and the second is the Intel Microcomputer Development System which supports both machine and assembly languages. For those of you unfamiliar with the advantages and disadvantages of machine and assembly languages see Section 1.2.

## 1.3.5 PROMPT 48

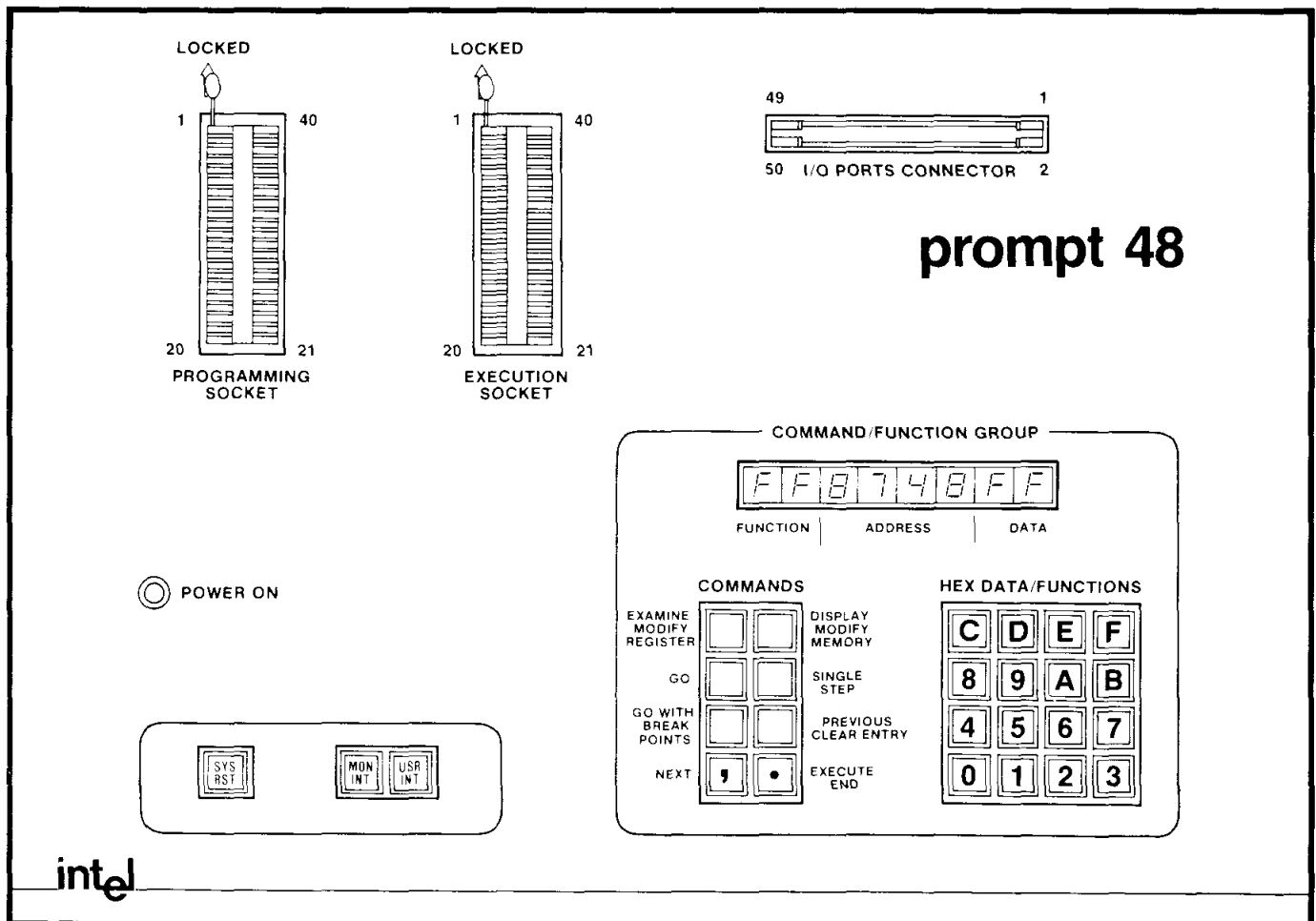
PROMPT 48 is a low cost design aid consisting of: an 8748 processor to execute programs, control circuitry to provide debug functions such as single step and break points, a monitor program stored in ROM, an EPROM programmer, and a hexadecimal keyboard and display. There are two processor sockets on the front of PROMPT 48, one for programming the 8748 and one in

which a programmed 8748 executes its program while under control of the monitor routine.

Use of PROMPT 48 involves the following steps:

1. Loading an application program into the PROMPT RAM memory via Hex keyboard or external terminal (TTY and RS232 interface provided).
2. Inserting an erased 8748 in the programming socket and transferring the application program to its internal EPROM.
3. Transferring programmed 8748 to execution socket where program is executed and debugged under control of the monitor.

The monitor routine allows the user to single step this processor, examine or modify all internal registers and data memory; or to run at full speed and stop the processor at predetermined breakpoints. PROMPT 48





also provides 1K of writeable program memory which may be used to debug user programs. A multiple single step feature is also provided in which the processor steps through its program dumping all internal contents to external RAM where it may be later displayed or typed out on an external terminal. Paper tape input and output in Intel's hexadecimal format is also available through the TTY.

### **1.3.6 Intellec Development System**

The Intellec Microcomputer Development System is a modular development system which can be expanded as necessary to meet the requirements of your design cycle. The system consists of the processor unit which is based on Intel's 8080A microprocessor, and several optional units such as the UPP Universal PROM Programmer, the PTR High Speed Paper tape reader, the DOS Disk Operating System, and the Intellec CRT terminal.

To support the development of MCS-48 systems a macro-assembler ASM 48 is available for the Intellec System as well as a personality module for the UPP which will program the EPROM of the 8748. Also to be provided is in-circuit emulation capability with ICE-48 which will allow emulation and debug of user's 8048 application programs on the 8080A-based Intellec Development System.

The Intellec system is a flexible high performance development system which can support Intel's various microcomputer families with various optional modules. The

macro-assembler and text editor programs provided allow the designer to write and edit his programs in assembly language and then generate the machine language output necessary to program the 8748 EPROM. The availability of a high speed CRT and a diskette operating system eliminates the laborious input and output of paper tape files normally required during the assembly process. Finally, ICE 48 allows the user to extend the resources of his entire Intellec system into the 8048 socket of his own system and use all its emulation, debug, and display facilities directly.

### **1.3.7 Production**

Once a working program has been achieved, a preproduction phase usually follows where several prototype systems are evaluated in simulated situations or in actual operation in the field. During this period the use of the 8748 EPROM allows quick alteration of the application program when problems or suggested changes arise. Depending on the magnitude and number of future changes anticipated, the first production units may also be shipped with EPROM processor. However, to achieve the maximum cost reduction potential in high volume applications, a conversion to the 8048 ROM is usually necessary. This is an easy transition since the 8048 and 8748 are pin and machine code compatible equivalents. The user merely develops a hexadecimal tape of his 8748 program memory contents using his Intellec System or PROMPT 48 development aid and sends it to Intel along with his 8048 order. As the 8048 ROM's arrive they can immediately replace the 8748 EPROMs.

# THE SINGLE COMPONENT MCS-48™ SYSTEM

## SECTION 1: 8048/8748/8035 and 8049/8039

2.0 Summary .....	2-1
2.1 Architecture .....	2-1
2.2 Pin Description .....	2-14
2.3 Programming, Verifying and Erasing EPROM .....	2-16
2.4 Test and Debug .....	2-18

## SECTION 2: 8021

2.5 Program Memory .....	2-21
2.6 Data Memory .....	2-21
2.7 Oscillator and Clock .....	2-22
2.8 Timer/Event Counter .....	2-22
2.9 Input/Output Capabilities .....	2-23
2.10 CPU .....	2-25
2.11 Reset .....	2-25

# THE SINGLE COMPONENT MCS-48™ SYSTEM

## 2.0 Summary

Sections 2.1 through 2.4 describe in detail the functional characteristics of the 8748 EPROM, 8048/8049 ROM and 8035/8039 single component microcomputers. Unless otherwise noted, details within these sections apply to all versions. Sections 2.5 through 2.11 describe the operation of the 8021. This chapter is limited to those functions useful in single-chip implementations of the MCS-48. Chapter 3 discusses functions which allow expansion of program memory, data memory, and input-output capability.

## 2.1 Architecture

The following sections break the 8048 into functional blocks and describe each in detail.

### 2.1.1 Arithmetic Section

The arithmetic section of the processor contains the basic data manipulation functions of the 8048 and can be divided into the following blocks:

- Arithmetic Logic Unit (ALU)
- Accumulator
- Carry Flag
- Instruction Decoder

In a typical operation data stored in the accumulator is combined in the ALU with data from another source on the internal bus (such as a register or I/O port) and the result is stored in the accumulator or another register. The following is a more detailed description of the function of each block:

#### Instruction Decoder

The operation code (op code) portion of each program instruction is stored in the Instruction Decoder and converted to outputs which control the function of each of the blocks of the Arithmetic Section. These lines control the source of data and the destination register as well as the function performed in the ALU.

#### Arithmetic Logic Unit

The ALU accepts 8-bit data words from one or two sources and generates an 8-bit result under control of the Instruction Decoder. The ALU can perform the following functions:

- Add With or Without Carry
- And, OR, Exclusive OR
- Increment/Decrement
- Bit Complement
- Rotate Left, Right
- Swap Nibbles
- BCD Decimal Adjust

If the operation performed by the ALU results in a value represented by more than 8 bits (overflow of most significant bit) a Carry Flag is set in the Program Status Word.

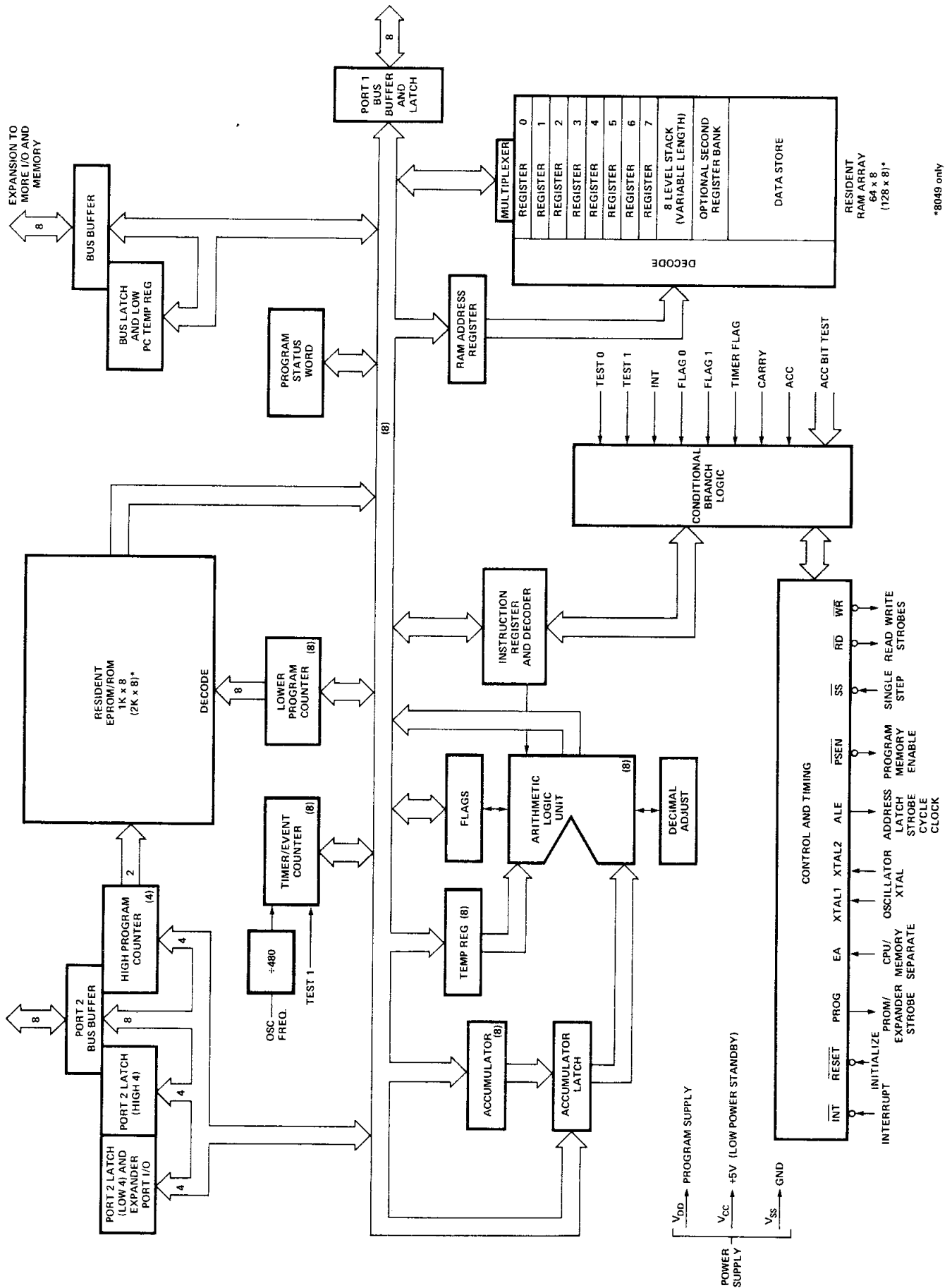
#### Accumulator

The accumulator is the single most important data register in the processor being one of the sources of input to the ALU and often the destination of the result of operations performed in the ALU. Data to and from I/O ports and memory also normally passes through the accumulator.

### 2.1.2 Program Memory

Resident program memory consists of 1024 or 2048 words eight bits wide which are addressed by the program counter. In the 8748 this memory is user programmable and erasable EPROM, in the 8048/8049 the memory is ROM which is mask programmable at the factory, while the 8035/8039 has no internal program memory and is used with external devices. Program code is completely interchangeable among the various versions. See Section 2.3 for EPROM programming techniques.

# SINGLE COMPONENT SYSTEM



**8048/8049 BLOCK DIAGRAM**

# SINGLE COMPONENT SYSTEM

There are three locations in Program Memory of special importance:

## LOCATION 0

Activating the Reset line of the processor causes the first instruction to be fetched from location 0.

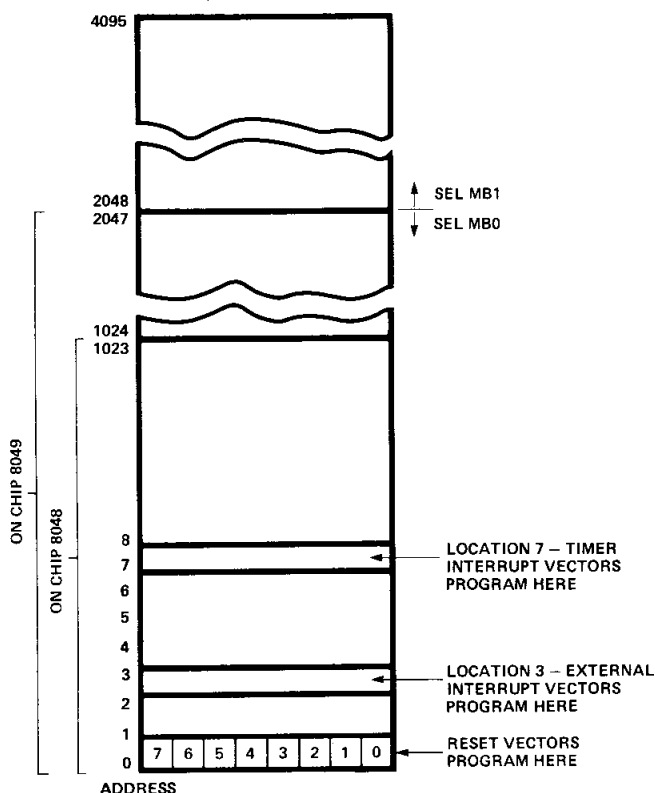
## LOCATION 3

Activating the Interrupt input line of the processor (if interrupt is enabled) causes a jump to subroutine.

## LOCATION 7

A timer/counter interrupt resulting from timer/counter overflow (if enabled) causes a jump to subroutine.

Therefore, the first instruction to be executed after initialization is stored in location 0, the first word of an external interrupt service subroutine is stored in location 3, and the first word of a timer/counter service routine is stored in location 7. Program memory can be used to store constants as well as program instructions. Instructions such as MOVP and MOVP3 allow easy access to data "lookup" tables.

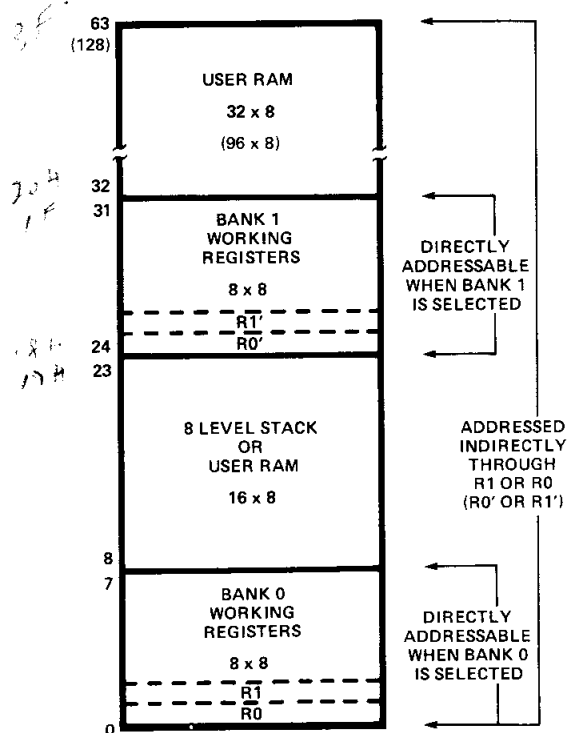


**MCS-48™ PROGRAM MEMORY MAP**

## Data Memory

Resident data memory is organized as 64 or 128 words 8-bits wide. All locations are indirectly addressable through either of two RAM Pointer Registers which reside at address 0 and 1 of the register array. In addition, the first 8 locations (0-7) of the array are designated as working registers and are directly addressable by several instructions. Since these registers are more easily addressed, they are usually used to store frequently accessed intermediate results. The DJNZ instruction makes very efficient use of the working registers as program loop counters by allowing the programmer to decrement and test the register in a single instruction.

By executing a Register Bank Switch instruction (SEL RB) RAM locations 24-31 are designated as the working registers in place of locations 0-7 and are then directly addressable. This second bank of working registers may be used as an extension of the first bank or reserved for use during interrupt service



IN ADDITION R0 OR R1 (R0' OR R1') MAY BE USED TO ADDRESS 256 WORDS OF EXTERNAL RAM.

( ) 8049 only

**DATA MEMORY MAP**

subroutines allowing the registers of Bank 0 used in the main program to be instantly "saved" by a Bank Switch. Note that if this second bank is not used, locations 24-31 are still addressable as general purpose RAM. Since the two RAM pointer Registers R0 and R1 are a part of the working register array, bank switching effectively creates two more pointer registers (R0' and R1') which can be used with R0 and R1 to easily access up to four separate working areas in Ram at one time. RAM locations (8-23) also serve a dual role in that they contain the program counter stack as explained in Sec. 2.1.6. These locations are addressed by the Stack Pointer during subroutine calls as well as by RAM Pointer Registers R0 and R1. If the level of subroutine nesting is less than 8, all stack registers are not required and can be used as general purpose RAM locations. Each level of subroutine nesting not used provides the user with two additional RAM locations.

**2.1.4 Input/Output**

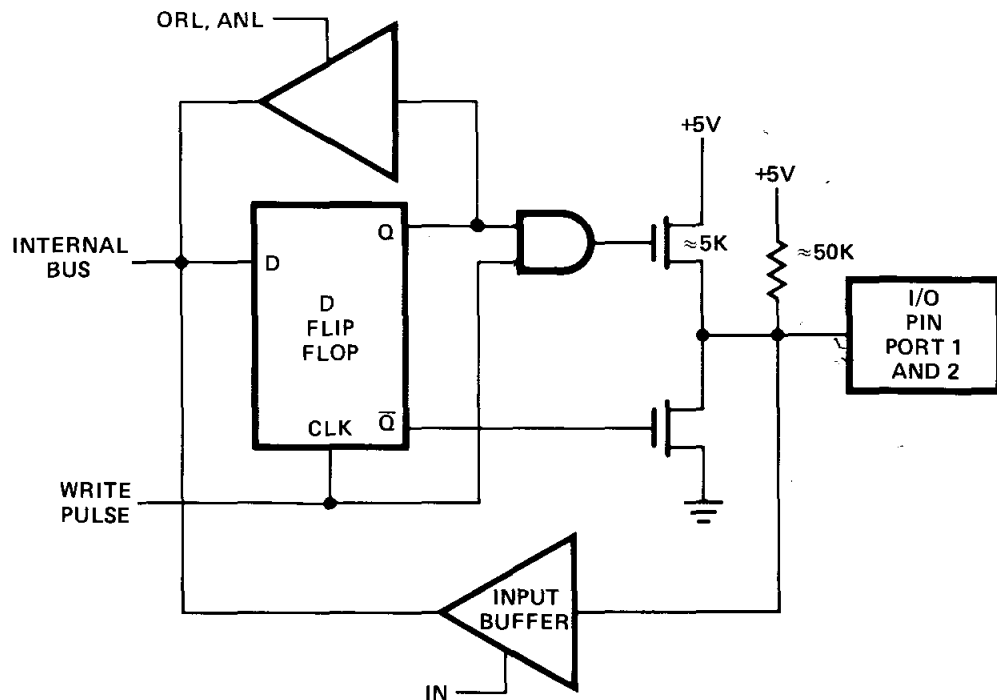
The 8048 has 27 lines which can be used for input or output functions. These lines are grouped as 3 ports of 8 lines each which serve as either inputs, outputs or bidirectional

ports and 3 "test" inputs which can alter program sequences when tested by conditional jump instructions.

**Ports 1 and 2**

Ports 1 and 2 are each 8 bits wide and have identical characteristics. Data written to these ports is statically latched and remains unchanged until rewritten. As input ports these lines are non latching, i.e., inputs must be present until read by an input instruction. Inputs are fully TTL compatible and outputs will drive one standard TTL load.

The lines of ports 1 and 2 are called quasi-bidirectional because of a special output circuit structure which allows each line to serve as an input, an output, or both even though outputs are statically latched. The figure shows the circuit configuration in detail. Each line is continuously pulled up to +5v through a resistive device of relatively high impedance (~50KΩ). This pullup is sufficient to provide the source current for a TTL high level yet can be pulled low by a standard TTL gate thus allowing the same pin to be used for both input and output. To provide fast switching times in a "0" to "1" transition a relatively low



**"QUASI BI DIRECTIONAL" PORT STRUCTURE**

impedance device ( $\sim 5K\Omega$ ) is switched in momentarily ( $\sim 500ns$ ) whenever a "1" is written to the line. When a "0" is written to the line a low impedance ( $\sim 300\Omega$ ) device overcomes the light pullup and provides TTL current sinking capability. Since the pulldown transistor is a low impedance device a "1" must first be written to any line which is to be used as an input. Reset initializes all lines to the high impedance "1" state. This structure allows input and output on the same pin and also allows a mix of input lines and output lines on the same port. The quasi-bidirectional port in combination with the ANL and ORL logical instructions provide an efficient means for handling single line inputs and outputs within an 8-bit processor. See also Section 3.7.

### Bus

Bus is also an 8-bit port which is a true bidirectional port with associated input and output strobes. If the bidirectional feature is not needed, Bus can serve as either a statically latched output port or non-latching input port. Input and output lines on this port cannot be mixed however.

As a static port, data is written and latched using the OUTL instruction and inputted using the INS instruction. The INS and OUTL instructions generate pulses on the corresponding  $\overline{RD}$  and  $\overline{WR}$  output strobe lines; however, in the static port mode they are generally not used. As a bidirectional port the MOVX instructions are used to read and write the port. A write to the port generates a pulse on the  $\overline{WR}$  output line and output data is valid at the trailing edge of  $\overline{WR}$ . A read of the port generates a pulse on the  $\overline{RD}$  output line and input data must be valid at the trailing edge of  $\overline{RD}$ . When not being written or read, the BUS lines are in a high impedance state. See also Sections 3.6 and 3.7.

### 2.1.5 Test and INT Inputs

Three pins serve as inputs and are testable with the conditional jump instruction. These are T0, T1, and  $\overline{INT}$ . These pins allow inputs

to cause program branches without the necessity to load an input port into the accumulator. The T0, T1, and  $\overline{INT}$  pins have other possible functions as well. See the pin description in Sec. 2.2.

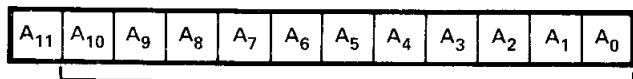
### 2.1.6 Program Counter and Stack

The Program Counter is an independent counter while the Program Counter Stack is implemented using pairs of registers in the Data Memory Array. Only 10 (or 11) bits of the Program Counter are used to address the 1024 (2048) words of on-board program memory while the most significant bits are used for external Program Memory fetches. The Program Counter is initialized to zero by activating the Reset line.

An interrupt or CALL to a subroutine causes the contents of the program counter to be stored in one of the 8 register pairs of the Program Counter Stack. The pair to be used is determined by a 3-bit Stack Pointer which is part of the Program Status Word (PSW). Data RAM locations 8 thru 23 are available as stack registers and are used to store the Program Counter and 4 bits of PSW as shown in the figure. The Stack Pointer when initialized to 000 points to RAM locations 8 and 9. The first subroutine jump or interrupt results in the program counter contents being transferred to locations 8 and 9 of the RAM array. The stack pointer is then incremented by one to point to locations 10 and 11 in anticipation of another CALL. Nesting of subroutines within subroutines can continue up to 8 times without overflowing the stack. If overflow does occur the deepest address stored (location 8 and 9) will be overwritten and lost since the stack pointer overflows from 111 to 000. It also underflows from 000 to 111.

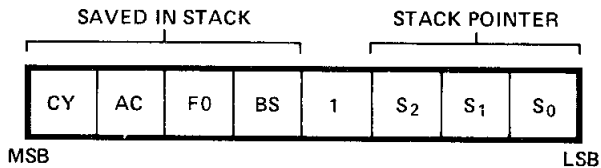
The end of a subroutine, which is signalled by a return instruction (RET or RETR), causes the Stack Pointer to be decremented and the contents of the resulting register pair to be transferred to the Program Counter.

# SINGLE COMPONENT SYSTEM



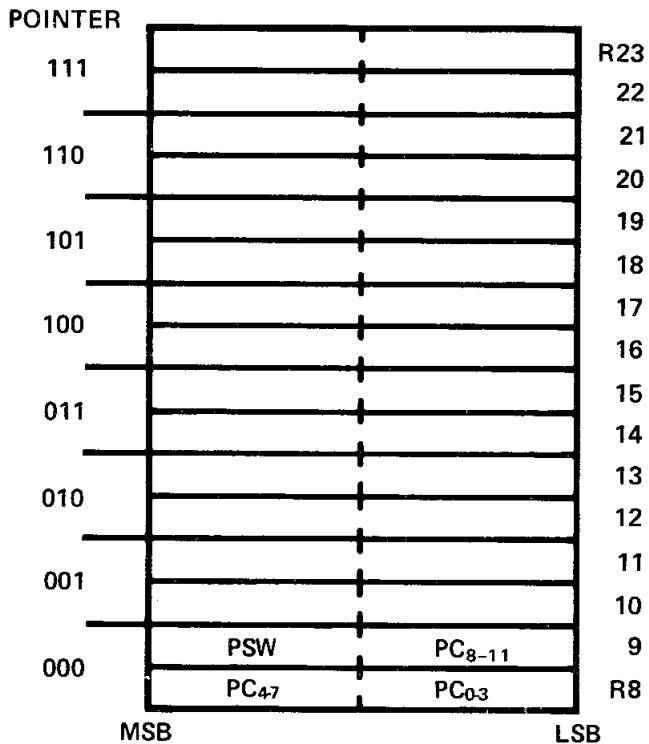
Conventional Program Counter  
 • Counts 000H to 7FFH  
 • Overflows 7FFH to 000H

## PROGRAM COUNTER



CY CARRY  
 AC AUXILLARY CARRY  
 F0 FLAG 0  
 BS REGISTER BANK SELECT

## PROGRAM STATUS WORD (PSW)



## PROGRAM COUNTER STACK

### 2.1.7 Program Status Word

An 8-bit status word which can be loaded to and from the accumulator exists called the Program Status Word (PSW). The accompanying figure shows the information available in the word. The Program Status Word is actually a collection of flip-flops throughout the machine which can be read or written as a whole. The ability to write to PSW allows for easy restoration of machine status after a power down sequence.

The upper four bits of PSW are stored in the Program Counter Stack with every call to subroutine or interrupt vector and are optionally restored upon return with the RETR instruction. The RET return instruction does not update PSW.

The PSW bit definitions are as follows:

- Bits 0 - 2: Stack Pointer bits (S<sub>0</sub>, S<sub>1</sub>, S<sub>2</sub>)
- Bit 3: Not used ("1" level when read)
- Bit 4: Working Register Bank Switch Bit (BS)  
0 = Bank 0  
1 = Bank 1
- Bit 5: Flag 0 bit (F0) user controlled flag which can be complemented or cleared, and tested with the conditional jump instruction JF0.
- Bit 6: Auxiliary Carry (AC) carry bit generated by an ADD instruction and used by the decimal adjust instruction DA A.
- Bit 7: Carry (CY) carry flag which indicates that the previous operation has resulted in overflow of the accumulator.

### 2.1.8 Conditional Branch Logic

The conditional branch logic within the processor enables several conditions internal and external to the processor to be tested by the users program. By using the conditional jump instruction the following conditions can effect a change in the sequence of the program execution.

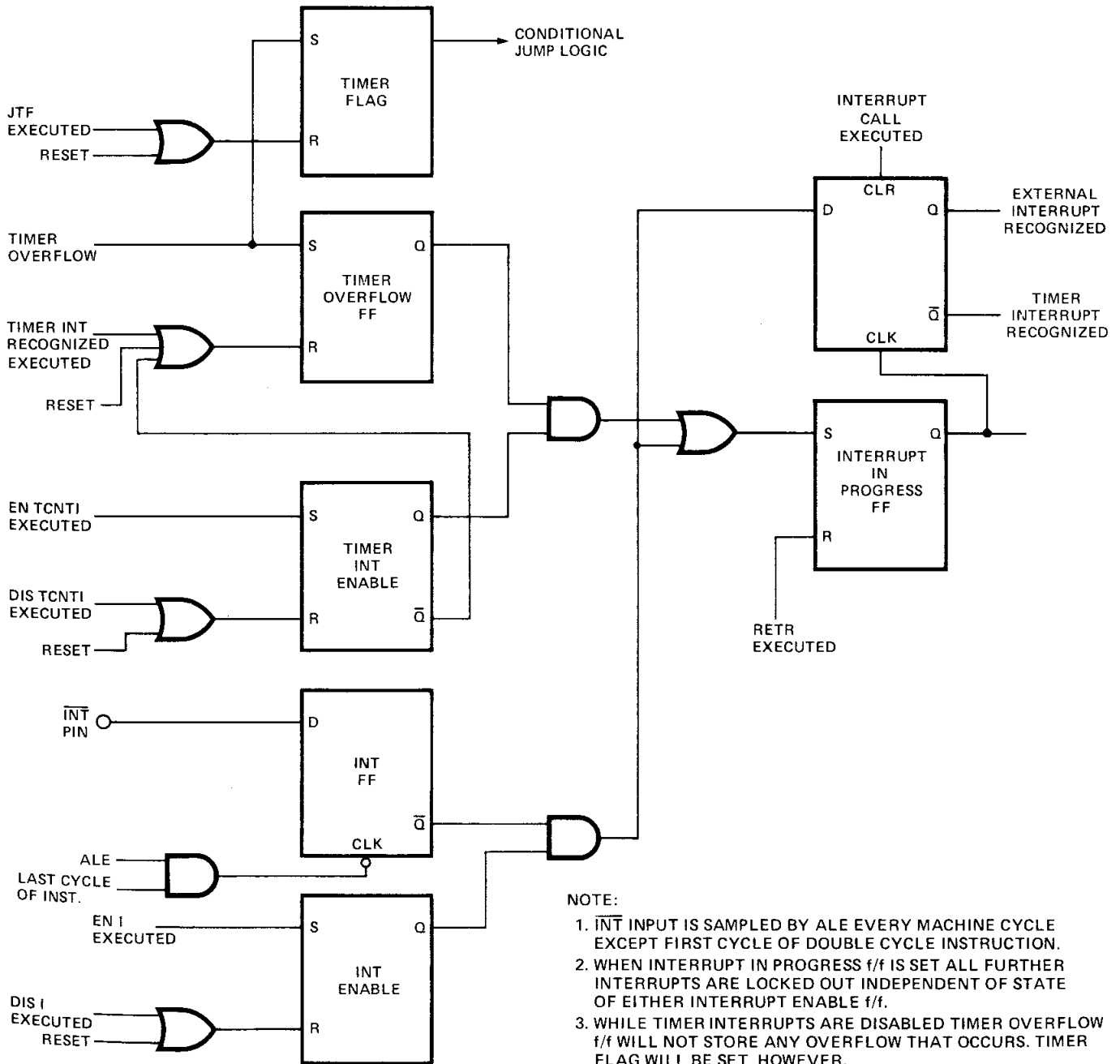


# SINGLE COMPONENT SYSTEM

Device Testable	Jump Conditions (Jump On)	
	All zeros	not all zeros
Accumulator	—	1
Accumulator Bit	—	1
Carry Flag	0	1
User Flags (F0, F1)	—	1
Timer Overflow Flag	—	1
Test Inputs (T0, T1)	0	1
Interrupt Input (INT)	0	—

## 2.1.9 Interrupt

An interrupt sequence is initiated by applying a low "0" level input to the  $\overline{\text{INT}}$  pin. Interrupt is level triggered and active low to allow "WIRE ORing" of several interrupt sources at the input pin. The Interrupt line is sampled every machine cycle during ALE and when detected causes a "jump to subroutine" at location 3 in program memory as soon as all cycles of the current instruction are complete. As in any CALL to subroutine, the Program Counter



## INTERRUPT LOGIC

and Program Status word are saved in the stack. For a description of this operation see the previous section, Program Counter and Stack. Program Memory location 3 usually contains an unconditional jump to an interrupt service subroutine elsewhere in program memory. The end of an interrupt service subroutine is signalled by the execution of a Return and Restore Status instruction RETR. The interrupt system is single level in that once an interrupt is detected all further interrupt requests are ignored until execution of an RETR re-enables the interrupt input logic. This occurs at the beginning of the second cycle of the RETR instruction. This sequence holds true also for an internal interrupt generated by timer overflow. If an internal timer/counter generated interrupt and an external interrupt are detected at the same time, the external source will be recognized. See the following Timer/Counter section for a description of timer interrupt. If needed, a second external interrupt can be created by enabling the timer/counter interrupt, loading FFH in the Counter (one less than terminal count), and enabling the event counter mode. A "1" to "0" transition on the T1 input will then cause an interrupt vector to location 7.

## Interrupt Timing

The interrupt input may be enabled or disabled under Program Control using the EN I and DIS I instructions. Interrupts are disabled by Reset and remain so until enabled by the users program. An interrupt request must be removed before the RETR instruction is executed upon return from the service routine otherwise the processor will re-enter the service routine immediately. Many peripheral devices prevent this situation by resetting their interrupt request line whenever the processor accesses (Reads or Writes) the peripherals data buffer register. If the interrupting device does not require access by the processor, one output line of the 8048 may be designated as an "interrupt acknowledge" which is activated by the service subroutine to reset the interrupt request. The  $\overline{\text{INT}}$  pin may also be tested using the conditional jump instruction JN1. This instruction may be used

to detect the presence of a pending interrupt before interrupts are enabled. If interrupt is left disabled,  $\overline{\text{INT}}$  may be used as another test input like T0 and T1.

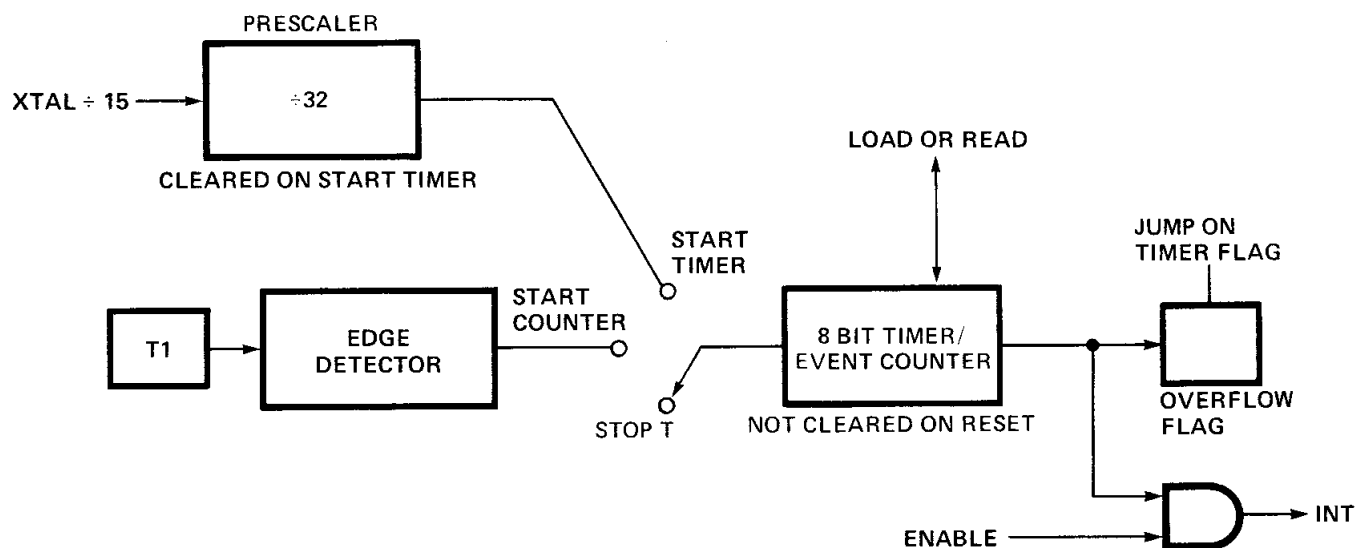
## 2.1.10 Timer/Counter

The 8048 contains a counter to aid the user in counting external events and generating accurate time delays without placing a burden on the processor for these functions. In both modes the counter operation is the same, the only difference being the source of the input to the counter.

### Counter

The 8-bit up binary counter is presetable and readable with two MOV instructions which transfer the contents of the accumulator to the counter and vice versa. The counter content is not affected by Reset and is initialized solely by the MOV T,A instruction. The counter is stopped by a Reset or STOP TCNT instruction and remains stopped until started as a timer by a START T instruction or as an event counter by a START CNT instruction. Once started the counter will increment to its maximum count (FF) and overflow to zero continuing its count until stopped by a STOP TCNT instruction or Reset.

The increment from maximum count to zero (overflow) results in the setting of an overflow flag flip-flop and in the generation of an interrupt request. The state of the overflow flag is testable with the conditional jump instruction JTF. The flag is reset by executing a JTF or by Reset. The interrupt request is stored in a latch and then ORed with the external interrupt input INT. The timer interrupt may be enabled or disabled independently of external interrupt by the EN TCNTI and DIS TCNTI instructions. If enabled, the counter overflow will cause a subroutine call to location 7 where the timer or counter service routine may be stored. If timer and external interrupts occur simultaneously, the external source will be recognized and the Call will be to location 3. Since the timer interrupt is latched it will remain pending until the external device is serviced and immediately be recognized upon return from the service routine. The



## TIMER/EVENT COUNTER

pending timer interrupt is reset by the Call to location 7 or may be removed by executing a DIS TCNTI instruction.

### As an Event Counter

Execution of a START CNT instruction connects the T1 input pin to the counter input and enables the counter. Subsequent high to low transitions on T1 will cause the counter to increment. The maximum rate at which the counter may be incremented is once per three instruction cycles (every  $7.5\mu\text{sec}$  when using a 6MHz crystal)—there is no minimum frequency. T1 input must remain high for at least 500ns (at 6MHz) after each transition.

### As a Timer

Execution of a START T instruction connects an internal clock to the counter input and enables the counter. The internal clock is derived by passing the basic 400 KHz machine cycle clock ALE through a  $\div 32$  prescaler. The prescaler is reset during the START T instruction. The resulting 12.5 KHz clock increments the counter every  $80\mu\text{sec}$  (assuming 6 MHz XTAL). Various delays between  $80\mu\text{sec}$  and 20 msec (256 counts) can be obtained by presetting the counter and detecting overflow. Times longer than 20 msec may be achieved by accumulating mul-

iple overflows in a register under software control. For time resolution less than  $80\mu\text{sec}$  an external clock can be applied to the T1 input and the counter operated in the event counter mode. ALE divided by 3 or more can serve as this external clock. Very small delays or “fine tuning” of larger delays can be easily accomplished by software delay loops.

### 2.1.11 Clock and Timing Circuits

Timing generation for the 8048 is completely self-contained with the exception of a frequency reference which can be XTAL, inductor, or external clock source. The Clock and Timing circuitry can be divided into the following functional blocks:

#### Oscillator

The on-board oscillator is a high gain series resonant circuit with a frequency range of 1 to 6MHz. The X1 external pin is the input to the amplifier stage while X2 is the output. A crystal or inductor connected between X1 and X2 provides the feedback and phase shift required for oscillation. A 5.9904 MHz crystal provides for easy derivation of all standard communications frequencies. If an accurate frequency reference and maximum processor speed are not required, an induc-

# SINGLE COMPONENT SYSTEM

tor may be used in place of the crystal. With an inductor the oscillator frequency can be approximately 3 to 5 MHz. For higher speed operation a crystal should be used. An externally generated clock may also be applied to X1-X2 as the frequency source.

## State Counter

The output of the oscillator is divided by 3 in the State Counter to create a clock which defines the state times of the machine (CLK). CLK can be made available on the external pin T0 by executing an ENTO CLK instruction. The output of CLK on T0 is disabled by Reset of the processor.

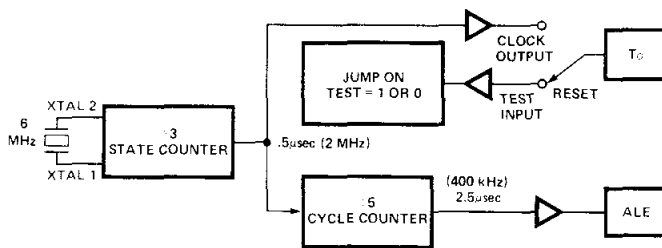
## Cycle Counter

CLK is then divided by 5 in the Cycle Counter to provide a clock which defines a machine cycle consisting of 5 machine states. This clock is called Address Latch Enable (ALE) because of its function in MCS-48 systems with external memory. It is provided continuously on the ALE output pin.

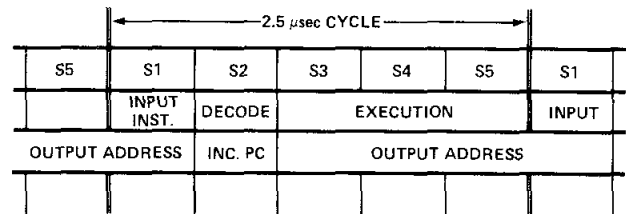
### 2.1.12 Reset

The reset input provides a means for initialization for the processor. This Schmitt-trigger input has an internal pullup resistor which in combination with an external 1  $\mu$ f capacitor provides an internal reset pulse of sufficient length to guarantee all circuitry is reset. If the

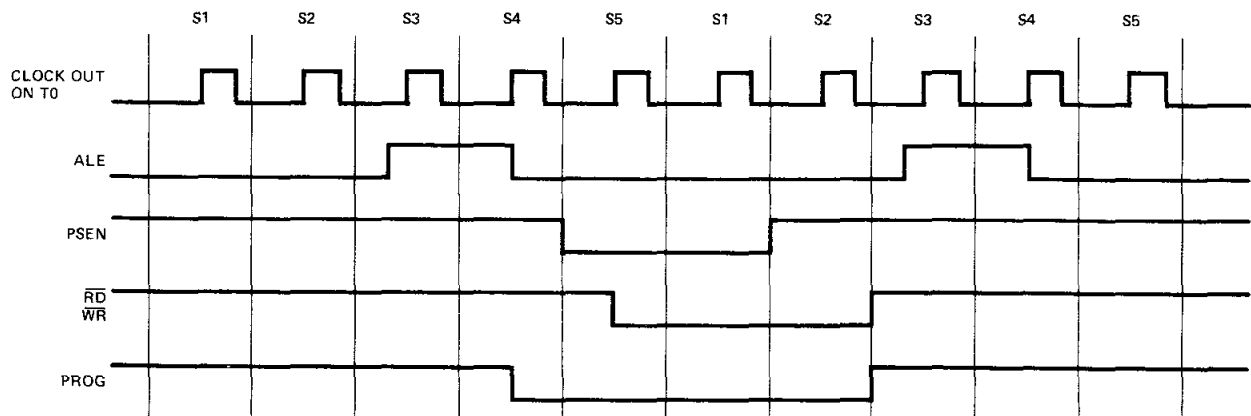
## DIAGRAM OF 8048 CLOCK UTILITIES



## INSTRUCTION CYCLE



## MCS-48™ CYCLE TIMING



# SINGLE COMPONENT SYSTEM

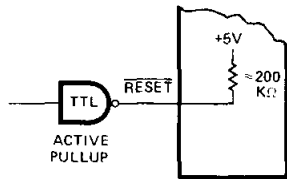
INSTRUCTION	CYCLE 1					CYCLE 2				
	S1	S2	S3	S4	S5	S1	S2	S3	S4	S5
IN A, P	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER		INCREMENT TIMER			READ PORT			
OUTL P, A	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER		INCREMENT TIMER	OUTPUT TO PORT					
ANL P, :DATA	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER		INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA		* INCREMENT PROGRAM COUNTER	OUTPUT TO PORT	
ORL P, :DATA	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER		INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA		* INCREMENT PROGRAM COUNTER	OUTPUT TO PORT	
INS A, BUS	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER		INCREMENT TIMER			READ PORT			
OUTL BUS, A	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER		INCREMENT TIMER	OUTPUT TO PORT					
ANL BUS, :DATA	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER		INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA		* INCREMENT PROGRAM COUNTER	OUTPUT TO PORT	
ORL BUS, :DATA	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER		INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA		* INCREMENT PROGRAM COUNTER	OUTPUT TO PORT	
MOVX @R, A	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER	OUTPUT RAM ADDRESS	INCREMENT TIMER	OUTPUT DATA TO RAM					
MOVX A, @R	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER	OUTPUT RAM ADDRESS	INCREMENT TIMER			READ DATA			
MOVD A, P1	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER	OUTPUT OP CODE/ADDRESS	INCREMENT TIMER			READ P2 LOWER			
MOVD P1, A	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER	OUTPUT OP CODE/ADDRESS	INCREMENT TIMER	OUTPUT DATA TO P2 LOWER					
ANLD P, A	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER	OUTPUT OP CODE/ADDRESS	INCREMENT TIMER	OUTPUT DATA					
ORLD P, A	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER	OUTPUT OP CODE/ADDRESS	INCREMENT TIMER	OUTPUT DATA					
J (CONDITIONAL)	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER	SAMPLE CONDITION	INCREMENT TIMER		FETCH IMMEDIATE DATA		* UPDATE PROGRAM COUNTER		
STRT T	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER			START COUNTER					
STRT CNT	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER			STOP COUNTER					
STOP TCNT	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER								
ENI	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER		ENABLE INTERRUPT						
DIS I	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER		DISABLE INTERRUPT						
ENTO CLK	FETCH INSTRUCTION	* INCREMENT PROGRAM COUNTER		ENABLE CLOCK						

\*VALID INSTRUCTION ADDRESSES ARE OUTPUT AT THIS TIME  
IF EXTERNAL PROGRAM MEMORY IS BEING ACCESSED.

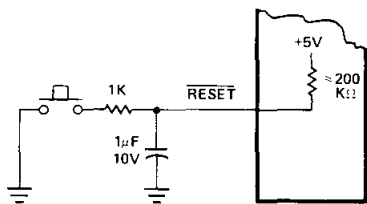
**INSTRUCTION TIMING DIAGRAM**

reset pulse is generated externally the reset pin must be held at ground (.5V) for at least 50 milliseconds after the power supply is within tolerance. Only 5 machine cycles (12.5 $\mu$ s @ 6 MHz) are required if power is already on and the oscillator has stabilized.

### EXTERNAL RESET



### POWER ON RESET



Reset performs the following functions:

1. Sets program counter to zero.
2. Sets stack pointer to zero.
3. Selects register bank 0.
4. Selects memory bank 0.
5. Sets BUS to high impedance state. (except when EA = 5V)
6. Sets Ports 1 and 2 to input mode.
7. Disables interrupts (timer and external)
8. Stops timer.
9. Clears timer flag.
10. Clears F0 and F1.
11. Disables clock output from T0.

### 2.1.13 Single-Step

This feature provides the user with a debug capability in that the processor can be stepped through the program one instruction at a time. While stopped, the address of the next instruction to be fetched is available concurrently on BUS and the lower half of Port 2. The user can therefore follow the program through each of the instruction steps. A timing diagram, showing the interaction between output ALE and input SS is shown. The BUS buffer contents are lost during single step, however, a latch may be added to re-establish the lost I/O capability if needed. (See 2.4.1).

### Timing

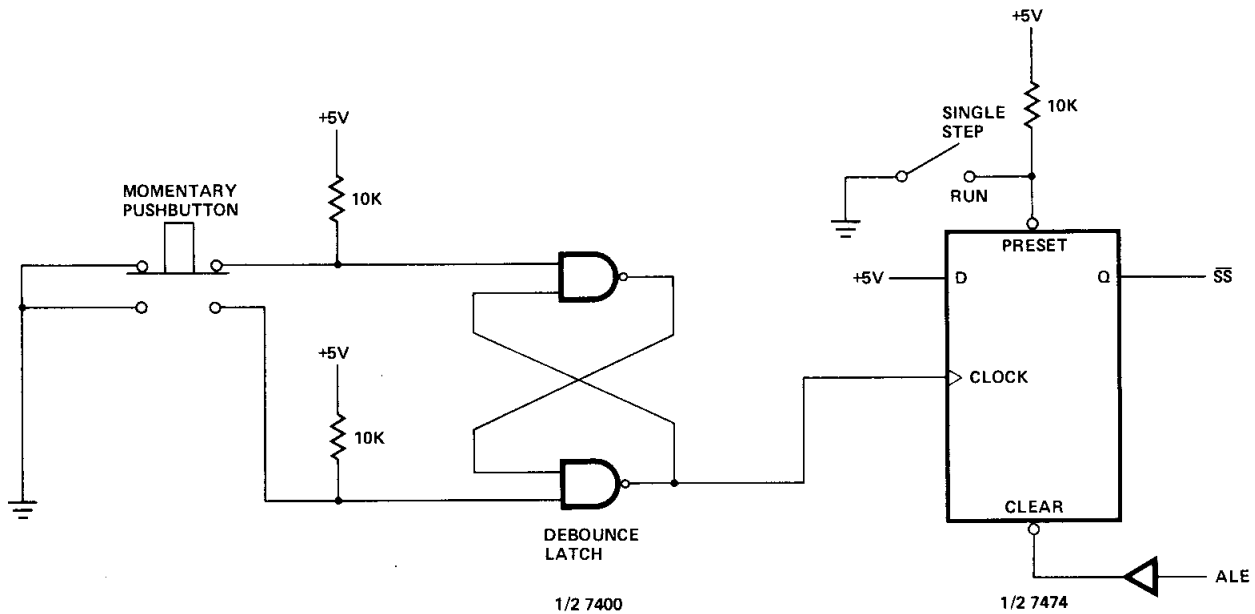
The 8048 operates in a single-step mode as follows:

1. The processor is requested to stop by applying a low level on  $\overline{SS}$ .
2. The processor responds by stopping during the instruction fetch portion of the next instruction. If a double cycle instruction is in progress when the single step command is received, both cycles will be completed before stopping.
3. The processor acknowledges it has entered the stopped state by raising ALE high. In this state (which can be maintained indefinitely) the address of the next instruction to be fetched is present on BUS and the lower half of port 2.
4.  $\overline{SS}$  is then raised high to bring the processor out of the stopped mode allowing it to fetch the next instruction. The exit from stop is indicated by the processor bringing ALE low.
5. To stop the processor at the next instruction  $\overline{SS}$  must be brought low again as soon as ALE goes low. If  $\overline{SS}$  is left high the processor remains in a "Run" mode.

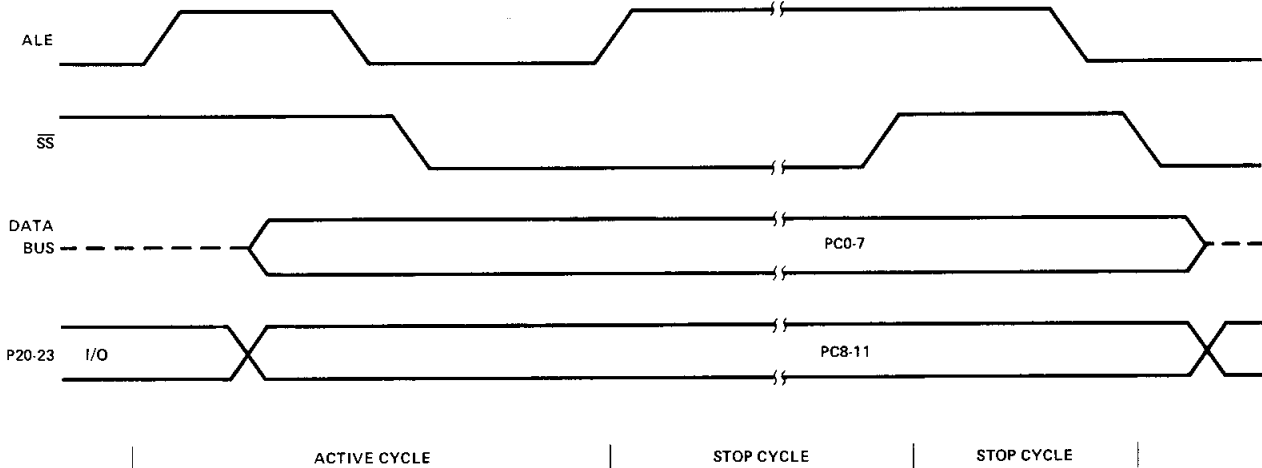
A diagram for implementing the single step function of the 8748 is shown. A D-type flip-flop with preset and clear is used to generate  $\overline{SS}$ . In the run mode  $\overline{SS}$  is held high by keeping the flip-flop preset (preset has precedence over the clear input). To enter single step, preset is removed allowing ALE to bring  $\overline{SS}$  low via the clear input. ALE should be buffered since the clear input of an SN7474 is the equivalent of 3 TTL loads. The processor is now in the stopped state. The next instruction is initiated by clocking a "1" into the flip-flop. This "1" will not appear on  $\overline{SS}$  unless ALE is high removing clear from the flip-flop. In response to  $\overline{SS}$  going high the processor begins an instruction fetch which brings ALE low resetting  $\overline{SS}$  through the clear input and causing the processor to again enter the stopped state.

# SINGLE COMPONENT SYSTEM

## SINGLE STEP CIRCUIT



## SINGLE STEP TIMING

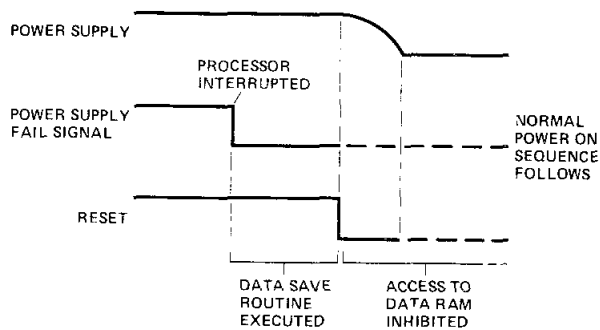


### 2.1.14 Power Down Mode (8048, 8049, 8039, 8035L)

Extra circuitry has been added to the 8048 ROM version to allow power to be removed from all but the 64/128 x 8 data RAM array for low power standby operation. In the power down mode the contents of data RAM can be maintained while drawing typically 10% to 15% of normal operating power requirements.

$V_{CC}$  serves as the 5V supply pin for the bulk of 8048 circuitry while the  $V_{DD}$  pin supplies only the RAM array. In normal operation both pins are at 5V while in standby  $V_{CC}$  is at ground and only  $V_{DD}$  is maintained at 5V. Applying Reset to the processor through the Reset pin inhibits any access to the RAM by the processor and guarantees that RAM cannot be inadvertently altered as power is removed from  $V_{CC}$ .

# SINGLE COMPONENT SYSTEM



## POWER DOWN SEQUENCE

A typical power down sequence occurs as follows:

1. Imminent power supply failure is detected by user defined circuitry. Signal must be early enough to allow 8048 to save all necessary data before  $V_{CC}$  falls below normal operating limits.
2. Power fail signal is used to interrupt processor and vector it to a power fail service routine.
3. Power fail routine saves all important data and machine status in the internal data RAM array. Routine may also initiate transfer of backup supply to the  $V_{DD}$  pin and indicate to external circuitry that power fail routine is complete.
4. Reset is applied to guarantee data will not be altered as the power supply falls out of limits. Reset must be held low until  $V_{CC}$  is at ground level.

Recovery from the Power Down mode can occur as any other power-on sequence with an external capacitor on the Reset input providing the necessary delay. See the previous section on Reset.

### 2.1.15 External Access Mode

Normally the first 1K (8048) or 2K (8049) words of program memory are automatically fetched from internal ROM or EPROM. The EA input pin however allows the user to

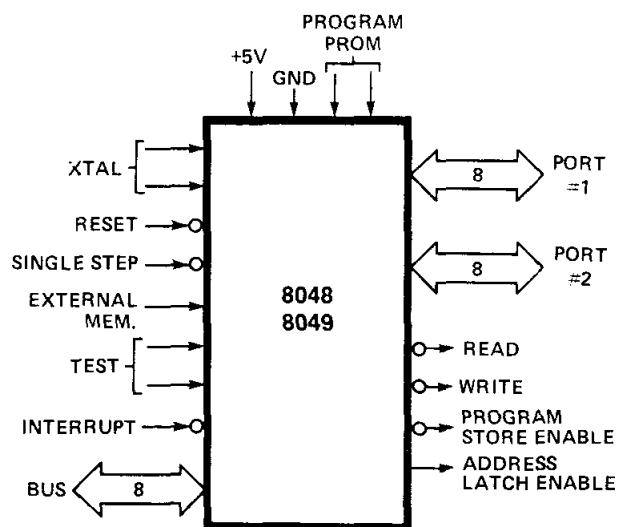
effectively disable internal program memory by forcing all program memory fetches to reference external memory. The following chapter explains how access to external program memory is accomplished.

The External Access mode is very useful in system test and debug because it allows the user to disable his internal applications program and substitute an external program of his choice—a diagnostic routine for instance. In addition, the section on Test and Debug explains how internal program memory can be read externally, independent of the processor.

A "1" level on EA initiates the external access mode. For proper operation, Reset should be applied while the EA input is changed.

## 2.2 Pin Description

The MCS-48 processors (except 8021) are packaged in 40 pin Dual In-Line Packages (DIP's). The following is a summary of the functions of each pin. Where it exists, the second paragraph describes each pin's function in an expanded MCS-48 system. Unless otherwise specified, each input is TTL compatible and each output will drive one standard TTL load.



8048 LOGIC SYMBOL



## SINGLE COMPONENT SYSTEM

Designation	Pin Number	Function
V <sub>SS</sub>	20	Circuit GND potential
V <sub>DD</sub>	26	Programming power supply; +25V during program, +5V during operation for both ROM and PROM. Low power standby pin in 8048 ROM version
V <sub>CC</sub>	40	Main power supply; +5V during operation and 8748 programming.
PROG	25	Program pulse (+25V) input pin during 8748 programming. Output strobe for 8243 I/O expander.
P10-P17 (Port 1)	27-34	8-bit quasi-bidirectional port. (Internal Pullup $\approx$ 50K $\Omega$ )
P20-P27 (Port 2)	21-24 35-38	8-bit quasi-bidirectional port. (Internal Pullup $\approx$ 50K $\Omega$ )  P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.
D0-D7 (BUS)	12-19	True bidirectional port which can be written or read synchronously using the $\overline{RD}$ , $\overline{WR}$ strobes. The port can also be statically latched.  Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, $\overline{RD}$ , and $\overline{WR}$ .
T0	1	Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENTO CLK instruction. T0 is also used during programming.
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the event counter input using the STRT CNT instruction.
$\overline{INT}$	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. (Active low)
$\overline{RD}$	8	Output strobe activated during a BUS read. Can be used to enable data onto the BUS from an external device. (Active low)  Used as a Read Strobe to External Data Memory.

## SINGLE COMPONENT SYSTEM

Designation	Pin Number	Function
RESET	4	Input which is used to initialize the processor. Also used during PROM programming and verification. (Active low) (Internal pullup $\approx 200K\Omega$ )
$\overline{WR}$	10	Output strobe during a BUS write. (Active low) Used as write strobe to external data memory.
ALE	11	Address Latch Enable. This signal occurs once during each cycle and is useful as a clock output. The negative edge of ALE strobes address into external data and program memory.
$\overline{PSEN}$	9	Program Store Enable. This output occurs only during a fetch to external program memory. (Active Low)
$\overline{SS}$	5	Single step input can be used in conjunction with ALE to "single step" the processor through each instruction. (Active Low) (Internal pullup $\approx 300K\Omega$ )
EA	7	External Access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing and program verification. (Active High) (Internal pullup $\approx 10M\Omega$ on 8048/8049, 8035L, 8039 only)
XTAL1	2	One side of crystal input for internal oscillator. Also input for external source.
XTAL2	3	Other side of crystal input.

Unless otherwise stated inputs do not have internal pullup resistors.

### 2.3 Programming, Verifying and Erasing EPROM

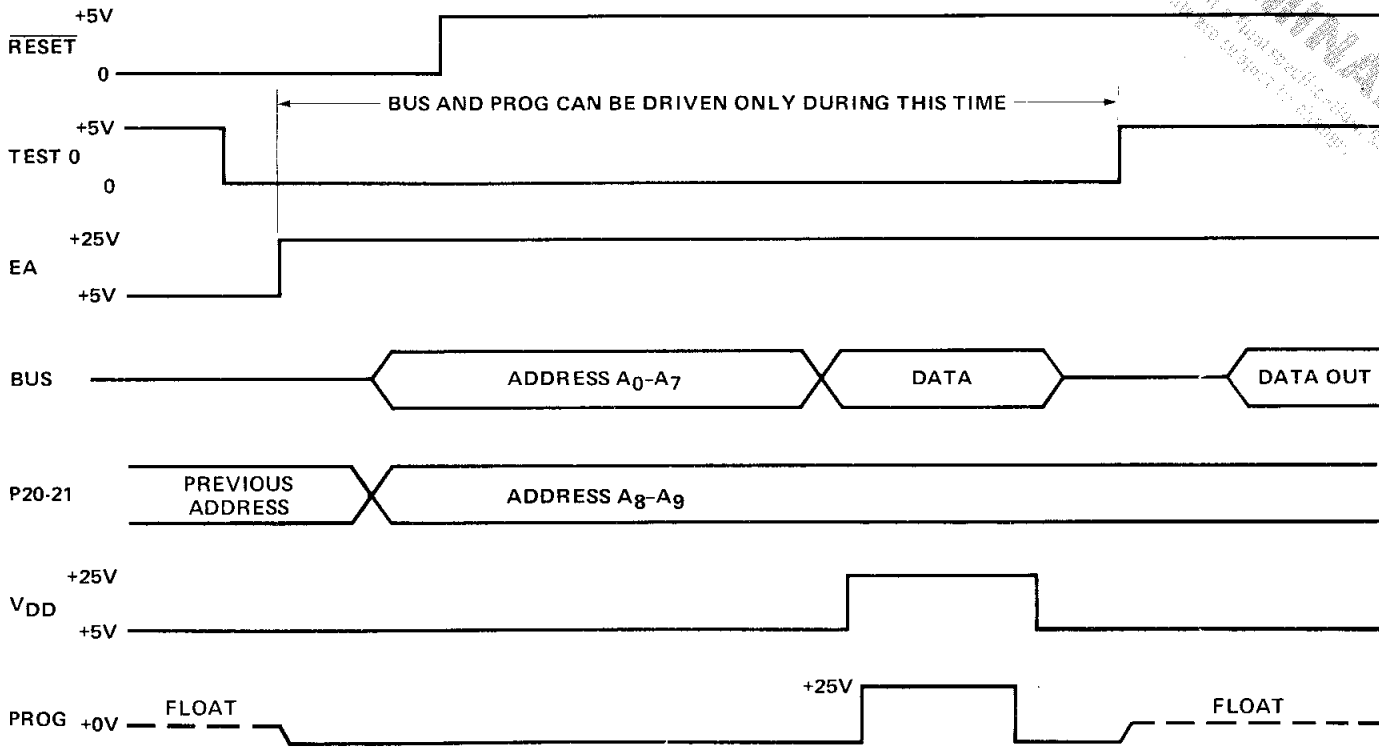
The internal Program Memory of the 8748 may be erased and reprogrammed by the user as explained in the following sections:

#### 2.3.1 Programming/Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

Pin	Function
XTAL 1	Clock Input (1 to 6MHz)
$\overline{Reset}$	Initialization and Address Latching
Test 0	Selection of Program or Verify Mode
EA	Activation of Program/Verify Modes
BUS	Address and Data Input Data Output During Verify
P20-1	Address Input
V <sub>DD</sub>	Programming Power Supply
PROG	Program Pulse Input

# SINGLE COMPONENT SYSTEM



SEE 8048/8748 DATA SHEET (CHAPTER 6) FOR DETAIL TIMING SPECIFICATIONS.

**WARNING:** An attempt to program a mis-socketed 8748 will result in severe damage to the part. An indication of a properly socketed part is the appearance of the ALE clock output. The lack of this clock may be used to disable the programmer.

## PROGRAMMING/VERIFY SEQUENCE

### 8748 Erasure Characteristics

The erasure characteristics of the 8748 are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms ( $\text{\AA}$ ). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000-4000 $\text{\AA}$  range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8748 in approximately 3 years while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 8748 is to be exposed to these types of lighting conditions for extended periods of time, opaque labels are available from Intel which should be placed over the 8748 window to prevent unintentional erasure.

When erased, bits of the 8748 Program Memory are in the logic "0" state.

The recommended erasure procedure for the 8748 is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms ( $\text{\AA}$ ). The integrated dose (i.e., UV intensity X exposure time) for erasure should be a minimum of 15W-sec/cm<sup>2</sup>. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000 $\mu\text{W}/\text{cm}^2$  power rating. The 8748 should be placed within one inch from the lamp tubes during erasure. Some lamps have a filter on their tubes and this filter should be removed before erasure.

The Program/Verify sequence is:

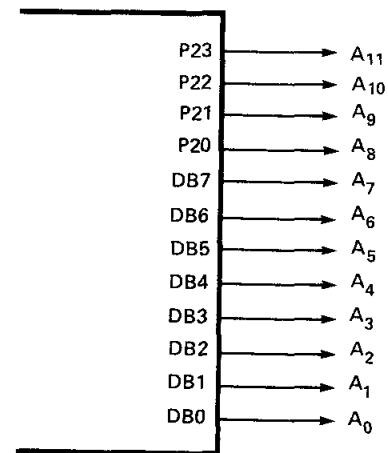
1.  $V_{DD} = 5v$ , Clock applied or internal oscillator operating, Reset = 0v Test 0 = 5v, EA = 5v, BUS and PROG floating
2. Insert 8748 in programming socket
3. Test 0 = 0v (Select Program Mode)
4. EA = 25v (Activate Program Mode)
5. Address applied to BUS and P20-1
6.  $\overline{\text{Reset}} = 5v$  (Latch Address)
7. Data applied to BUS
8.  $V_{DD} = 25v$  (Programming Power)
9. PROG = 0v followed by one 50ms pulse to 25v
10.  $V_{DD} = 5v$
11. TEST 0 = 5v (Verify Mode)
12. Read and Verify Data on BUS
13. TEST 0 = 0v
14.  $\overline{\text{Reset}} = 0v$  and repeat from Step 5
15. Programmer should be at conditions of Step 1 when 8748 is removed from socket.

## 2.4 Test and Debug

Several MCS-48 features described in the previous sections are discussed here to emphasize their use in testing MCS-48 components and in debugging MCS-48 based systems.

### 2.4.1 Single Step

Single step circuitry within the microcomputer in combination with the external circuitry described in Section 2.1.13 allows the user to execute one instruction at a time whether the instruction is one or two cycles in length. After completion of the instruction the processor halts with the address of the next instruction to be fetched available on the eight lines of BUS and the lower 4-bits of port 2.



### ADDRESS OUTPUT DURING SINGLE STEP

This allows the user to step through his program and note the sequence of instructions being executed.

While the processor is stopped, the I/O information on BUS and the 4-bits of port 2 is, of course, not available. I/O information is, however, valid at the leading edge of ALE and can be latched externally using this signal if necessary.

### 2.4.2 Disabling Internal Program Memory

Applying +5V to the EA (external access) pin of the MCS-48 microcomputers allows the user to effectively disable internal program memory by forcing all instruction fetches to occur from an external memory. This external memory can be connected as explained in the section on program memory expansion and can contain a diagnostic routine to exercise the processor, the internal RAM, the timer, and the I/O lines. EA should be switched only when the processor is in RESET.

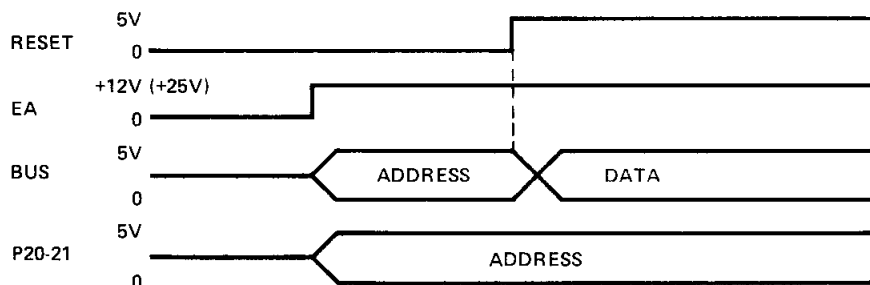
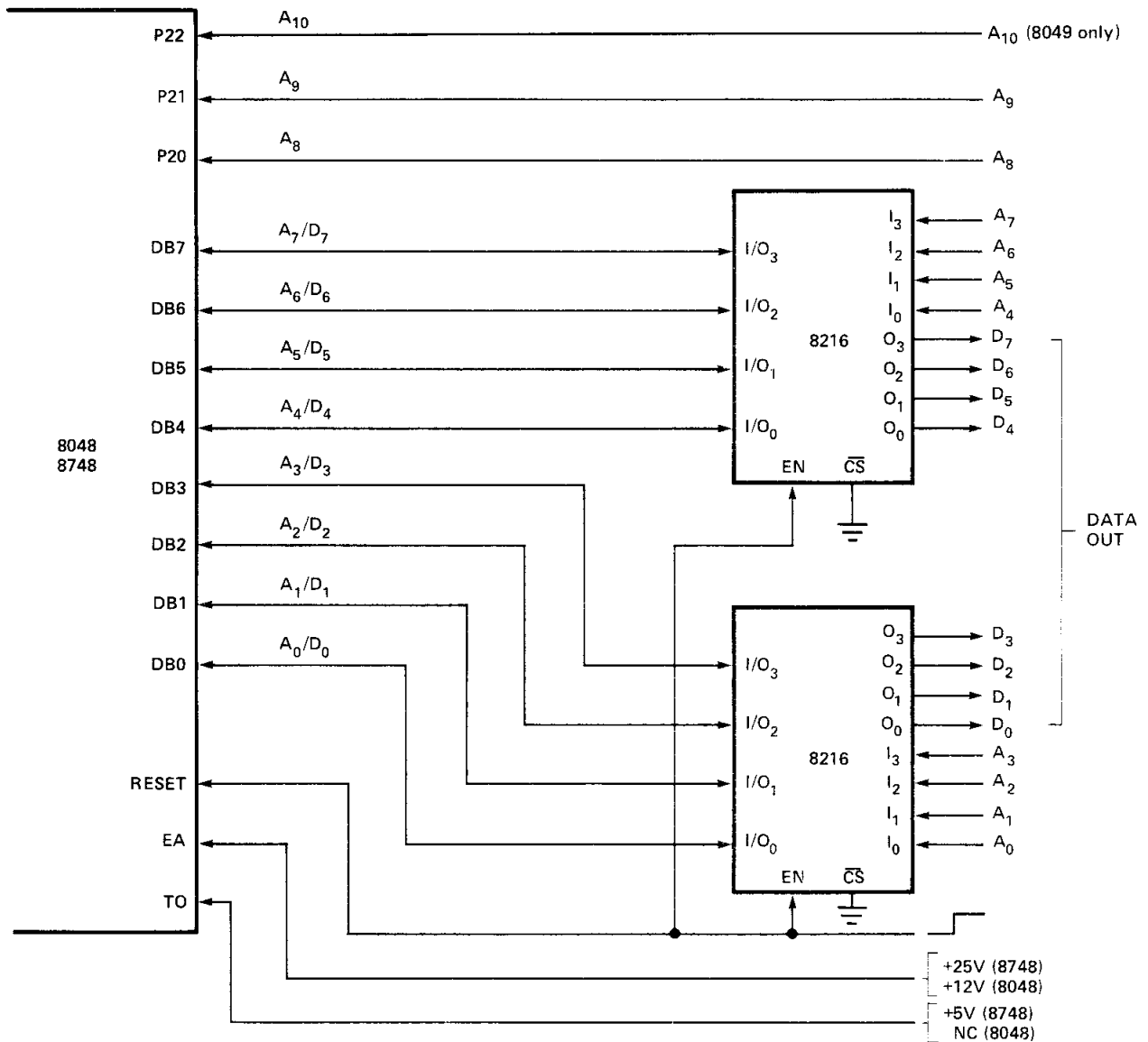
### 2.4.3 Reading Internal Program Memory

Just as the processor may be isolated from internal program memory using EA, program memory can be read independent of the processor using the verification mode described in the previous section, Programming/Verification.

## SINGLE COMPONENT SYSTEM

The processor is placed in the READ mode by applying a high voltage (+25V for the 8748, +12V for the 8048/8049) to the EA pin and +5V to the T0 (8748 only) input pin. RESET must be at 0V when voltage is applied to EA. The address of the location to be read is then applied to the same lines

(TTL levels) of BUS and Port 2 which output the address during single step (see below). The address is latched by a "0" to "1" transition on RESET and a high level on RESET causes the contents of the program memory location addressed to appear on the eight lines of BUS.



### READING INTERNAL PROGRAM MEMORY

## 8021 Functional Specifications

The following is a functional description of the major elements of the 8021.

### 2.5 Program Memory

The 8021 contains 1K x 8 of mask programmable ROM. No external ROM expansion capability is provided.

### 2.6 Data Memory

A 64 x 8 dynamic RAM is located on chip for data storage. All locations are indirectly addressable and eight designated locations are directly addressable. Also, included in the memory is the address stack, addressed by a 3-bit stack pointer.

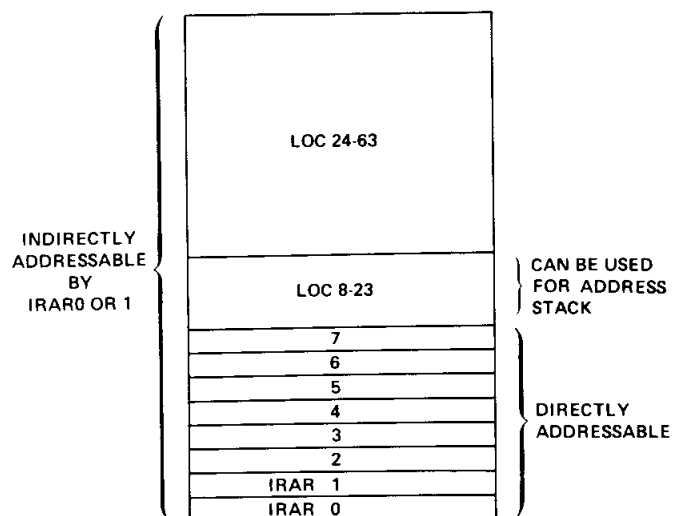
Memory is organized as shown in Figure 1. The least significant 8 addresses, 0-7, are directly addressable by any of the 11 direct register instructions. The locations are readily accessible for a variety of operations with the least number of instruction bytes required for their manipulation.

Registers 0 and 1 have another function, in that they can be used to indirectly address all locations in memory, using the indirect register instructions. These indirect RAM address registers, IRAR's, are especially useful for repetitive-type operations on adjacent memory locations. The indirect register instruction specifies which IRAR to use, and the contents of the IRAR is used to address a location in RAM. The contents of the addressed location is used during the execution of the instruction and may be modified. A value larger than 63 should not be preset in the IRAR when selected by an indirect register instruction. IRAR's may point to address 0-7, if desired.

Locations 8-23 may be used as the address stack. The address stack enables the processor to keep track of the return addresses generated from CALL instructions. A 3-bit stack pointer (SP) supplies the address of the locations to be loaded with

the next return address generated. The SP to this pushdown stack is incremented by one after a return address is stored, and decremented by one before an address is fetched during a RET. A total of 8 levels of nesting is possible. The SP is initialized to location 8 upon RESET. Since each address is 10-bits long, two bytes must be used to store a single address. The SP is incremented and decremented by one, but each increment or decrement moves the address pointed to by two. Therefore, only even numbered addresses are pointed to.

If a particular application does not require 8 levels of nesting, the unused portion of the stack may be used as any other indirectly addressable scratchpad location. For example, if only 3 levels of subroutine nesting are used, then only locations 8-13 need be reserved for the address stack, and locations 14-63 can be used for data storage. The unincremented program counter address is stored in the address stack. The stack contents is incremented before being loaded into the program counter during a return from subroutine.



**FIGURE 1. INTERNAL RAM ORGANIZATION**

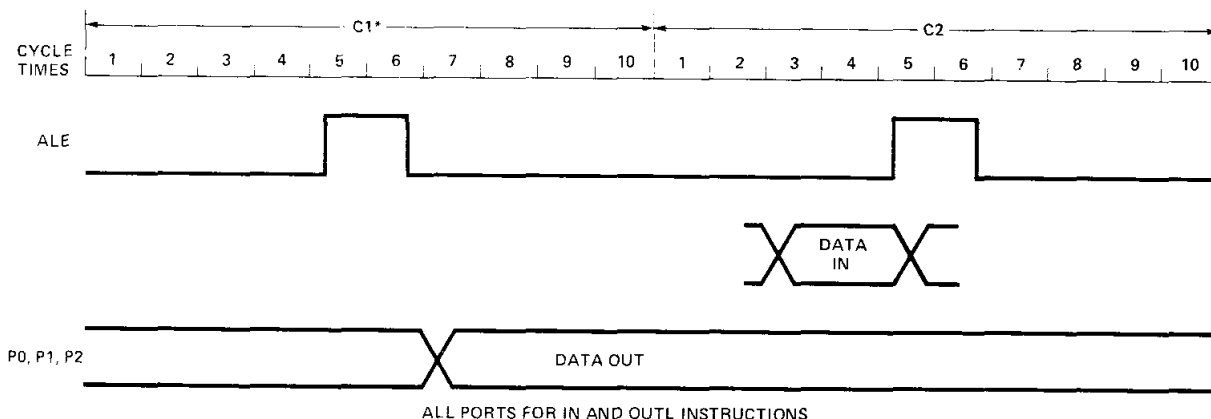
## 2.7 Oscillator and Clock

The 8021 contains its own onboard oscillator and clock circuit, requiring only an external timing control element. This control element can be a crystal, inductor, resistor, or clock in. The capacitor normally required in resistor or inductor timing control operation is integrated onto the 8021. All internal time slots are derived from the external element, and all outputs are a function of the oscillator frequency. Pins XTAL1 and XTAL2 are used to input the particular control element. An instruction cycle consists of 10 states, and each state is a time slot of 3 oscillator periods. (See Figure 2) Therefore, to obtain a 10  $\mu$ sec instruction cycle, a 3 MHz crystal should be used. An oscillator frequency of 3 MHz may also be obtained by connecting a 10K $\Omega$  resistor between XTAL1 and XTAL2. Note that the required resistance may vary from 10K $\Omega$ , and should be adjusted as necessary.

The 8021 utilizes dynamic RAM and certain other dynamic logic. Due to the clocking required with dynamic circuits, the oscillator frequency must be equal to or greater than 600K Hz, or improper operation may occur.

## 2.8 Timer/Event Counter

The 8021 has internal timer/event counter circuits that can monitor elapsed time or count external events that occur during program execution. The circuit has an 8-bit binary up-counter that is presettable and readable with two MOV instructions. These instructions transfer the contents of the accumulator to the counter and vice-versa. The counter content is not affected by Reset, and is initialized solely by the MOV T,A instruction. The counter is stopped by a RESET or STOP TCNT instruction and remains stopped until started as a timer by a STRT T instruction or as an event counter by a STRT CNT instruction. Once started,



**FIGURE 2. 8021 TIMING DIAGRAM**

the counter increments to its maximum count (FF), and overflows to zero. The count continues until stopped by a STOP TCNT instruction or RESET. The increment from maximum count to zero (overflow) sets an overflow flag. The state of the overflow flag is testable with the conditional jump instruction JTF. The flag is reset by JTF or by executing a RESET.

By a MOV T,A instruction, the contents of the accumulator are loaded to the timer. At the STRT T command an internal prescaler is zeroed and thereafter increments once each 30 input clocks (once each single cycle instruction, twice each double cycle instruction). The prescaler is a divide by 32. At the (11111) to (00000) transition the timer is incremented. The timer is 8-bits and an overflow (FFH) to (00H) timer flag is set. A conditional branch instruction (JTF) is available for testing this flag, the flag being reset each test. Total count capacity for the timer is  $2^8 \times 2^5 = 8192$  or 81.9 msec at a 10  $\mu$ sec cycle time. Contents of the timer are moved to the accumulator by the MOV A,T instruction without disturbing the counting process. The timer stops upon the STOP TCNT instruction.

The STRT CNT instruction connects the T1 input pin to the event counter input and enables the counter. Subsequent high-to-low transitions on T1 increment the counter. The maximum rate at which the counter can increment is once per three instruction cycles (30 $\mu$ s for a 3 MHz oscillator). There is no minimum frequency. T1 input must remain high for at least 500ns after each transition. The event counter is stopped by a STOP TCNT instruction.

## 2.9 Input/Output Capabilities

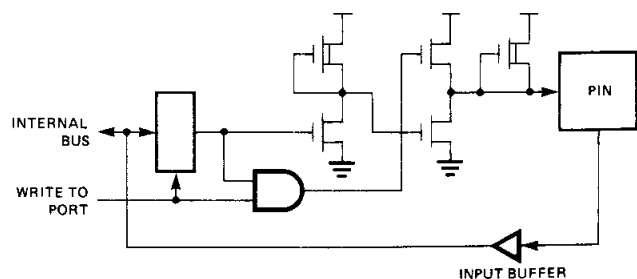
The 8021 I/O configurations are highly flexible. A number of different configurations are possible, tailoring an 8021 to a given task. Other than the power supply and dedicated pins, all other pins (20) can be used for input, output, or both, depending on the configuration.

All ports are quasi-bidirectional to facilitate stand-alone use. A simplified sche-

matic of the quasi-bidirectional interface is shown in Figure 3. This configuration allows buffered outputs, and also allows external input. Data written to these ports is statically latched and remains unchanged until rewritten. As input ports these lines are non-latching, i.e., inputs must be present until read by an input instruction. When writing a "0" or low value to these ports, the large pulldown device sinks an external TTL load. When writing a "1", a large current is supplied through the large pullup device to allow a fast data transfer. After a short time (less than one instruction cycle), the large device is shut off and the small pullup maintains the "1" level indefinitely. However, in this situation, an input device capable of overriding the small amount of sustaining current supplied by the pullup device can be read. (Alternatively, the data written can be read). So, by writing a "1" to any particular pin, that pin can serve either as a true high-level latched output pin, or as just a pullup resistor on an input. This allows maximum user flexibility in selecting his input or latched output pins, with a minimum of external components.

Port 00-07 is also quasi-bidirectional, except there is no large pullup device. As outputs, this port is essentially open drain.

By mask option the small pullup devices on P00-P07 may be deleted on any pin providing a true open drain output. This is useful in driving analog circuits and certain loads, such as keyboards.



**FIGURE 3. QUASI-BIDIRECTIONAL PORT STRUCTURE**



## 2.9.1 T1 Input

The 8021 T1 input line can be used as an input for the following functions:

- Event Counter (external input)
- Test input for branch instructions
- Zero voltage crossing detection

The operation of T1 as an input to the Event Counter is described in the Timer/Event Counter section. When used as a test input, the JT1 and JNT1 instructions test for 1 and 0 levels, respectively.

The T1 pin can also be used to detect the zero crossing of slowly moving AC signals (60 Hz). The self-biasing circuit shown in Figure 4 permits the Test 1 input to detect when the input voltage crosses zero within  $\pm 5\%$ , then the voltage is coupled through a  $1.0\mu\text{f}$  capacitor. Maximum input voltage is 3V peak-to-peak. The zero cross detection is especially useful in SCR control of 60 Hz power and in developing time-of-day and other timing routines. As a ROM mask option there is a pullup resistor that is useful for switch contact input or standard TTL.

## 2.9.2 High Current Outputs

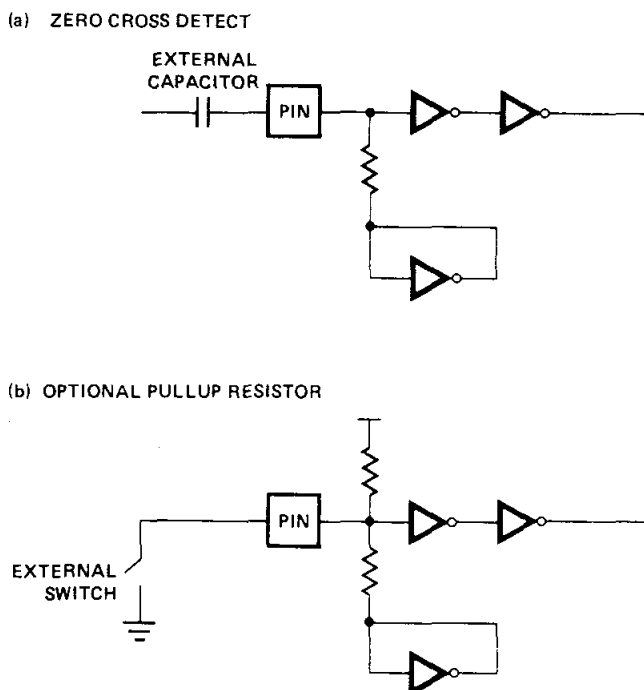
Very high current drive is desirable for

minimizing external parts required to do high power control. P10 and P11 have been designated high drive outputs capable of sinking 7mA at  $V_{SS} + 2.5$  volts. (For clarity, this is 7mA to  $V_{SS}$  with a 2.5 volt drop across the buffer.) These pins may, of course, be paralleled for 14mA drive if the output logic states are always the same.

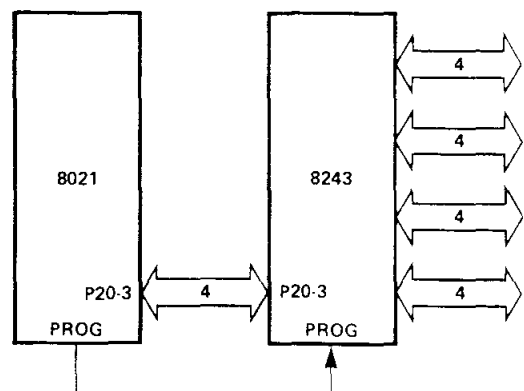
## 2.9.3 Expanded I/O

The 8021 can be used with the 8243 I/O expander chip, which provides additional I/O capability with a limited number of overhead pins. This chip has 4 directly addressable 4-bit ports. It connects to the PROG pin, which provides a clock, and pins P20-P23, which provide address and data. These ports can be written with a MOVD P,A; ANLD P,A; and ORLD P,A for Ports 4-7. A high to low transition on PROG signifies that address and control are available on P20-P23. The previous data on P20-P23 before an output expander instruction is lost. Therefore, when using an output expander P20-P23 are not useful for general input/output. Reading is via the MOVD A,P. This circuit configuration is shown in Figure 5. The timing diagram is shown in Figure 6.

The 8021 can also use standard low cost TTL to expand the number of I/O lines. An example is shown in the Applications section.

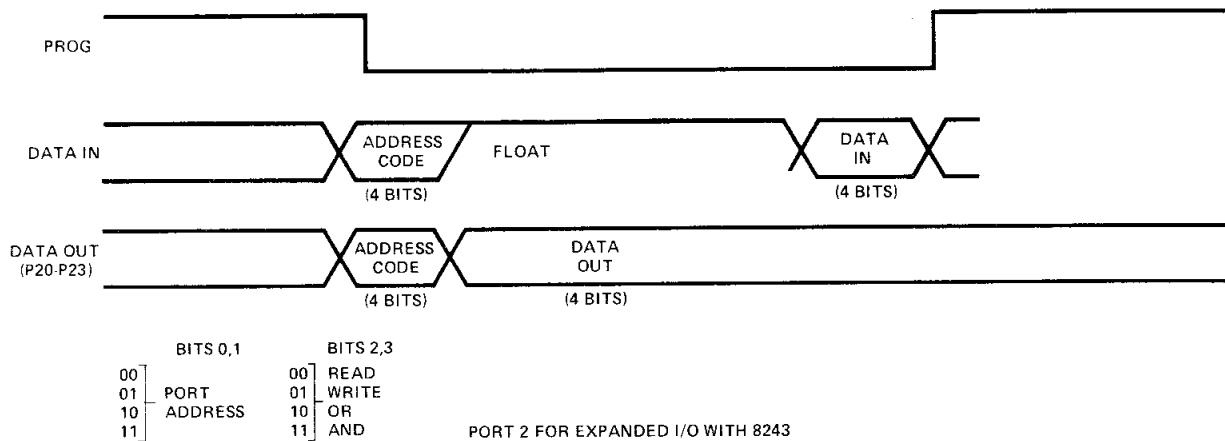


**FIGURE 4. TEST 1 PIN**



**FIGURE 5. I/O EXPANDER INTERFACE**

# SINGLE COMPONENT SYSTEM



**FIGURE 6. EXPANDED I/O TIMING DIAGRAM**

## 2.10 CPU

The 8021 CPU has arithmetic and logical capability. A wide variety of arithmetic and logic instructions may be exercised, which affect the contents of the accumulator, and/or direct or indirect scratchpad locations. Provisions have been made for simplified BCD arithmetic capability using the DAA, SWAP A, and XCHD instructions. In addition, MOVP A,@A allows table lookup for display formatting and constants. The conditional branch logic within the processor enables several conditions internal and external to the processor to be tested by the users program. Use the conditional jump instructions with the tests listed below to effect a change in the program execution sequence.

Test	Jump Condition	Jump Instructions
Accumulator	A=0 A≠0	JZ JNZ
Carry Flag	0 1	JNC, JC
Timer Overflow Flag	— 1	JTF
Test Input-T1	0 1	JNT1, JT1

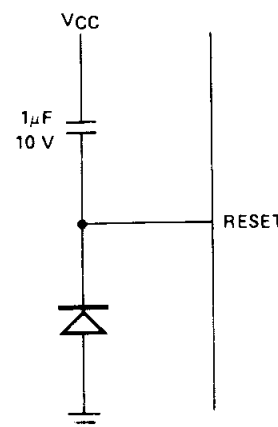
## 2.11 Reset

A positive-going signal to the RESET input resets the necessary miscellaneous flip-flops and sets the program counter and stack pointer to zero.

The 8021 may see poorly regulated and noisy power supplies. A useful feature is to sense when the power supply dips and do a RESET to prevent continued operation with incorrect data. This feature may be implemented on the 8021 by connecting a diode between the RESET node and ground. See Figure 7.

A RESET will then be forced if the supply drops approximately 1.5 volts and rapidly recovers. One instruction cycle will RESET the 8021 to the initialized state.

By removing the diode and using only the capacitor, voltage drops in Vcc will not cause a RESET.

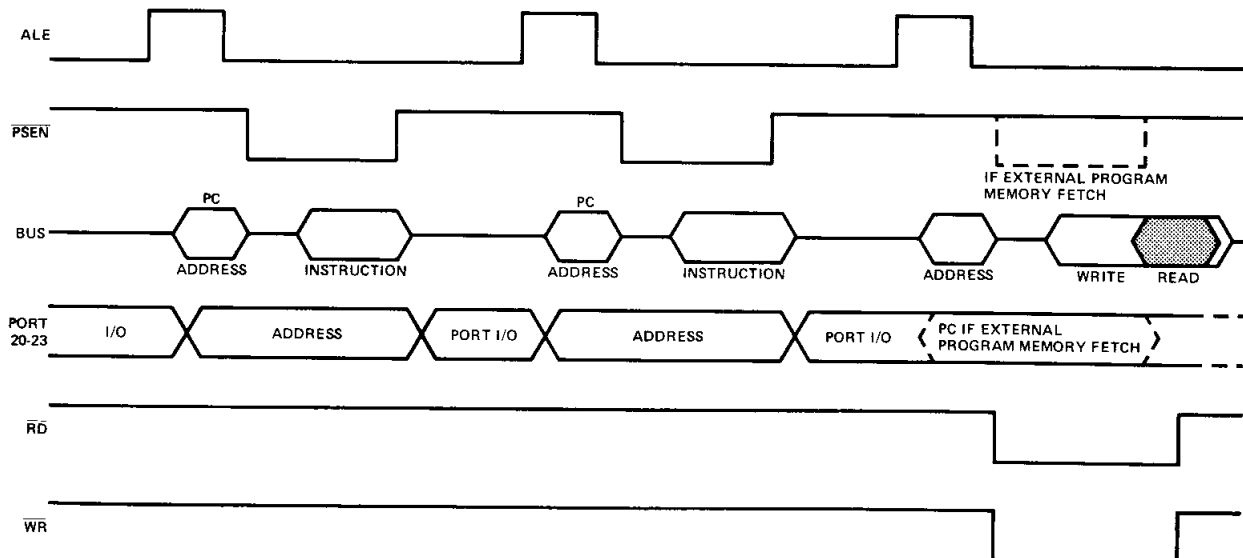


**FIGURE 7. POWER ON RESET**

# THE EXPANDED MCS-48™ SYSTEM

3.0 Summary .....	3-1
3.1 Expansion of Program Memory .....	3-1
3.2 Expansion of Data Memory .....	3-4
3.3 Expansion of Input/Output .....	3-5
3.4 Multi-Chip MCS-48 Systems .....	3-9
3.5 Memory Bank Switching .....	3-10
3.6 Control Signal Summary .....	3-11
3.7 Port Characteristics .....	3-11

## MCS-48™ CYCLE TIMING FOR EXTERNAL MEMORY



# THE EXPANDED MCS-48™ SYSTEM

## 3.0 Summary

If the capabilities resident on the single-chip 8048/8049, 8748, or 8035/8039 are not sufficient for your system requirements, special on-board circuitry allows the addition of a wide variety external memory, I/O, or special peripherals you may require. The processors can be directly and simply expanded in the following areas:

- Program Memory to 4K words
- Data Memory to 320 words (384 words with 8049)
- I/O by unlimited amount
- Special Functions using 8080/8085 peripherals

By using bank switching techniques maximum capability is essentially unlimited. Bank switching is discussed later in the chapter. Expansion is accomplished in two ways:

1. Expander I/O—A special I/O Expander circuit the 8243 provides for the addition of four 4-bit Input/Output ports with the sacrifice of only the lower half (4 bits) of port 2 for inter-device communication. Multiple 8243's may be added to this 4-bit bus by generating the required "chip select" lines.
2. Standard 8085 Bus—One port of the 8048 is like the 8 bit bidirectional data bus of the 8085 microcomputer system allowing interface to the numerous standard memories and peripherals of the MCS-80/85 microcomputer family.

MCS-48 systems can be configured using either or both of these expansion features to optimize system capabilities to the application. Both expander devices and standard memories and peripherals can be added in virtually any number and combination required.

## 3.1 Expansion of Program Memory

Program Memory is expanded beyond the resident 1K or 2K words by using the 8085 BUS feature of the MCS-48. All program memory fetches from addresses less than 1024 (2048) occur internally with no external signals being generated (except ALE which is always present). At address 1024 the 8048 automatically initiates external program memory fetches.

### 3.1.1 Instruction Fetch Cycle (External)

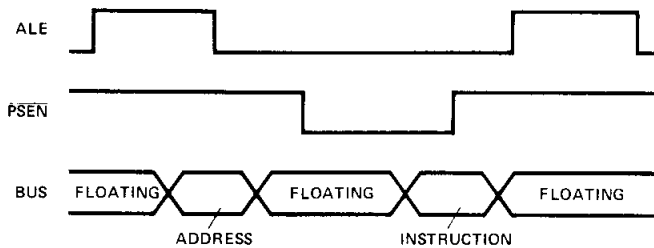
For all instruction fetches from addresses of 1024 (2048) or greater the following will occur:

1. The contents of the 12 bit program counter will be output on BUS and the lower half of port 2.
2. Address Latch Enable (ALE) will indicate the time at which address is valid. The trailing edge of ALE is used to latch the address externally.
3. Program Store Enable ( $\overline{\text{PSEN}}$ ) indicates that an external instruction fetch is in progress and serves to enable the external memory device.
4. BUS reverts to input (floating) mode and the processor accepts its 8 bit contents as an instruction word.

All instruction fetches including internal addresses can be forced to be external by activating the EA pin of the 8048/8049. The 8035/8039 processors without program memory always operate in the external program memory mode (EA=5V).

### 3.1.2 Extended Program Memory Addressing (Beyond 2K)

For programs of 2K words or less, the 8048/8049 addresses program memory in



**INSTRUCTION FETCH FROM EXTERNAL PROGRAM MEMORY**

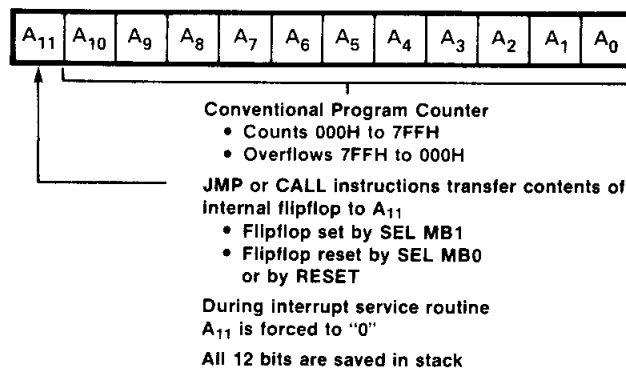
the conventional manner. Addresses beyond 2047 can be reached by executing a program memory bank switch instruction (SEL MB0, SEL MB1) followed by a branch instruction (JMP or CALL). The bank switch feature extends the range of branch instructions beyond their normal 2K range and at the same time prevents the user from inadvertently crossing the 2K boundary.

**Program Memory Bank Switch**

The switching of 2K program memory banks is accomplished by directly setting or resetting the most significant bit of the program counter (bit 11). Bit 11 is not altered by normal incrementing of the program counter but is loaded with the contents of a special flip-flop each time a JMP or CALL instruction is executed. This special flip-flop is set by executing an SEL MB1 instruction and reset by SEL MB0. Therefore, the SEL MB instruction may be executed at any time prior to the actual bank switch which occurs during the next branch instruction encountered. Since all twelve bits of the program counter including bit (11) are stored in the stack when a Call is executed, the user may jump to subroutines across the 2K boundary and the proper bank will be restored upon return. However, the bank switch flipflop will not be altered on return.

**Interrupt Routines**

Interrupts always vector the program counter to location 3 or 7 in the **first** 2K bank and bit 11 of the program counter is held at "0" during the interrupt service routine. The end of the service routine is signalled by the execution of an RETR instruction. Interrupt service routines should therefore be contained



**PROGRAM COUNTER**

entirely in the lower 2K words of program memory. The execution of a SEL MB0 or SEL MB1 instruction within an interrupt routine is not recommended since it will not alter PC11 while in the routine, but will change the internal flip flop.

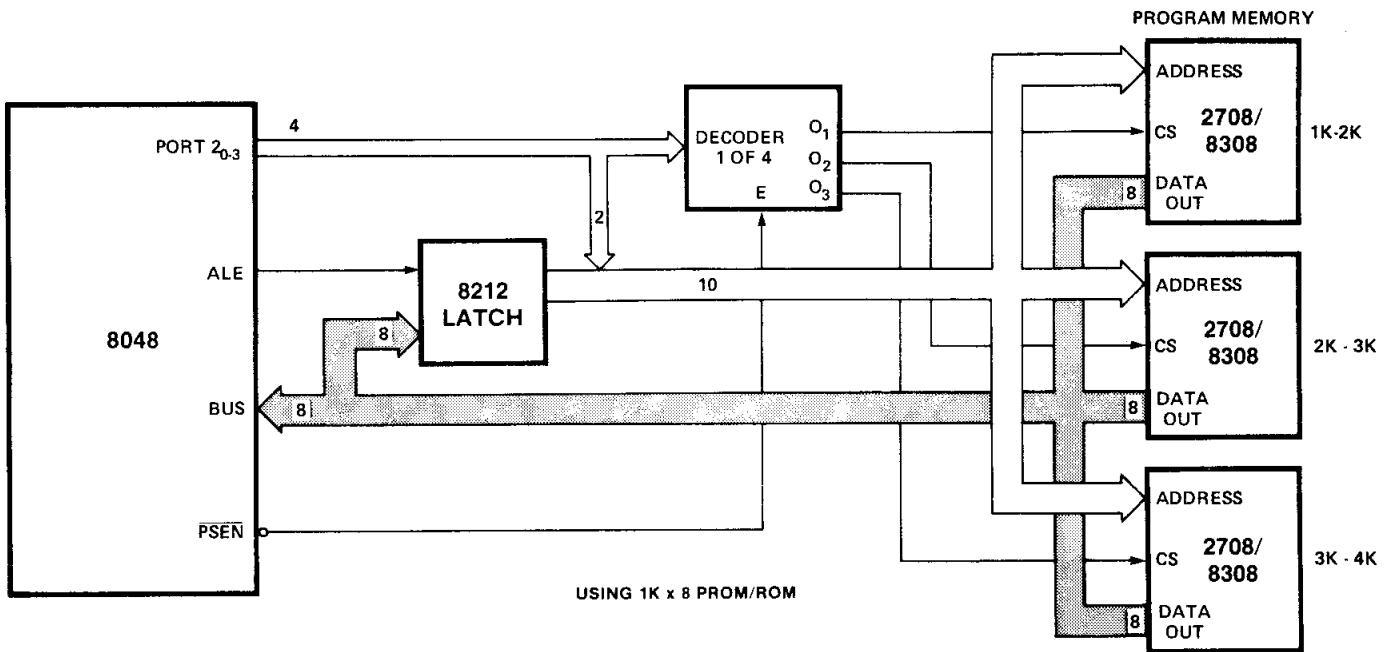
**3.1.3 Restoring I/O Port Information**

Although the lower half of Port 2 is used to output the four most significant bits of address during an external program memory fetch, the I/O information is still outputted during certain portions of each machine cycle. I/O information is always present on Port 2 lower at the rising edge of ALE and can be sampled or latched at this time.

**3.1.4 Expansion Examples**

The accompanying figure shows the addition of three 2708 1K X 8 EPROMs or three 8308 pin-compatible ROM replacements for a total of 4K words of program memory. The BUS port of the 8048 is connected directly to the data output lines of the memories. The lower 8 bits of address are latched in an 8212 8-bit latch using ALE as the strobe. The lower half of Port 2 provides the upper 4 bits of address and since these address bits are stable for the duration of the program memory fetch, they do not have to be latched. Two of the upper address bits are connected directly to the address inputs of the memories while the two most significant bits are decoded to provide the three chip selects needed. The PSEN output of the 8048/8748 is used to enable the chip select lines and therefore the memories.

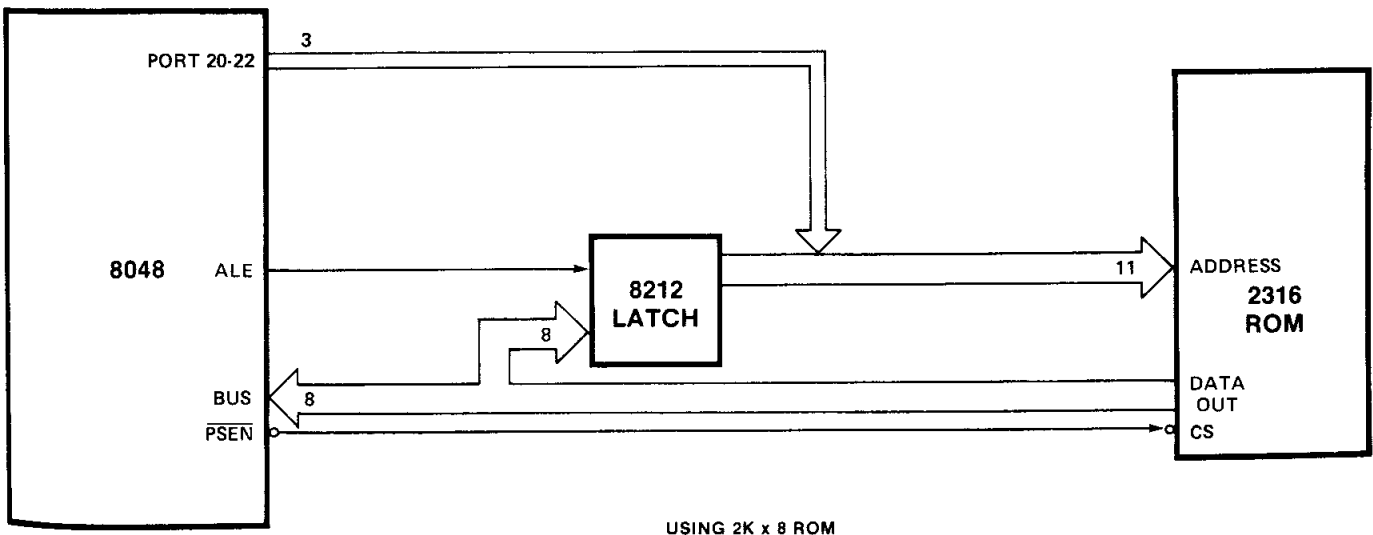
## EXPANDED MCS-48 SYSTEM



### EXPANDING MCS-48™ PROGRAM MEMORY USING STANDARD MEMORY PRODUCTS

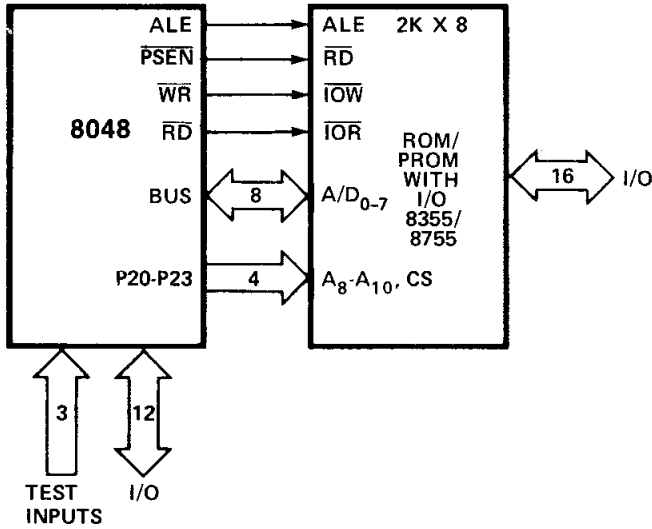
Also shown is the addition of 2K words of program memory using an 8316A 2K x 8 ROM to give a total of 3K words of program memory. In this case no chip select decoding is required and  $\overline{PSEN}$  enables the memory directly through the chip select input. If the system requires only 2K of program the same configuration can be used with an 8035 substituted for the 8048. The 8049 would provide 4K with the same configuration.

The next figure shows how the new 8755/8355 EPROM/ROM with I/O interfaces directly to the 8048 without the need for an address latch. The 8755/8355 contains an internal 8-bit address latch eliminating the need for an 8212 latch. In addition to a 2K X 8 program memory the 8755/8355 also contains 16 I/O lines addressable as two 8-bit ports. These ports are addressed as external RAM; there-



### EXPANDING MCS-48™ PROGRAM MEMORY USING STANDARD MEMORY PRODUCTS

fore, the  $\overline{RD}$  and  $\overline{WR}$  outputs of the 8048 are required. See the following section on data memory expansion for more detail. The subsequent section on I/O expansion explains the operation of the 16 I/O lines.



**EXTERNAL PROGRAM MEMORY INTERFACE**

**3.2 Expansion of Data Memory**

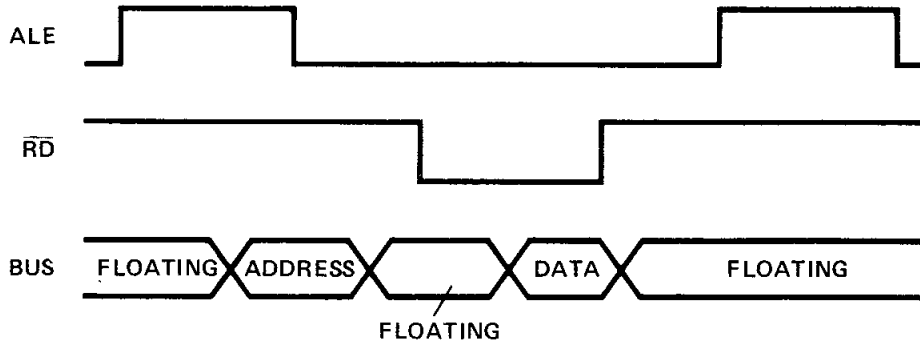
Data Memory is expanded beyond the resident 64 words by using the 8085 type bus feature of the MCS-48.

**3.2.1 Read/Write Cycle**

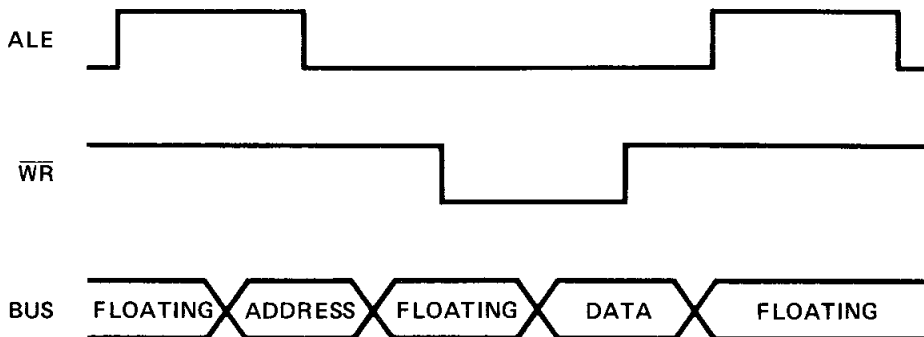
All address and data is transferred over the 8 lines of BUS. A read or write cycle occurs as follows:

1. The contents of register R0 or R1 is outputted on BUS.
2. Address Latch Enable (ALE) indicates address is valid. The trailing edge of ALE is used to latch the address externally.
3. A read ( $\overline{RD}$ ) or write ( $\overline{WR}$ ) pulse on the corresponding output pins of the 8048 indicates the type of data memory access in progress. Output data is valid at the trailing edge of  $\overline{WR}$  and input data must be valid at the trailing edge of  $\overline{RD}$ .
4. Data (8-bits) is transferred in or out over BUS.

**READ FROM EXTERNAL DATA MEMORY**



**WRITE TO EXTERNAL DATA MEMORY**



### 3.2.2 Addressing External Data Memory

External Data Memory is accessed with its own two-cycle move instructions MOVX A, @R and MOVX @R, A which transfer 8 bits of data between the accumulator and the external memory location addressed by the contents of one of the RAM Pointer Registers R0 or R1. This allows 256 locations to be addressed in addition to the resident locations. Additional pages may be added by "bank switching" with extra output lines of the 8048.

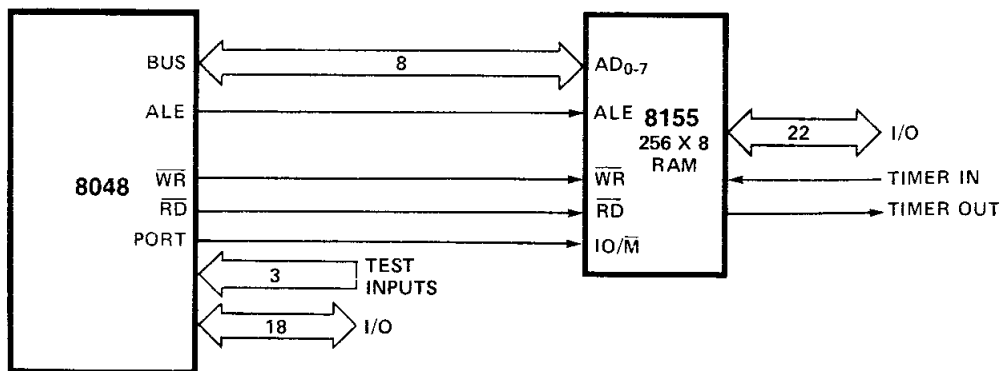
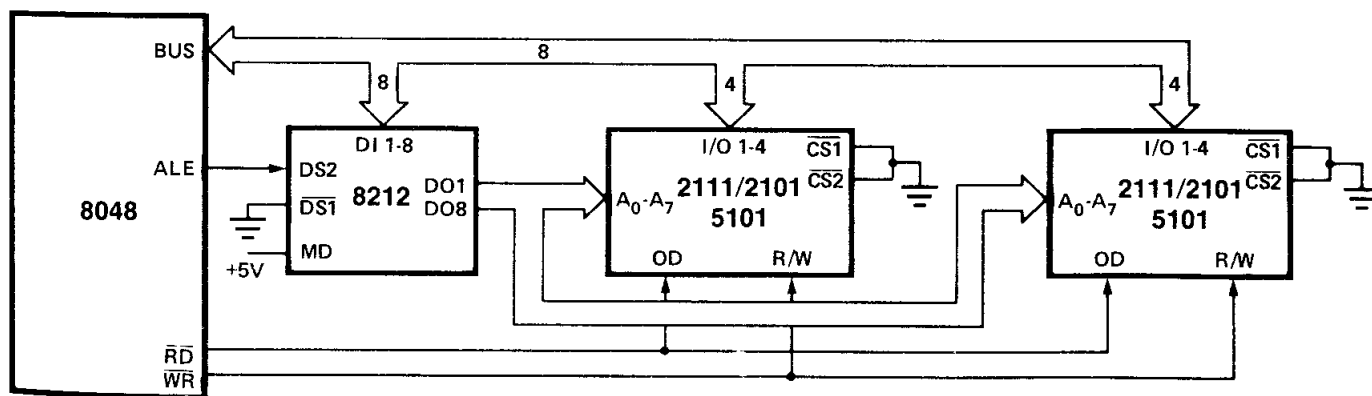
### 3.2.3 Examples of Data Memory Expansion

The accompanying figure shows how the 8048 can be expanded using standard 256 X 4 static RAMs such as the 2101-2 or its low power CMOS equivalent, the 5101. An 8212 serves as an address latch while each 4-bit half of BUS is connected directly to a bidirec-

tional 4-bit data bus of the memories. The  $\overline{WR}$  output of the processor controls the Read/Write input of the memories while the data bus output drivers of the memories are controlled by  $\overline{RD}$ . The chip select lines of the memories are continuously enabled unless additional pages of RAM are required. Also shown is the expansion of data memory using the 8155 memory and I/O expanding device. Since the 8155 has an internal 8-bit address latch it can interface directly to the 8048 without the use of an external 8212 latch. The 8155 provides an additional 256 words of static data memory and also includes 22 I/O lines and a 14 bit timer. See the following section on I/O expansion and the 8155 data sheet for more details on these additional features.

### 3.3 Expansion of Input/Output

There are four possible modes of I/O expansion with the 8048: one using a special low cost expander, the 8243;



8048 INTERFACE TO 256 X 8 STANDARD MEMORIES



## EXPANDED MCS-48 SYSTEM

another using standard MCS-80/85 I/O devices; and a third using the combination memory/I/O expander devices the 8155, 8355, and 8755. It is also possible to expand using standard TTL devices as shown in Chapter 5.

### 3.3.1 I/O Expander Device

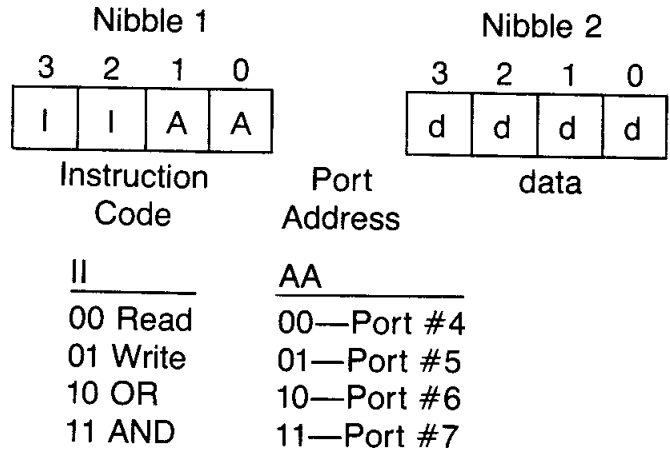
The most efficient means of I/O expansion for small systems is the 8243 I/O Expander Device which requires only 4 port lines (lower half of Port 2) for communication with the 8048. The 8243 contains four 4-bit I/O ports which serve as extension of the on chip I/O and are addressed as ports #4-7. The following operations may be performed on these ports:

1. Transfer Accumulator to Port.
2. Transfer Port to Accumulator.
3. AND Accumulator to Port.
4. OR Accumulator to Port.

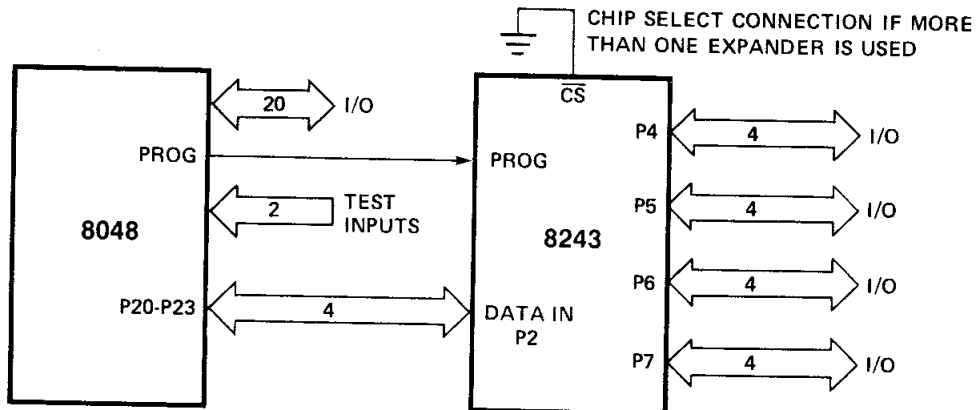
A 4-bit transfer from a port to the lower half of the Accumulator sets the most significant four

bits to zero. All communication between the 8048 and the 8243 occurs over Port 2 lower (P20-P23) with timing provided by an output pulse on the PROG pin of the processor. Each transfer consists of two 4-bit nibbles:

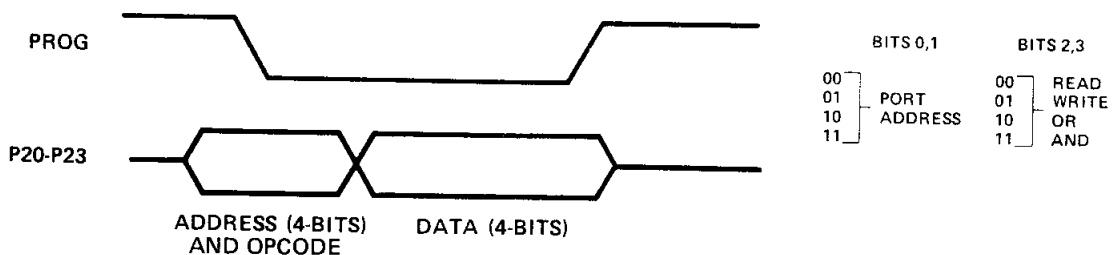
The first containing the "op code" and port address and the second containing the actual 4 bits of data.



### EXPANDER INTERFACE



### OUTPUT EXPANDER TIMING



A high to low transition of the PROG line indicates that address is present while a low to high transition indicates the presence of data. Additional 8243's may be added to the four bit bus and chip selected using additional output lines from the 8048/8748.

**I/O Port Characteristics**

Each of the four 4-bit ports of the 8243 can serve as either input or output and can provide high drive capability in both the high and low state.

**3.3.2 I/O Expansion with Standard Peripherals**

Standard MCS-80/85 type I/O devices may be added to the MCS-48 using the same bus and timing used for Data Memory expansion. I/O devices reside on the Data Memory bus and in the data memory address space and are accessed with the same MOVX instructions. See the previous section on data memory expansion for a description of timing. The following are a few of the Standard MCS-80 devices which are very useful in MCS-48 systems.

- 8214 Priority Interrupt Encoder
- 8251 Serial Communications Interface
- 8255 General Purpose Programmable I/O
- 8279 Keyboard/Display Interface
- 8253 Interval Timer

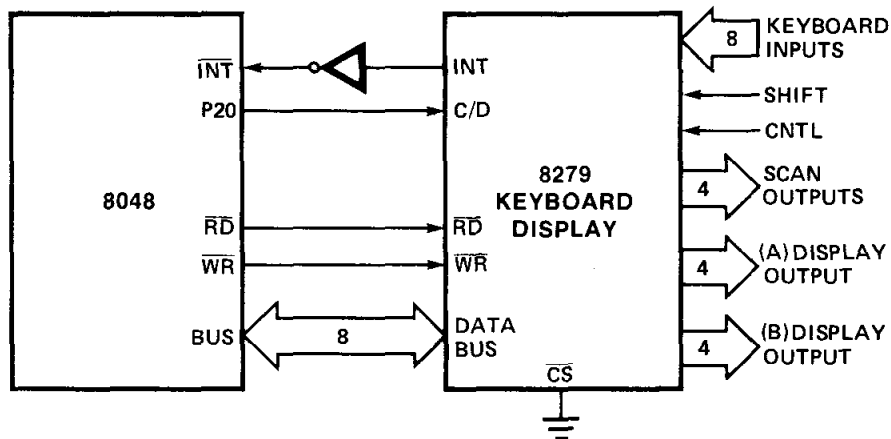
See Chapter 7 for detailed data sheets on these and other components.

**3.3.3 Combination Memory and I/O Expanders**

As mentioned in the sections on program and data memory expansion the 8355/8755 and 8155 expanders also contain I/O capability.

**8355/8755:** These two parts are ROM and EPROM equivalents and therefore contain the same I/O structure. I/O consists of two 8-bit ports which normally reside in the external data memory address space and are accessed with MOVX instructions. Associated with each port is an 8-bit Data Direction Register which defines each bit in the port as either an input or an output. The data direction registers are directly addressable thereby allowing the user to define under software control each individual bit of the ports as either input or output. All outputs are statically latched and double buffered. Inputs are not latched.

**8155/8156:** I/O on the 8155/8156 is configured as two 8-bit programmable I/O ports and one 6-bit programmable port. These three registers and a Control/Status register are accessible as external data memory with the MOVX instructions. The contents of the control register determines the mode of the three ports. The ports can be programmed as input or output with or without associated handshake communication lines. In the handshake mode, lines of the six-bit port become input and output strobes for the two 8-bit ports. See the



**KEYBOARD/DISPLAY INTERFACE**

## EXPANDED MCS-48 SYSTEM

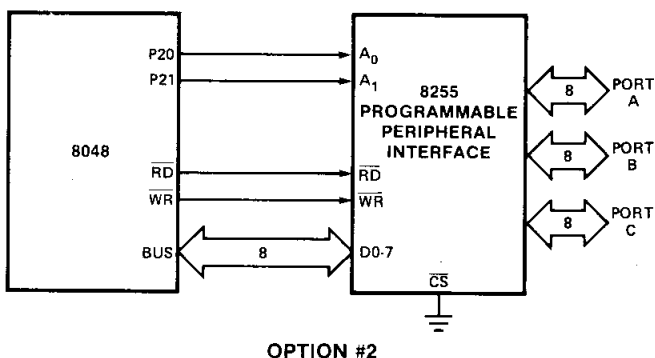
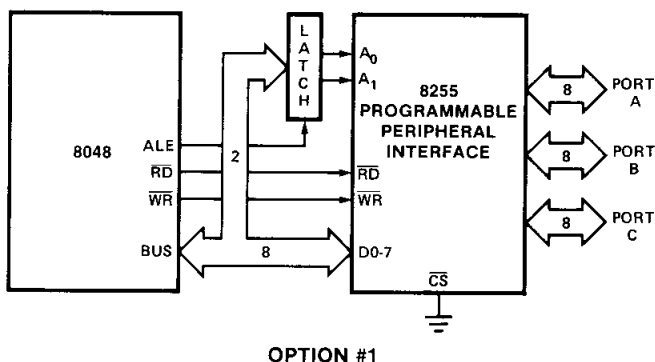
data sheet in the Chapter 6 for details. Also included in the 8155 is a 14-bit programmable timer. The clock input to the timer and the timer overflow output are available on external pins. The timer can be programmed to stop on terminal count or to continuously re-load itself. A square wave or pulse output on terminal count can also be specified.

### I/O Expansion Examples (See Also Chapter 5)

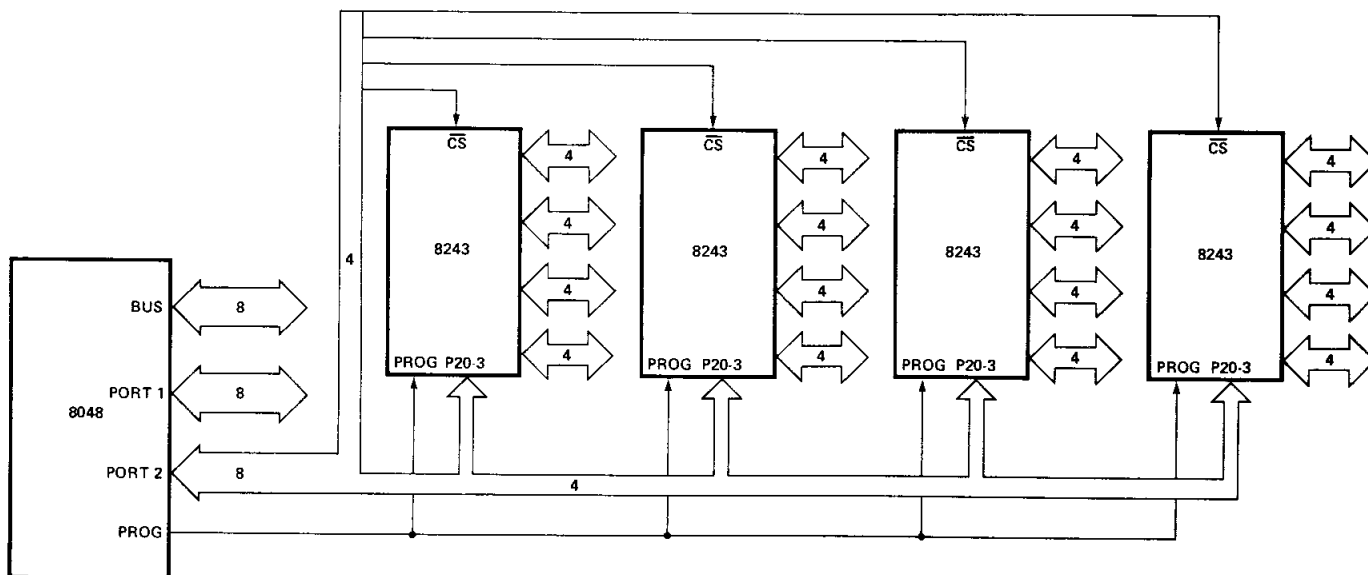
The accompanying figure shows the expansion of I/O using multiple 8243's. The only difference from a single 8243 system is the addition of chip selects provided by additional 8048 output lines. Two output lines and two inverters could also be used to address the four chips. Large numbers of 8243's would require a chip select decoder chip such as the 8205 to save I/O pins.

Also shown is the 8048 interface to a standard MCS-80 peripheral; in this case, the 8255 Programmable Peripheral Interface, a 40 pin part which provides three 8-bit programmable I/O ports. The 8255 bus interface is typical of programmable MCS-80 peripherals with an 8-bit bidirectional data bus, a  $\overline{RD}$  and  $\overline{WR}$  input for Read/Write control, a  $\overline{CS}$

(chip select) input used to enable the Read/Write control logic and the address inputs used to select various internal registers.



### INTERFACE TO MCS-80 PERIPHERALS



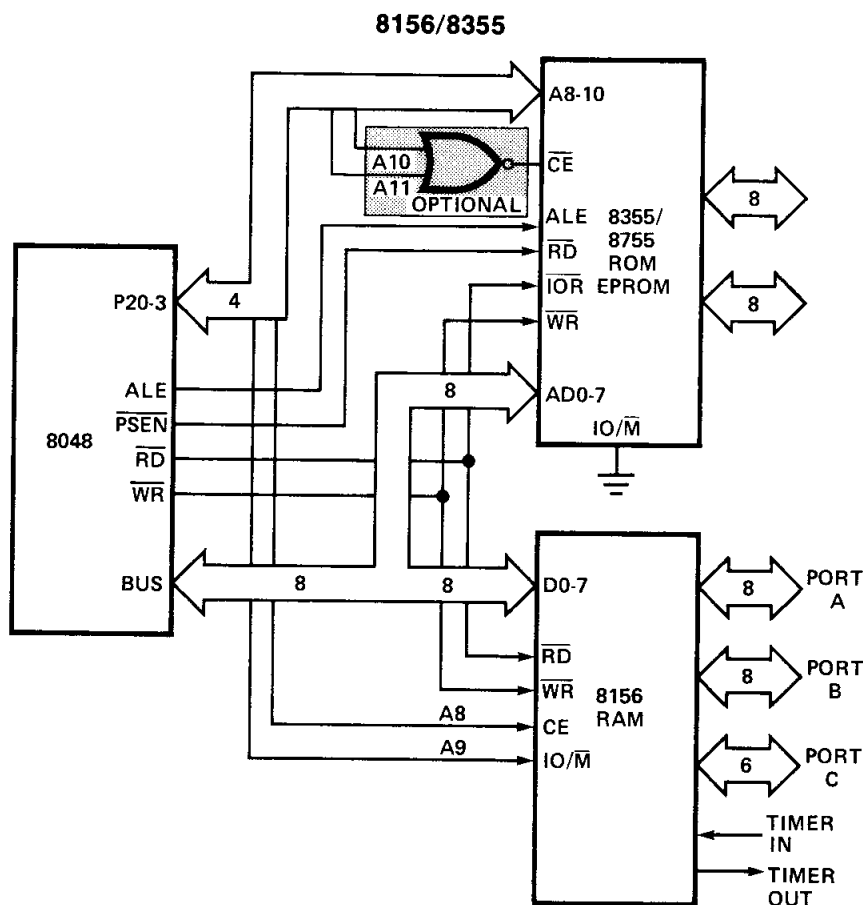
### LOW COST I/O EXPANSION

interconnection to the 8048 is very straightforward with BUS,  $\overline{RD}$ , and  $\overline{WR}$  connecting directly to the corresponding pins on the 8255. The only design consideration is the way in which the internal registers of the 8255 are to be addressed. If the registers are to be addressed as external data memory using the MOVX instructions, the appropriate number of address bits (in this case, 2) must be latched on BUS using ALE as described in the section on external data memories. If only a single device is connected to BUS, the 8255 may be continuously selected by grounding  $\overline{CS}$ . If multiple 8255's are used, additional address bits can be latched and used as chip selects.

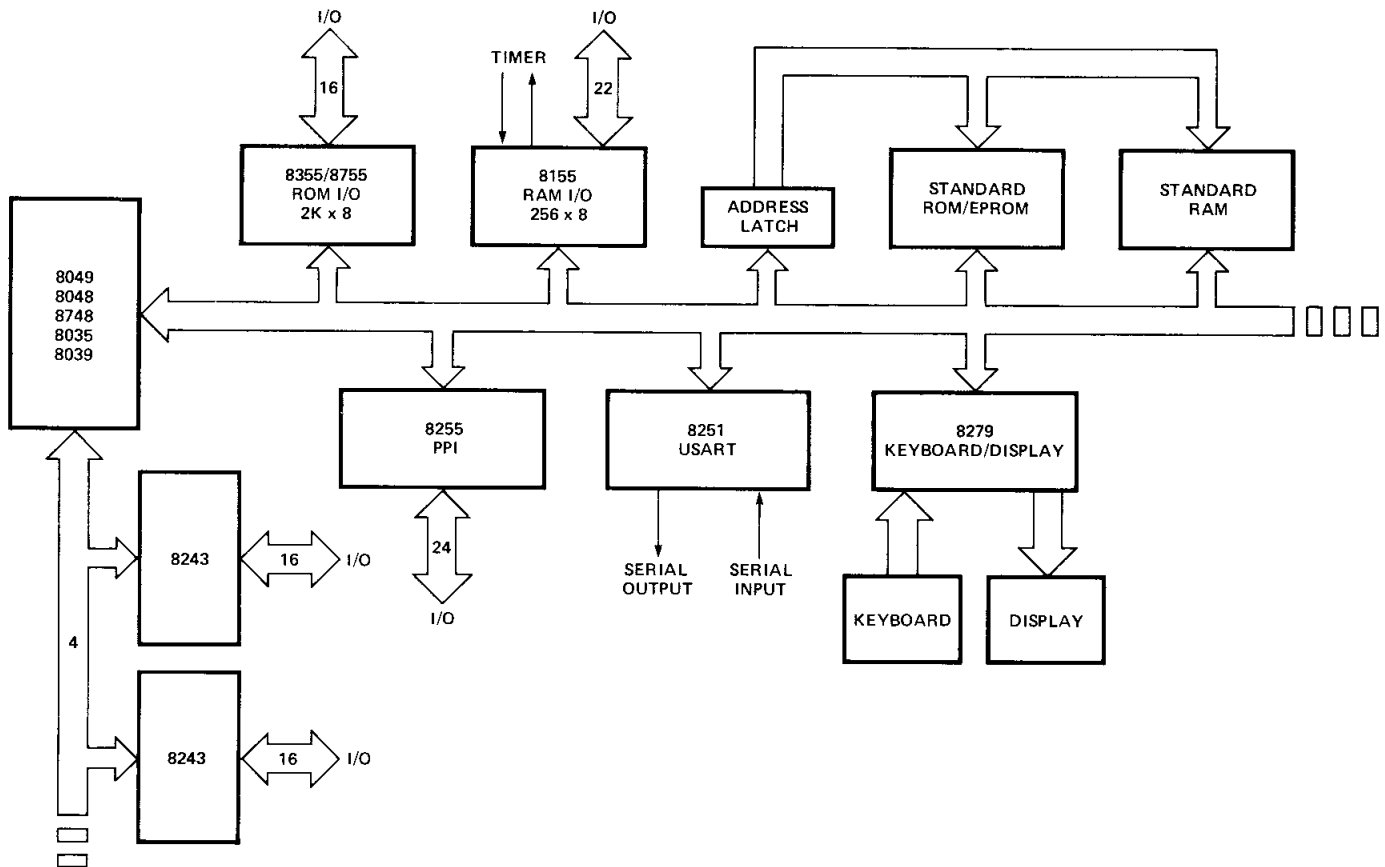
A second addressing method eliminates external latches and chip select decoders by using output port lines as address and chip select lines directly. This method, of course, requires the setting of an output port with address information prior to executing a MOVX instruction.

### 3.4 Multi-Chip MCS-48 Systems

The accompanying figure shows the addition of two memory expanders to the 8048, one 8355/8755 ROM and one 8156 RAM. The main consideration in designing such a system is the addressing of the various memories and I/O ports. Note that in this configuration address lines  $A_{10}$  and  $A_{11}$  have been ORed to chip select the 8355. This ensures that the chip is active for all external program memory fetches in the 1K to 3K range and is disabled for all other addresses. This gating has been added to allow the I/O port of the 8355 to be used. If the chip was left selected all the time there would be conflict between these ports and the RAM and I/O of the 8156. The NOR gate could be eliminated and  $A_{11}$  connected directly to the CE (instead of  $\overline{CE}$ ) input of the 8355; however, this would create a 1K word "hole" in the program memory by causing the 8355 to be active in the 2K to 4K range instead of the normal 1K to 3K range.



## EXPANDED MCS-48 SYSTEM



### MCS-48 EXPANSION CAPABILITY

In this system the various locations are addressed as follows:

**Data RAM**—Addresses 0 to 255 when Port 2 Bit 0 has been previously set = 1 and Bit 1 set = 0

**RAM I/O**—Addresses 0 to 3 when Port 2 Bit 0 = 1 and Bit 1 = 1

**ROM I/O**—Addresses 0 to 3 when Port 2 Bit 2 or Bit 3 = 1

### 3.5 Memory Bank Switching

Certain systems may require more than the 4K words of program memory which are directly addressable by the program counter or more than the 256 data memory and I/O locations directly addressable by the pointer

registers R0 and R1. These systems can be achieved using “bank switching” techniques. Bank switching is merely the selection of various blocks or “banks” of memory using dedicated output port lines from the processor. In the case of the 8048 program memory is selected in blocks of 4K words at a time while data memory and I/O are enabled 256 words at a time.

The most important consideration in implementing two or more banks is the software required to cross the bank boundaries. Each crossing of the boundary requires that the processor first write a control bit to an output port before accessing memory or I/O in the new bank. If program memory is being switched, programs should be organized to

keep boundary crossings to a minimum. Jumping to subroutines across the boundary should be avoided when possible since the programmer must keep track of which bank to return to after completion of the subroutine. If these subroutines are to be nested and accessed from either bank, a software "stack" should be implemented to save the bank switch bit just as if it were another bit of the program counter.

From a hardware standpoint bank switching is very straight-forward and involves only the connection of an I/O line or lines as bank enable signals. These enables are ANDed with normal memory and I/O chip select signals to activate the proper bank.

### 3.6 Control Signal Summary

The following table summarizes the instructions which activate the various control outputs of the MCS-48 processors.

CONTROL SIGNAL	WHEN ACTIVE
$\overline{RD}$	DURING MOVX A,@R OR INS BUS
$\overline{WR}$	DURING MOVX @R,A OR OUTL BUS
ALE	EVERY MACHINE CYCLE
$\overline{PSEN}$	DURING FETCH OF EXTERNAL PROGRAM MEMORY (INSTRUCTION OR IMMEDIATE DATA)
PROG	DURING MOVD A,P ANLD P,A MOVD P,A ORLD P,A

During all other instructions these outputs are driven to the inactive state.

### 3.7 Port Characteristics

#### BUS Port Operations

The BUS port can operate in three different modes: as a latched I/O port, as a bi-

directional bus port, or as a program memory address output when external memory is used. The BUS port lines are either active high, active low, or high impedance (floating). The latched mode (INS, OUTL) is intended for use in the single chip configuration where BUS is not being used as an expander port. OUTL and MOVX instructions can be mixed if necessary. However, a previously latched output will be destroyed by executing a MOVX instruction and BUS will be left in the high impedance state. OUTL should never be used in a system with external program memory, since latching BUS can cause the next instruction, if external, to be fetched improperly.

#### Port 2 Operations

The lower half of Port 2 can be used in three different ways: as a quasi, bi-directional static port, as an 8243 expander port, and to address external program memory. In all cases outputs are driven low by an active device and driven high momentarily by an active device and held high by a 50K $\Omega$  resistor to +5V.

The port may contain latched I/O data prior to its use in another mode without affecting operation of either. If lower Port 2 (P20-3) is used to output address for an external program memory fetch the I/O information previously latched will be automatically removed temporarily while address is present then restored when the fetch is complete. However, if lower Port 2 is used to communicate with an 8243, previously latched I/O information will be removed and not restored. After an input from the 8243 P20-3 will be left in the input mode (floating). After an output to the 8243 P20-3 will contain the value written, ANDed, or ORed to the 8243 port.

# INSTRUCTION SET

4.0 Introduction .....	4-1
4.1 Instruction Set Description .....	4-4

# INSTRUCTION SET

## 4.0 INTRODUCTION

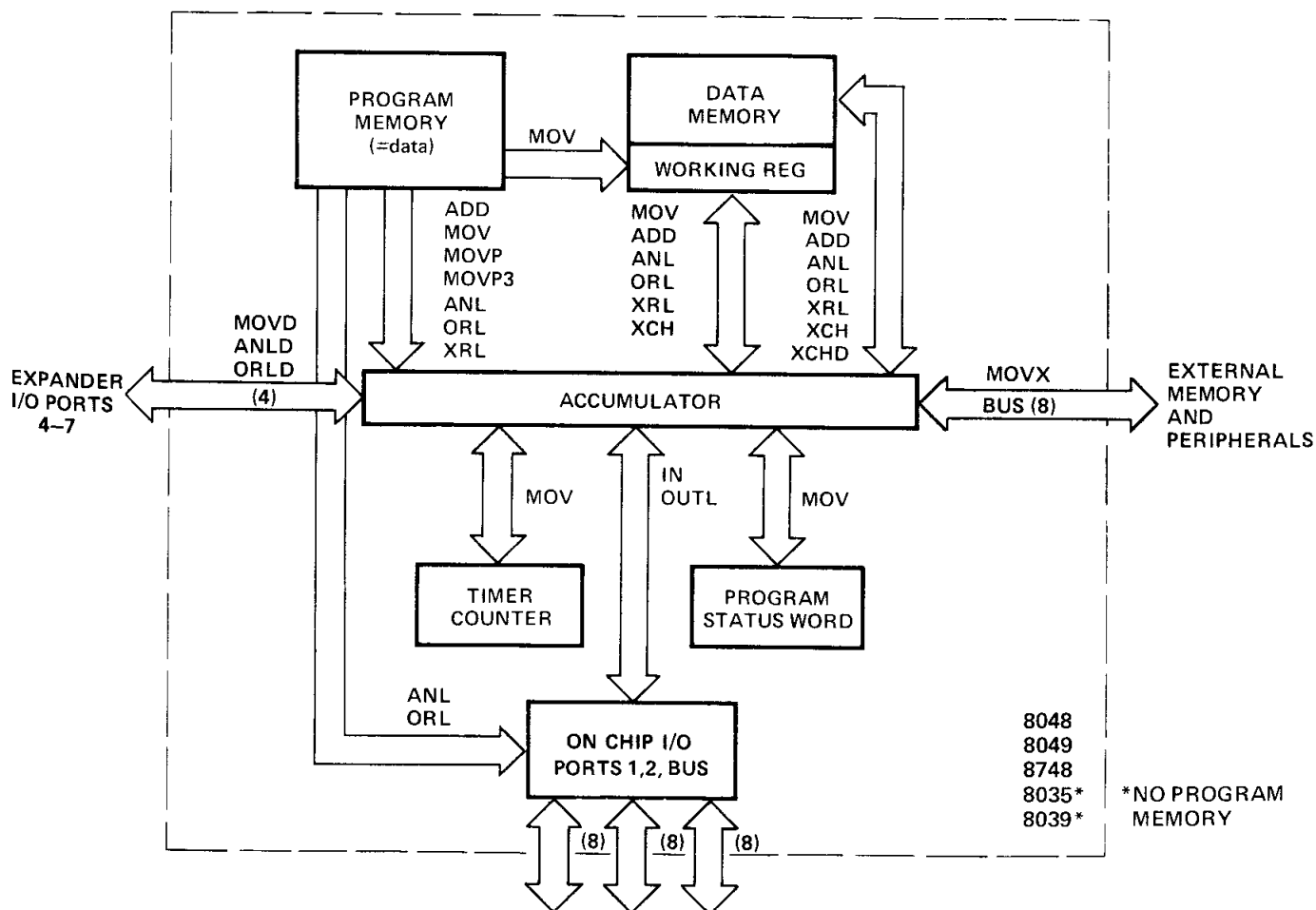
The MCS-48 microcomputers have been designed to efficiently handle arithmetic operations in both binary and BCD as well as to efficiently handle the single bit operations required in control applications. Special instructions have also been included to simplify loop counters, table lookup routines, and N-way branch routines.

The MCS-48 instruction set is extensive for a machine of its size and has been tailored to be straightforward and very efficient in its use of program memory. All instructions are either one or two bytes in length and over 70% are only one byte long. Also, all instructions execute in either one or two cycles (2.5 $\mu$ sec or 5.0 $\mu$ sec when using a 6 MHz XTAL) and over 50% of all instructions execute in a single cycle. Double cycle instructions include all immediate instructions, and all I/O instructions.

The MCS-48 microcomputers have been designed to efficiently handle arithmetic operations in both binary and BCD as well as to efficiently handle the single bit operations required in control applications. Special instructions have also been included to simplify loop counters, table lookup routines, and N-way branch routines.

### Data Transfers

As can be seen in the accompanying diagram, the 8-bit accumulator is the central



## DATA TRANSFER INSTRUCTIONS



point for all data transfers within the 8048. Data can be transferred between the 8 registers of each working register bank and the accumulator directly, i.e. the source or destination register is specified by the instruction. The remaining locations of the internal RAM array are referred to as Data Memory and are addressed indirectly via an address stored in either R0 or R1 of the active working register bank. R0 and R1 are also used to indirectly address external data memory when it is present. Transfers to and from internal RAM require one cycle while transfers to external RAM require two. Constants stored in Program Memory can be loaded directly to the accumulator and to the 8 working registers. Data can also be transferred directly between the accumulator and the on-board timer/counter or the accumulator and the Program Status word (PSW). Writing to the PSW alters machine status accordingly and provides a means of restoring status after an interrupt or of altering the stack pointer if necessary.

### Accumulator Operations

Immediate data, data memory, or the working registers can be added with or without carry to the accumulator. These sources can also be ANDed, ORed, or Exclusive ORed to the accumulator. Data may be moved to or from the accumulator and working registers or data memory. The two values can also be exchanged in a single operation.

In addition, the lower 4 bits of the accumulator can be exchanged with the lower 4-bits of any of the internal RAM locations. This instruction, along with an instruction which swaps the upper and lower 4-bit halves of the accumulator, provides for easy handling of 4-bit quantities, including BCD numbers. To facilitate BCD arithmetic, a Decimal Adjust instruction is included. This instruction is used to correct the result of the binary addition of two two-digit BCD numbers. Performing a decimal adjust on the result in the accumulator produces the required BCD result.

Finally, the accumulator can be: incremented, decremented, cleared, or complemented and can be rotated left or right 1-bit at a time with or without carry.

Although there is no subtract instruction in the 8048, this operation can be easily implemented with three single-byte single-cycle instructions.

A value may be subtracted from the accumulator with the result in the accumulator by:

- Complementing the accumulator
- Adding the value to the accumulator
- Complementing the accumulator.

### Register Operations

The working registers can be accessed via the accumulator as explained above, or can be loaded immediate with constraints from program memory. In addition, they can be incremented or decremented or used as loop counters using the decrement and skip, if not zero instruction, as explained under branch instructions.

All Data Memory including working registers can be accessed with indirect instructions via R0 and R1 and can be incremented.

### Flags

There are four user accessible flags in the 8048: Carry, Auxillary Carry, F0, and F1. Carry indicates overflow of the accumulator, and Auxillary Carry is used to indicate overflow between BCD digits and is used during decimal adjust operation. Both Carry and Auxillary Carry are accessible as part of the program status word and are stored on the stack during subroutines. F0 and F1 are undedicated general purpose flags to be used as the programmer desires. Both flags can be cleared or complemented and tested by conditional jump instructions. F0 is also accessible via the Program Status word and is stored on the stack with the carry flags.

### Branch Instructions

The unconditional jump instruction is two bytes and allows jumps anywhere in the first

2K words of program memory. Jumps to the second 2K of memory (4K words are directly addressible) are made by first executing a select memory bank instruction then executing the jump instruction. The 2K boundary can only be crossed via a jump or subroutine call instruction i.e. the bank switch does not occur until a jump is executed. Once a memory bank has been selected all subsequent jumps will be to the selected bank until another select memory bank instruction is executed. A subroutine in the opposite bank can be accessed by a select memory bank instruction followed by a call instruction. Upon completion of the subroutine execution will automatically return to the original bank; however, unless the original bank is reselected, the next jump instruction encountered will again transfer execution to the opposite bank.

Conditional jumps can test the following inputs and machine status:

- T0 Input pin
- T1 Input pin
- $\overline{\text{INT}}$  Input pin
- Accumulator Zero
- Any bit of Accumulator
- Carry Flag
- F0 Flag
- F1 Flag

Conditional jumps allow a branch to any address within the current page (256 words) of execution. The conditions tested are the instantaneous values at the time the conditional jump is executed. For instance, the jump on accumulator zero instruction tests the accumulator itself not an intermediate zero flag.

The decrement register and skip if not zero instruction combines a decrement and a branch instruction to create an instruction very useful in implementing a loop counter. This instruction can designate any one of the 8 working registers as a counter and can effect a branch to any address within the current page of execution.

A single byte indirect jump instruction allows the program to be vectored to any one of

several different locations based on the contents of the accumulator. The contents of the accumulator points to a location in program memory which contains the jump address. The 8-bit jump address refers to the current page of execution. This instruction could be used, for instance, to vector to any one of several routines based on an ASCII character which has been loaded in the accumulator. In this way ASCII key inputs can be used to initiate various routines.

### Subroutines

Subroutines are entered by executing a call instruction. Calls can be made like unconditional jumps to any address in a 2K word bank and jumps across the 2K boundary are executed in the same manner. Two separate return instructions determine whether or not status (upper 4-bits of PSW) is restored upon return from the subroutine.

The return and restore status instruction also signals the end of an interrupt service routine if one has been in progress.

### Timer Instructions

The 8-bit on board timer/counter can be loaded or read via the accumulator while the counter is stopped or while counting. The counter can be started as a timer with an internal clock source or as an event counter or timer with an external clock applied to the T1 input pin. The instruction executed determines which clock source is used. A single instruction stops the counter whether it is operating with an internal or an external clock source. In addition, two instructions allow the timer interrupt to be enabled or disabled.

### Control Instructions

Two instructions allow the external interrupt source to be enabled or disabled. Interrupts are initially disabled and are automatically disabled while an interrupt service routine is in progress and re-enabled afterward.

There are four memory bank select instructions, two to designate the active working register bank and two to control program

memory banks. The operation of the program memory bank switch is explained in section 3.1.2. The working register bank switch instructions allow the programmer to immediately substitute a second 8 register working register bank for the one in use. This effectively provides 16 working registers or it can be used as a means of quickly saving the contents of the registers in response to an interrupt. The user has the option to switch or not to switch banks on interrupt. However, if the banks are switched, the original bank will be automatically restored upon execution of a return and restore status instruction at the end of the interrupt service routine.

A special instruction enables an internal clock, which is the XTAL frequency divided by three, to be output on pin T0. This clock can be used as a general purpose clock in the users system. This instruction should be used only to initialize the system since the clock output can be disabled only by application of system reset.

## Input/Output Instructions

Ports 1 and 2 are 8-bit static I/O ports which can be loaded to and from the accumulator. Outputs are statically latched but inputs are not latched and must be read while inputs are present. In addition, immediate data from program memory can be ANDed or ORed directly to Port 1 and Port 2 with the result remaining on the port. This allows "masks" stored in program memory to selectively set or reset individual bits of the I/O ports. Ports 1 and 2 are configured to allow input on a given pin by first writing a "1" out to the pin.

An 8-bit port called BUS can also be accessed via the accumulator and can have statically latched outputs as well. It too can have immediate data ANDed or ORed directly to its outputs, however, unlike ports 1 and 2, all eight lines of BUS must be treated as either input or output at any one time. In addition to being a static port, BUS can be used as a true synchronous bi-directional port using the Move External instructions used to access external data memory. When these instructions are executed a cor-

responding READ or WRITE pulse is generated and data is valid only at that time. When data is not being transferred BUS is in a high impedance state.

The basic three on board I/O ports can be expanded via a 4-bit expander bus using half of port 2. I/O expander devices on this bus consist of four 4-bit ports which are addressed as ports 4 through 7. These ports have their own AND and OR instructions like the on board ports as well as move instructions to transfer data in or out. The expander AND and OR instructions, however, combine the contents of accumulator with the selected port rather than immediate data as is done with the on board ports.

I/O devices can also be added externally using the BUS port as the expansion bus. In this case the I/O ports become "memory mapped", i.e. they are addressed in the same way as external data memory and exist in the external data memory address space addressed by pointer register R0 or R1.

## 4.1 Instruction Set Description

The following pages describe the MCS-48 instruction set in detail. The instruction set is first summarized with instructions grouped functionally. This summary page is followed by a detailed description listed alphabetically by mnemonic opcode.

The alphabetical listing includes the following information:

- Mnemonic
- Machine Code
- Verbal Description
- Symbolic Description
- Assembly Language Example

The machine code is represented with the most significant bit (7) to the left and two byte instructions are represented with the first byte on the left. The assembly language examples are formulated as follows:

Arbitrary  
Label: Mnemonic, Operand; Descriptive Comment  
See section 1.2.2 for a description and example of an assembly language program.

# 8048/8049 INSTRUCTION SET SUMMARY

	Mnemonic	Description	Bytes	Cycle		Mnemonic	Description	Bytes	Cycles	
Accumulator	ADD A, R	Add register to A	1	1	Subroutine	CALL	Jump to subroutine	2	2	
	ADD A, @R	Add data memory to A	1	1		RET	Return	1	2	
	ADD A, #data	Add immediate to A	2	2		RETR	Return and restore status	1	2	
	ADDC A, R	Add register with carry	1	1		Flags	CLR C	Clear Carry	1	1
	ADDC A, @R	Add data memory with carry	1	1			CPL C	Complement Carry	1	1
	ADDC A, #data	Add immediate with carry	2	2	CLR F0		Clear Flag 0	1	1	
	ANL A, R	And register to A	1	1	CPL F0		Complement Flag 0	1	1	
	ANL A, @R	And data memory to A	1	1	CLR F1		Clear Flag 1	1	1	
	ANL A, #data	And immediate to A	2	2	CPL F1	Complement Flag 1	1	1		
	ORL A, R	Or register to A	1	1	Data Moves	MOV A, R	Move register to A	1	1	
	ORL A, @R	Or data memory to A	1	1		MOV A, @R	Move data memory to A	1	1	
	ORL A, #data	Or immediate to A	2	2		MOV A, #data	Move immediate to A	2	2	
	XRL A, R	Exclusive Or register to A	1	1		MOV R, A	Move A to register	1	1	
	XRL A, @R	Exclusive or data memory to A	1	1		MOV @R, A	Move A to data memory	1	1	
	XRL A, #data	Exclusive or immediate to A	2	2		MOV R, #data	Move immediate to register	2	2	
	INC A	Increment A	1	1		MOV @R, #data	Move immediate to data memory	2	2	
	DEC A	Decrement A	1	1		MOV A, PSW	Move PSW to A	1	1	
	CLR A	Clear A	1	1		MOV PSW, A	Move A to PSW	1	1	
	CPL A	Complement A	1	1		XCH A, R	Exchange A and register	1	1	
	DA A	Decimal Adjust A	1	1		XCH A, @R	Exchange A and data memory	1	1	
	SWAP A	Swap nibbles of A	1	1		XCHD A, @R	Exchange nibble of A and register	1	1	
	RL A	Rotate A left	1	1		MOVX A, @R	Move external data memory to A	1	2	
RLC A	Rotate A left through carry	1	1	MOVX @R, A		Move A to external data memory	1	2		
RRC A	Rotate A right	1	1	MOVP A, @A		Move to A from current page	1	2		
RRC A	Rotate A right through carry	1	1	MOVP3 A, @A	Move to A from Page 3	1	2			
Input/Output	IN A, P	Input port to A	1	2	Timer/Counter	MOV A, T	Read Timer/Counter	1	1	
	OUTL P, A	Output A to port	1	2		MOV T, A	Load Timer/Counter	1	1	
	ANL P, #data	And immediate to port	2	2		STRT T	Start Timer	1	1	
	ORL P, #data	Or immediate to port	2	2		STRT CNT	Start Counter	1	1	
	INS A, BUS	Input BUS to A	1	2		STOP TCNT	Stop Timer/Counter	1	1	
	OUTL BUS, A	Output A to BUS	1	2		EN TCNTI	Enable Timer/Counter Interrupt	1	1	
	ANL BUS, #data	And immediate to BUS	2	2		DIS TCNTI	Disable Timer/Counter Interrupt	1	1	
	ORL BUS, #data	Or immediate to BUS	2	2		Control	EN I	Enable external interrupt	1	1
	MOVD A, P	Input Expander port to A	1	2	DIS I		Disable external interrupt	1	1	
	MOVD P, A	Output A to Expander port	1	2	SEL RB0		Select register bank 0	1	1	
ANLD P, A	And A to Expander port	1	2	SEL RB1	Select register bank 1		1	1		
ORLD P, A	Or A to Expander port	1	2	SEL MB0	Select memory bank 0		1	1		
Registers	INC R	Increment register	1	1	SEL MB1	Select memory bank 1	1	1		
	INC @R	Increment data memory	1	1	ENT0 CLK	Enable Clock output on T0	1	1		
	DEC R	Decrement register	1	1	NOP	NOP	No Operation	1	1	
Branch	JMP addr	Jump unconditional	2	2						
	JMPP @A	Jump indirect	1	2						
	DJNZ R, addr	Decrement register and skip	2	2						
	JC addr	Jump on Carry = 1	2	2						
	JNC addr	Jump on Carry = 0	2	2						
	JZ addr	Jump on A Zero	2	2						
	JNZ addr	Jump on A not Zero	2	2						
	JT0 addr	Jump on T0 = 1	2	2						
	JNT0 addr	Jump on T0 = 0	2	2						
	JT1 addr	Jump on T1 = 1	2	2						
	JNT1 addr	Jump on T1 = 0	2	2						
	JF0 addr	Jump on F0 = 1	2	2						
	JF1 addr	Jump on F1 = 1	2	2						
	JTF addr	Jump on timer flag	2	2						
	JNI addr	Jump on $\overline{INT}$ = 0	2	2						
	JBb addr	Jump on Accumulator Bit	2	2						

# 8021 INSTRUCTION SET SUMMARY

	Mnemonic	Description	Bytes	Cycle	
Accumulator	ADD A,R	Add register to A	1	1	
	ADD A,@R	Add data memory to A	1	1	
	ADD A,#data	Add immediate to A	2	2	
	ADDC A,R	Add with carry	1	1	
	ADDC A,@R	Add with carry	1	1	
	ADDC A,#data	Add with carry	2	2	
	ANL A,R	And register to A	1	1	
	ANL A,@R	And data memory to A	1	1	
	ANL A,#data	And immediate to A	2	2	
	ORL A,R	Or register to A	1	1	
	ORL A,@R	Or data memory to A	1	1	
	ORL A,#data	Or immediate to A	2	2	
	XRL A,R	Exclusive Or register to A	1	1	
	XRL A,@R	Exclusive or data memory to A	1	1	
	XRL A,#data	Exclusive or immediate to A	2	2	
	INC A	Increment A	1	1	
	DEC A	Decrement A	1	1	
	CLR A	Clear A	1	1	
	CPL A	Complement A	1	1	
	DA A	Decimal Adjust A	1	1	
	SWAP A	Swap nibbles of A	1	1	
	RL A	Rotate A left	1	1	
	RLC A	Rotate A left through carry	1	1	
	RR A	Rotate A right	1	1	
	RRC A	Rotate A right through carry	1	1	
	Registers	IN A,P	Input port to A	1	2
		OUTL P,A	Output A to port	1	2
		MOVD A,P	Input Expander port to A	1	2
MOVD P,A		Output A to Expander port	1	2	
ANLD P,A		And A to Expander port	1	2	
ORLD P,A		Or A to Expander port	1	2	
Input/Output	INC R	Increment register	1	1	
	INC @R	Increment data memory	1	1	

	Mnemonic	Description	Bytes	Cycle
Branch	JMP addr	Jump unconditional	2	2
	JMPP @A	Jump indirect	1	2
	DJNZ R,addr	Decrement register and Jump on R not zero	2	2
	JC addr	Jump on Carry = 1	2	2
	JNC addr	Jump on Carry = 0	2	2
	JZ addr	Jump on A Zero	2	2
	JNZ addr	Jump on A not Zero	2	2
	JT1 addr	Jump on T1 = 1	2	2
	JNT1 addr	Jump on T1 = 0	2	2
	JTF addr	Jump on timer flag	2	2
Subroutine	CALL	Jump to subroutine	2	2
	RET	Return	1	2
Flags	CLR C	Clear Carry	1	1
	CPL C	Complement Carry	1	1
Data Moves	MOV A,R	Move register to A	1	1
	MOV A,@R	Move data memory to A	1	1
	MOV A,#data	Move immediate to A	2	2
	MOV R,A	Move A to register	1	1
	MOV @R,A	Move A to data memory	1	1
	MOV R,#data	Move immediate to register	2	2
	MOV @R,#data	Move immediate to data memory	2	2
	XCH A,R	Exchange A and register	1	1
	XCH A,@R	Exchange A and data memory	1	2
	XCHD A,@R	Exchange nibble of A and register	1	1
MOV P A,@A	Move to A from current page	1	2	
Timer/Counter	MOV A,T	Read Timer/Counter	1	1
	MOV T,A	Load Timer/Counter	1	1
	STRT T	Start Timer	1	1
	STRT CNT	Start Counter	1	1
	STOP TCNT	Stop Timer/Counter	1	1
	NOP	No Operation	1	1

*Instruction Set* — The following instructions, which are found in the 8748, have been deleted from the 8021 instruction set.

Data Moves	Registers	Branch	Timer	Control	Input/Output
MOV A,PSW	DEC R	JT0 addr	EN TCNTI	EN I	ANL P,#data
MOV PSW,A		JNT0 addr	DIS TCNTI	DIS I	ORL P,#data
MOVX A,@R	Flags	JF0 addr	Subroutine	SEL RB0	INS A,BUS *
MOVX @R,A		JF1 addr		SEL RB1	OUTL BUS,A *
MOV P3 A,@A	CLR F0	JNI addr	RETR	SEL MB0	ANL BUS,#data
	CPL F0	J8b addr		SEL MB1	ORL BUS,#data
	CLR F1			ENT0 CLK	
	CPL F1				

\*These Instructions have been replaced in the 8021 by IN A,PO and OUTL PO,A respectively.

# MCS-48™ INSTRUCTION SET

## SYMBOLS AND ABBREVIATIONS USED

A	Accumulator
AC	Auxillary Carry
addr	12-Bit Program Memory Address
Bb	Bit Designator (b=0-7)
BS	Bank Switch
BUS	BUS Port
C	Carry
CLK	Clock
CNT	Event Counter
D	Mnemonic for 4-Bit Digit (Nibble)
data	8-Bit Number or Expression
DBF	Memory Bank Flip-Flop
F0, F1	Flag 0, Flag 1
I	Interrupt
P	Mnemonic for "in-page" Operation
PC	Program Counter
Pp	Port Designator (p=1, 2 or 4-7)
PSW	Program Status Word
Rr	Register Designator (r=0, 1 or 0-7)
SP	Stack Pointer
T	Timer
TF	Timer Flag
T0, T1	Test 0, Test 1
X	Mnemonic for External RAM
#	Immediate Data Prefix
@	Indirect Address Prefix
\$	Current Value of Program Counter
(X)	Contents of X
((X))	Contents of Location Addressed by X
←	Is Replaced by

**ADD A,R<sub>r</sub> Add Register Contents to Accumulator**

0 1 1 0	1 r r r
---------	---------

The contents of register 'r' are added to the accumulator. Carry is affected.

$$(A) \leftarrow (A) + (Rr) \quad r=0-7$$

**Example:** ADDREG: ADD A,R6 ;ADD REG 6 CONTENTS  
;TO ACC

**ADD A,@R<sub>r</sub> Add Data Memory Contents to Accumulator**

0 1 1 0	0 0 0 r
---------	---------

The contents of the resident data memory location addressed by register 'r' bits 0-5\*are added to the accumulator. Carry is affected.

$$(A) \leftarrow (A) + ((Rr)) \quad r=0-1$$

**Example:** ADDM: MOV R0, #01FH ;MOVE '1F' HEX TO REG 0  
ADD A, @R0 ;ADD VALUE OF LOCATION  
? ;47 TO ACC

**ADD A,#data Add Immediate Data to Accumulator**

0 0 0 0	0 0 1 1	d <sub>7</sub> d <sub>6</sub> d <sub>5</sub> d <sub>4</sub>	d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub>
---------	---------	---	---

This is a 2-cycle instruction. The specified data is added to the accumulator. Carry is affected

$$(A) \leftarrow (A) + \text{data}$$

**Example:** ADDID: ADD A,#ADDER: ;ADD VALUE OF SYMBOL  
;'ADDER' TO ACC

**ADDC A,R<sub>r</sub> Add Carry and Register Contents to Accumulator**

0 1 1 1	1 r r r
---------	---------

The content of the carry bit is added to accumulator location 0 and the carry bit cleared. The contents of register 'r' are then added to the accumulator. Carry is affected.

$$(A) \leftarrow (A) + (Rr) + (C) \quad r=0-7$$

**Example:** ADDRGC: ADDC A,R4 ;ADD CARRY AND REG 4  
;CONTENTS TO ACC

**ADDC A,@R<sub>r</sub> Add Carry and Data Memory Contents to Accumulator**

---

0 1 1 1	0 0 0 r
---------	---------

The content of the carry bit is added to accumulator location 0 and the carry bit cleared. Then the contents of the resident data memory location addressed by register 'r' bits 0-5\*are added to the accumulator. Carry is affected.

$$(A) \leftarrow (A) + ((Rr)) + (C) \quad r=0-1$$

**Example:** ADDMC: MOV R1,#40 ;MOVE '40' DEC TO REG 1  
                   ADDC A,@R1 ;ADD CARRY AND LOCATION 40  
                                   ;CONTENTS TO ACC

**ADDC A,#data Add Carry and Immediate Data to Accumulator**

---

0 0 0 1	0 0 1 1	d <sub>7</sub> d <sub>6</sub> d <sub>5</sub> d <sub>4</sub>	d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub>
---------	---------	---	---

This is a 2-cycle instruction. The content of the carry bit is added to accumulator location 0 and the carry bit cleared. Then the specified data is added to the accumulator. Carry is affected.

$$(A) \leftarrow (A) + \text{data} + (C)$$

**Example:**           ADDC A,#225                   ;ADD CARRY AND '225' DEC  
   ;TO ACC

**ANL A,R<sub>r</sub> Logical AND Accumulator With Register Mask**

---

0 1 0 1	1 r r r
---------	---------

Data in the accumulator is logically ANDed with the mask contained in working register 'r'.

$$(A) \leftarrow (A) \text{ AND } (Rr) \quad r=0-7$$

**Example:** ANDREG: ANL A,R3                   ;'AND' ACC CONTENTS WITH MASK  
   ;IN REG 3

**ANL A,@R<sub>r</sub> Logical AND Accumulator With Memory Mask**

---

0 1 0 1	0 0 0 r
---------	---------

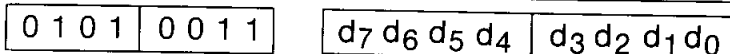
Data in the accumulator is logically ANDed with the mask contained in the data memory location referenced by register 'r', bits 0-5\*.

$$(A) \leftarrow (A) \text{ AND } ((Rr)) \quad r=0-1$$

**Example:** ANDDM: MOV R0,#03FH ;MOVE '3F' HEX TO REG 0  
                   ANL A, @R0 ;'AND' ACC CONTENTS WITH  
                                   ;MASK IN LOCATION 63



**ANL A,#data Logical AND Accumulator With Immediate Mask**

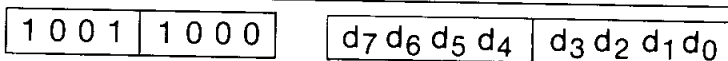


This is a 2-cycle instruction. Data in the accumulator is logically ANDed with an immediately-specified mask.

(A) ← (A) AND data

**Examples:** ANDID: ANL A,#0AFH ;'AND' ACC CONTENTS  
 ;WITH MASK 10101111  
 ANL A,#3+X/Y ;'AND' ACC CONTENTS  
 ;WITH VALUE OF EXP  
 ;'3+X/Y'

**ANL BUS,#data Logical AND BUS With Immediate Mask (Not in 8021)**

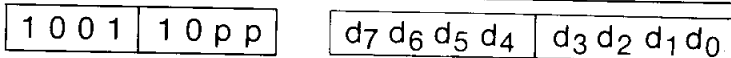


This is a 2-cycle instruction. Data on the BUS port is logically ANDed with an immediately-specified mask. This instruction assumes prior specification of an 'OUTL BUS, A' instruction.

(BUS) ← (BUS) AND data

**Example:** ANDBUS: ANL BUS, #MASK ;'AND' BUS CONTENTS  
 ;WITH MASK EQUAL VALUE  
 ;OF SYMBOL 'MASK'

**ANL Pp,#data Logical AND Port 1-2 With Immediate Mask (Not in 8021)**

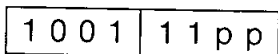


This is a 2-cycle instruction. Data on port 'p' is logically ANDed with an immediately-specified mask.

(Pp) ← (Pp) AND data    p=1-2

**Example:** ANDP2: ANL P2,#0F0H ;'AND' PORT 2 CONTENTS  
 ;WITH MASK 'F0' HEX  
 ;(CLEAR P20-23)

**ANLD Pp,A Logical AND Port 4-7 With Accumulator Mask**



This is a 2-cycle instruction. Data on port 'p' is logically ANDed with the digit mask contained in accumulator bits 0-3.

(Pp) ← (Pp) AND (A0-3)    p=4-7

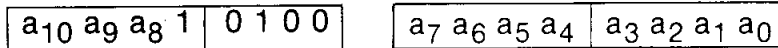
## INSTRUCTION SET

Note: The mapping of port 'p' to opcode bits 0-1 is as follows:

1 0	Port
0 0	4
0 1	5
1 0	6
1 1	7

**Example:** ANDP4: ANLD P4,A ;'AND' PORT 4 CONTENTS  
;WITH ACC BITS 0-3

### CALL address Subroutine Call



This is a 2-cycle instruction. The program counter and PSW bits 4-7 are saved in the stack. The stack pointer (PSW bits 0-2) is updated. Program control is then passed to the location specified by 'address'. PC bit 11 is determined by the most recent SEL MB instruction.

Execution continues at the instruction following the CALL upon return from the subroutine.

((SP)) — (PC), (PSW 4-7)  
 (SP) — (SP)+1  
 (PC<sub>8-10</sub>) — (addr<sub>8-10</sub>)  
 (PC<sub>0-7</sub>) — addr<sub>0-7</sub>  
 (PC<sub>11</sub>) — DBF

**Example:** Add three groups of two numbers. Put subtotals in locations 50, 51 and total in location 52.

```

      MOV R0,#50 ;MOVE '50' DEC TO ADDRESS
                        ;REG 0
BEGADD: MOV A,R1 ;MOVE CONTENTS OF REG 1
                        ;TO ACC
      ADD A,R2 ;ADD REG 2 TO ACC
      CALL SUBTOT;CALL SUBROUTINE 'SUBTOT'
      ADD A,R3 ;ADD REG 3 TO ACC
      ADD A,R4 ;ADD REG 4 TO ACC
      CALL SUBTOT;CALL SUBROUTINE 'SUBTOT'
      ADD A,R5 ;ADD REG 5 TO ACC
      ADD A,R6 ;ADD REG 6 TO ACC
      CALL SUBTOT;CALL SUBROUTINE 'SUBTOT'

SUBTOT: MOV @R0,A ;MOVE CONTENTS OF ACC TO
                        ;LOCATION ADDRESSED BY
                        ;REG 0
      INC R0 ;INCREMENT REG 0
      RET ;RETURN TO MAIN PROGRAM
  
```

**CLR A Clear Accumulator**

0 0 1 0	0 1 1 1
---------	---------

The contents of the accumulator are cleared to zero.

$A \leftarrow 0$

**CLR C Clear Carry Bit**

1 0 0 1	0 1 1 1
---------	---------

During normal program execution, the carry bit can be set to one by the ADD, ADDC, RLC, CPL C, RRC, and DAA instructions. This instruction resets the carry bit to zero.

$C \leftarrow 0$

**CLR F1 Clear Flag 1 (Not in 8021)**

1 0 1 0	0 1 0 1
---------	---------

Flag 1 is cleared to zero.

$(F1) \leftarrow 0$

**CLR F0 Clear Flag 0 (Not in 8021)**

1 0 0 0	0 1 0 1
---------	---------

Flag 0 is cleared to zero.

$(F0) \leftarrow 0$

**CPL A Complement Accumulator**

0 0 1 1	0 1 1 1
---------	---------

The contents of the accumulator are complemented. This is strictly a one's complement. Each one is changed to zero and vice-versa.

$(A) \leftarrow \text{NOT } (A)$

**Example:** Assume accumulator contains 01101010.

```
CPLA: CPL A           ;ACC CONTENTS ARE COMPLE-
                      ;MENTED TO 10010101
```

**CPL C Complement Carry Bit**

1 0 1 0	0 1 1 1
---------	---------

The setting of the carry bit is complemented; one is changed to zero, and zero is changed to one.

$(C) \leftarrow \text{NOT } (C)$

**Example:** Set C to one; current setting is unknown.

```
CTO1: CLR C           ;C IS CLEARED TO ZERO
      CPL C           ;C IS SET TO ONE
```

**CPL F0 Complement Flag 0** (Not in 8021)

1 0 0 1	0 1 0 1
---------	---------

The setting of flag 0 is complemented; one is changed to zero, and zero is changed to one.

$F0 \leftarrow \text{NOT } (F0)$

**CPL F1 Complement Flag 1** (Not in 8021)

1 0 1 1	0 1 0 1
---------	---------

The setting of flag 1 is complemented; one is changed to zero, and zero is changed to one.

$(F1) \leftarrow \text{NOT } (F1)$

**DA A Decimal Adjust Accumulator**

0 1 0 1	0 1 1 1
---------	---------

The 8-bit accumulator value is adjusted to form two 4-bit Binary Coded Decimal (BCD) digits following the binary addition of BCD numbers. The carry bit C is affected. If the contents of bits 0-3 are greater than nine, or if AC is one, the accumulator is incremented by six.

The four high-order bits are then checked. If bits 4-7 exceed nine, or if C is one, these bits are increased by six. If an overflow occurs, C is set to one.

**Example:** Assume accumulator contains 10011011.

DA A ;ACC ADJUSTED TO 00000001  
;WITH C SET

C	AC	7	4	3	0	
0	0	1	0	0	1	1 0 1 1
						0 1 1 0
						ADD SIX TO BITS 0-7
0	0	1	0	1	0	0 0 0 1
						0 1 1 0
						ADD SIX TO BITS 4-7
1	0	0	0	0	0	0 0 0 1
						OVERFLOW TO C

**DEC A Decrement Accumulator**

0 0 0 0	0 1 1 1
---------	---------

The contents of the accumulator are decremented by one.

$(A) \leftarrow (A) - 1$

## INSTRUCTION SET

**Example:** Decrement contents of external data memory location 63.

```
MOV R0,#3FH      ;MOVE '3F' HEX TO REG 0
MOVX A,@R0      ;MOVE CONTENTS OF LOCATION 63
                ;TO ACC
DEC A           ;DECREMENT ACC
MOVX @R0,A      ;MOVE CONTENTS OF ACC TO
                ;LOCATION 63 IN EXPANDED
                ;MEMORY
```

### DEC R<sub>r</sub> Decrement Register (Not in 8021)

1100	1rrr
------	------

The contents of working register 'r' are decremented by one.

$(Rr) \leftarrow (Rr) - 1$        $r = 0-7$

**Example:** DECR1: DEC R1      ;DECREMENT CONTENTS OF REG 1

### DIS I Disable External Interrupt (Not in 8021)

0001	0101
------	------

External interrupts are disabled. A low signal on the interrupt input pin has no effect.

### DIS TCNTI Disable Timer/Counter Interrupt (Not in 8021)

0011	0101
------	------

Timer/counter interrupts are disabled. Any pending timer interrupt request is cleared. The interrupt sequence is not initiated by an overflow, but the timer flag is set and time accumulation continues.

### DJNZ R<sub>r</sub>, address Decrement Register and Test

1110	1rrr	a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub>	a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>
------	------	---	---

This is a 2-cycle instruction. Register 'r' is decremented and tested for zero. If the register contains all zeros, program control falls through to the next instruction. If the register contents are not zero, control jumps to the specified 'address'.

The address in this case must evaluate to 8-bits, that is, the jump must be to a location within the current 256-location page.

$(Rr) \leftarrow (Rr) - 1$        $r = 0-7$

If Rr not 0

$(PC_{0-7}) \leftarrow \text{addr}$

Note: A 12-bit address specification does not cause an error if the DJNZ instruction and the jump target are on the same page. If the DJNZ instruction begins in location 255 of a page, it must jump to a target address on the following page.

**Example:** Increment values in data memory locations 50-54.

```

MOV R0,#50      ;MOVE '50' DEC TO ADDRESS
                ;REG 0
MOV R3,#5       ;MOVE '5' DEC TO COUNTER
                ;REG 3
INCR: INC @R0   ;INCREMENT CONTENTS OF
                ;LOCATION ADDRESSED BY
                ;REG 0
INC R0          ;INCREMENT ADDRESS IN REG 0
DJNZ R3, INCR  ;DECREMENT REG 3 — JUMP TO
                ;'INCR' IF REG 3 NONZERO
NEXT —        ;'NEXT' ROUTINE EXECUTED
                ;IF R3 IS ZERO
    
```

**EN I Enable External Interrupt** (Not in 8021)

0 0 0 0	0 1 0 1
---------	---------

External interrupts are enabled. A low signal on the interrupt input pin initiates the interrupt sequence.

**EN TCNTI Enable Timer/Counter Interrupt** (Not in 8021)

0 0 1 0	0 1 0 1
---------	---------

Timer/counter interrupts are enabled. An overflow of the timer/counter initiates the interrupt sequence.

**ENT0 CLK Enable Clock Output** (Not in 8021)

0 1 1 1	0 1 0 1
---------	---------

The test 0 pin is enabled to act as the clock output. This function is disabled by a system reset.

**Example:** EMTST0: ENT0 CLK ;ENABLE T0 AS CLOCK OUTPUT

**IN A,Pp Input Port or Data to Accumulator**

0 0 0 0	1 0 p p
---------	---------

This is a 2-cycle instruction. Data present on port 'p' is transferred (read) to the accumulator. In the 8021 IN A,P2 inputs P20-P23 to A0-A3 while A4-A7 is set to zero.

$(A) \leftarrow (Pp) \quad p=1-2$

## INSTRUCTION SET

**Example:** INP12: IN A,P1 ;INPUT PORT 1 CONTENTS  
;TO ACC  
MOV R6,A ;MOVE ACC CONTENTS TO  
;REG 6  
IN A,P2 ;INPUT PORT 2 CONTENTS  
;TO ACC  
MOV R7,A ;MOVE ACC CONTENTS TO REG 7

### INC A Increment Accumulator

0001	0111
------	------

The contents of the accumulator are incremented by one.

$(A) \leftarrow (A)+1$

**Example:** Increment contents of location 100 in external data memory.

INCA: MOV R0,#100 ;MOVE '100' DEC TO ADDRESS  
;REG 0  
MOVX A,@R0 ;MOVE CONTENTS OF LOCATION  
;100 TO ACC  
INC A ;INCREMENT A  
MOVX @R0,A ;MOVE ACC CONTENTS TO  
;LOCATION 100

### INC R<sub>r</sub> Increment Register

0001	1rrr
------	------

The contents of working register 'r' are incremented by one.

$(R_r) \leftarrow (R_r)+1$        $r=0-7$

**Example:** INCR0: INC R0 ;INCREMENT ADDRESS REG 0

### INC @R<sub>r</sub> Increment Data Memory Location

0001	000r
------	------

The contents of the resident data memory location addressed by register 'r' bits 0-5\*are incremented by one.

$((R_r)) \leftarrow ((R_r))+1$        $r=0-1$

**Example:** INCDM: MOV R1,#03FH ;MOVE ONES TO REG 1  
INC @R1 ;INCREMENT LOCATION 63

**IN A,P0      Input of Port 0 Data to Accumulator**

Same as INS A,BUS except no  $\overline{RD}$  pulse generated.

**INS A,BUS    Strobed Input of BUS Data to Accumulator**

0 0 0 0	1 0 0 0
---------	---------

This is a 2-cycle instruction. Data present on the BUS port is transferred (read) to the accumulator when the  $\overline{RD}$  pulse is dropped. (Refer to section on programming memory expansion for details).

$(A) \leftarrow (BUS)$

**Example:** INPBUS: INS A,BUS           ;INPUT BUS CONTENTS  
  ;TO ACC

**JBb address    Jump If Accumulator Bit is Set    (Not in 8021)**

$b_2$ $b_1$ $b_0$ 1	0 0 1 0	$a_7$ $a_6$ $a_5$ $a_4$	$a_3$ $a_2$ $a_1$ $a_0$
---------------------	---------	-------------------------	-------------------------

This is a 2-cycle instruction. Control passes to the specified address if accumulator bit 'b' is set to one.

$(PC_{0-7}) \leftarrow \text{addr}$                     If Bb=1  
 $(PC) = (PC)+2$                          If Bb=0

**Example:** JB4IS1: JB4 NEXT           ;JUMP TO 'NEXT' ROUTINE  
  ;IF ACC BIT 4=1

**JC address    Jump If Carry Is Set**

1 1 1 1	0 1 1 0	$a_7$ $a_6$ $a_5$ $a_4$	$a_3$ $a_2$ $a_1$ $a_0$
---------	---------	-------------------------	-------------------------

This is a 2-cycle instruction. Control passes to the specified address if the carry bit is set to one.

$(PC_{0-7}) \leftarrow \text{addr}$                     If C=1  
 $(PC) = (PC)+2$                          If C=0

**Example:** JC1: JC OVFLOW           ;JUMP TO 'OVFLOW' ROUTINE  
  ;IF C=1

**JF0 address    Jump If Flag 0 Is Set    (Not in 8021)**

1 0 1 1	0 1 1 0	$a_7$ $a_6$ $a_5$ $a_4$	$a_3$ $a_2$ $a_1$ $a_0$
---------	---------	-------------------------	-------------------------

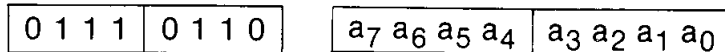
This is a 2-cycle instruction. Control passes to the specified address if flag 0 is set to one.

$(PC_{0-7}) \leftarrow \text{addr}$                     If F0=1  
 $(PC) = (PC)+2$                          If F0=0

**Example:** JF0IS1: JF0 TOTAL       ;JUMP TO 'TOTAL' ROUTINE  
  ;IF F0=1



**JF1 address    Jump If Flag 1 Is Set    (Not in 8021)**

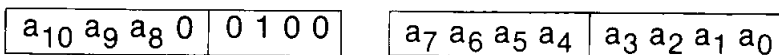


This is a 2-cycle instruction. Control passes to the specified address if flag 1 is set to one.

(PC<sub>0-7</sub>) ← addr                    If F1=1  
(PC) = (PC)+2                    IF F1=0

**Example:** JF1IS1: JF1 FILBUF            ;JUMP TO 'FILBUF'  
   ;ROUTINE IF F1=1

**JMP address    Direct Jump Within 2K Block**

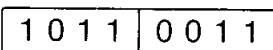


This is a 2-cycle instruction. Bits 0-10 of the program counter are replaced with the directly-specified address. The setting of PC bit 11 is determined by the most recent SELECT MB instruction.

(PC<sub>8-10</sub>) ← addr 8-10  
(PC<sub>0-7</sub>) ← addr 0-7  
(PC<sub>11</sub>) ← DBF

**Example:** JMP SUBTOT                    ;JUMP TO SUBROUTINE 'SUBTOT'  
   ;JUMP TO INSTRUCTION SIX LOCATIONS  
   ;BEFORE CURRENT LOCATION  
   ;JUMP TO ADDRESS '2F' HEX

**JMPP @A    Indirect Jump Within Page**

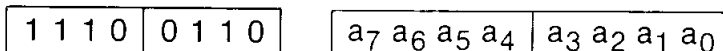


This is a 2-cycle instruction. The contents of the program memory location pointed to by the accumulator are substituted for the 'page' portion of the program counter (PC bits 0-7).

(PC<sub>0-7</sub>) ← ((A))

**Example:** Assume accumulator contains 0FH.  
JMPPAG: JMPP @A                    ;JUMP TO ADDRESS STORED IN  
   ;LOCATION 15 IN CURRENT PAGE

**JNC address    Jump If Carry Is Not Set**



This is a 2-cycle instruction. Control passes to the specified address if the carry bit is not set, that is, equals zero.

## INSTRUCTION SET

$(PC_{0-7}) \leftarrow \text{addr}$       If C=0  
 $(PC) = (PC)+2$       If C=1

**Example:** JC0: JNC NOVFO      ;JUMP TO 'NOVFO' ROUTINE  
                                 ;If C=0

### JNI address    **Jump If Interrupt Input is Low**    (Not in 8021)

1	0	0	0
---	---	---	---

0	1	1	0
---	---	---	---

a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>
----------------	----------------	----------------	----------------

a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
----------------	----------------	----------------	----------------

This is a 2-cycle instruction. Control passes to the specified address if the interrupt input signal is low (=0), that is, an external interrupt has been signaled. (This signal initiates an interrupt service sequence if the external interrupt is enabled.)

$(PC_{0-7}) \leftarrow \text{addr}$       If I=0  
 $(PC) = (PC)+2$       If I=1

**Example:** LOC 3: JNI EXTINT      ;JUMP TO 'EXTINT' ROUTINE  
                                 ;If I=0

### JNT0 address    **Jump If Test 0 Is Low**    (Not in 8021)

0	0	1	0
---	---	---	---

0	1	1	0
---	---	---	---

a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>
----------------	----------------	----------------	----------------

a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
----------------	----------------	----------------	----------------

This is a 2-cycle instruction. Control passes to the specified address, if the test 0 signal is low

$(PC_{0-7}) \leftarrow \text{addr}$       If T0=0  
 $(PC) = (PC)+2$       If T0=1

**Example:** JT0LOW: JNT0 60      ;JUMP TO LOCATION 60 DEC  
                                 ;IF T0=0

### JNT1 address    **Jump If Test 1 Is Low**

0	1	0	0
---	---	---	---

0	1	1	0
---	---	---	---

a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>
----------------	----------------	----------------	----------------

a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
----------------	----------------	----------------	----------------

This is a 2-cycle instruction. Control passes to the specified address, if the test 1 signal is low.

$(PC_{0-7}) \leftarrow \text{addr}$       If T1=0  
 $(PC) = (PC)+2$       If T1=1

### JNZ address    **Jump If Accumulator Is Not Zero**

1	0	0	1
---	---	---	---

0	1	1	0
---	---	---	---

a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>
----------------	----------------	----------------	----------------

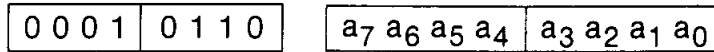
a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
----------------	----------------	----------------	----------------

This is a 2-cycle instruction. Control passes to the specified address if the accumulator contents are nonzero at the time this instruction is executed.

$(PC_{0-7}) \leftarrow \text{addr}$       If A≠0  
 $(PC) = (PC)+2$       If A=0

**Example:** JACCN0: JNZ 0ABH      ;JUMP TO LOCATION 'AB' HEX  
                                 ;IF ACC VALUE IS NONZERO

**JTF address    Jump If Timer Flag Is Set**

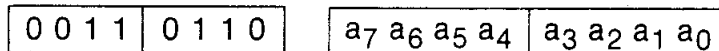


This is a 2-cycle instruction. Control passes to the specified address if the timer flag is set to one, that is, the timer/counter register has overflowed. Testing the timer flag resets it to zero. (This overflow initiates an interrupt service sequence if the timer-overflow interrupt is enabled.)

$(PC_{0-7}) \leftarrow \text{addr}$	If TF=1
$(PC) = (PC)+2$	If TF=0

**Example:**    JTF1: JTF TIMER                        ;JUMP TO 'TIMER' ROUTINE  
    ;IF TF=1

**JT0 address    Jump If Test 0 Is High    (Not in 8021)**

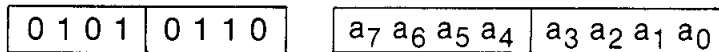


This is a 2-cycle instruction. Control passes to the specified address if the test 0 signal is high (=1).

$(PC_{0-7}) \leftarrow \text{addr}$	If T0=1
$(PC) = (PC)+2$	If T0=0

**Example:**    JT0HI: JT0 53                        ;JUMP TO LOCATION 53 DEC  
    ;IF T0=1

**JT1 address    Jump If Test 1 Is High**

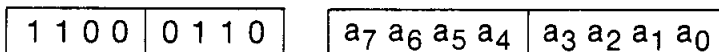


This is a 2-cycle instruction. Control passes to the specified address if the test 1 signal is high (=1).

$(PC_{0-7}) \leftarrow \text{addr}$	If T1=1
$(PC) = (PC)+2$	If T1=0

**Example:**    JT1HI: JT1 COUNT                    ;JUMP TO 'COUNT' ROUTINE  
    ;IF T1=1

**JZ address    Jump If Accumulator Is Zero**

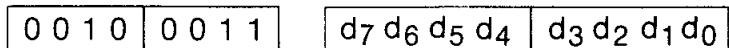


This is a 2-cycle instruction. Control passes to the specified address if the accumulator contains all zeros at the time this instruction is executed.

$(PC_{0-7}) \leftarrow \text{addr}$	If A=0
$(PC) = (PC)+2$	If A≠0

**Example:**    JACCO: JZ 0A3H                        ;JUMP TO LOCATION 'A3' HEX  
    ;IF ACC VALUE IS ZERO

**MOV A,#data Move Immediate Data to Accumulator**

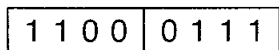


This is a 2-cycle instruction. The 8-bit value specified by 'data' is loaded in the accumulator.

(A) ← data

**Example:** MOV A,#0A3H ;MOVE 'A3' HEX TO ACC

**MOV A,PSW Move PSW Contents to Accumulator (Not in 8021)**



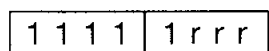
The contents of the program status word are moved to the accumulator.

(A) ← (PSW)

**Example:** Jump to 'RB1SET' routine if PSW bank switch, bit 4, is set.

```
BSCHK: MOV A,PSW ;MOVE PSW CONTENTS TO ACC
        JB4 RB1SET ;JUMP TO 'RB1SET' IF ACC
        ;BIT 4=1
```

**MOV A,Rr Move Register Contents to Accumulator**

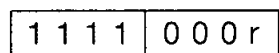


8-bits of data are moved from working register 'r' into the accumulator.

(A) ← (Rr) r=0-7

**Example:** MAR: MOV A,R3 ;MOVE CONTENTS OF REG 3  
;TO ACC

**MOV A,@Rr Move Data Memory Contents to Accumulator**



The contents of the resident data memory location addressed by bits 0-5\*of register 'r' are moved to the accumulator. Register 'r' contents are unaffected.

(A) ← ((Rr)) r=0-1

**Example:** Assume R1 contains 00110110.

```
MADM: MOV A,@R1 ;MOVE CONTENTS OF DATA MEM
        ;LOCATION 54 TO ACC
```

**MOV A,T Move Timer/Counter Contents to Accumulator**

0 1 0 0	0 0 1 0
---------	---------

The contents of the timer/event-counter register are moved to the accumulator.

$(A) \leftarrow (T)$

**Example:** Jump to "EXIT" routine when timer reaches '64', that is, when bit 6 set — assuming initialization 64,

```
TIMCHK: MOV A,T      ;MOVE TIMER CONTENTS TO
          ;ACC
          JB6 EXIT    ;JUMP TO 'EXIT' IF ACC BIT
          ;6=1
```

**MOV PSW,A Move Accumulator Contents to PSW (Not in 8021)**

1 1 0 1	0 1 1 1
---------	---------

The contents of the accumulator are moved into the program status word. All condition bits and the stack pointer are affected by this move.

$(PSW) \leftarrow (A)$

**Example:** Move up stack pointer by two memory locations, that is, increment the pointer by one.

```
INCPTR: MOV A,PSW    ;MOVE PSW CONTENTS TO ACC
          INC A       ;INCREMENT ACC BY ONE
          MOV PSW,A   ;MOVE ACC CONTENTS TO PSW
```

**MOV R<sub>r</sub>,A Move Accumulator Contents to Register**

1 0 1 0	1 r r r
---------	---------

The contents of the accumulator are moved to register 'r'.

$(R_r) \leftarrow (A)$  r=0-7

**Example:** MRA: MOV R0,A ;MOVE CONTENTS OF ACC TO ;REG 0

**MOV R<sub>r</sub>,#data Move Immediate Data to Register**

1 0 1 1	1 r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	d <sub>7</sub> d <sub>6</sub> d <sub>5</sub> d <sub>4</sub>	d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub>
---------	--	---	---

This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to register 'r'.

$(R_r) \leftarrow \text{data}$  r=0-7

## INSTRUCTION SET

**Examples:** MIR4: MOV R4,#HEXTEN ;THE VALUE OF THE SYMBOL  
; 'HEXTEN' IS MOVED INTO  
; REG 4  
MIR 5: MOV R5,#PI\*(R\*R);THE VALUE OF THE  
; EXPRESSION 'PI\*(R\*R)  
; IS MOVED INTO REG 5  
MIR 6: MOV R6, #0ADH ;'AD' HEX IS MOVED INTO  
; REG 6

### **MOV @R<sub>r</sub>,A Move Accumulator Contents to Data Memory**

1 0 1 0	0 0 0 r
---------	---------

This is a 2-cycle instruction. The contents of the accumulator are moved to the resident data memory location whose address is specified by bits 0-5\* of register 'r'. Register 'r' contents are unaffected.

$((Rr)) \leftarrow (A)$                       r=0-1

**Example:** Assume R0 contains 00000111.  
MDMA: MOV @R0,A            ;MOVE CONTENTS OF ACC TO  
;LOCATION 7 (REG 7)

### **MOV @R<sub>r</sub>,#data Move Immediate Data to Data Memory**

1 0 1 1	0 0 0 r	d <sub>7</sub> d <sub>6</sub> d <sub>5</sub> d <sub>4</sub>	d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub>
---------	---------	---	---

This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to the resident data memory location addressed by register 'r', bits 0-5\*.

$((Rr)) \leftarrow \text{data}$                       r=0-1

**Examples:** Move the hexadecimal value AC3F to locations 62-63.  
MIDM: MOV R0,#62            ;MOVE '62' DEC TO ADDR REG 0  
      MOV @R0,#0ACH ;MOVE 'AC' HEX TO LOCATION 62  
      INC R0                ;INCREMENT REG 0 TO '63'  
      MOV @R0,#3FH        ;MOVE '3F' HEX TO LOCATION 63

### **MOV T,A Move Accumulator Contents to Timer/Counter**

0 1 1 0	0 0 1 0
---------	---------

The contents of the accumulator are moved to the timer/event-counter register.

$(T) \leftarrow (A)$

**Example:** Initialize and start event counter.  
INITEC: CLR A                ;CLEAR ACC TO ZEROS  
      MOV T,A                ;MOVE ZEROS TO EVENT COUNTER  
      STRT CNT               ;START COUNTER

**MOVD A,Pp Move Port 4-7 Data to Accumulator**

0 0 0 0	1 1 p p
---------	---------

This is a 2-cycle instruction. Data on 8243 port 'p' is moved (read) to accumulator bits 0-3. Accumulator bits 4-7 are zeroed.

(0-3) ← (Pp)                      p=4-7  
(4-7) ← 0

Note: Bits 0-1 of the opcode are used to represent ports 4-7. If you are coding in binary rather than assembly language, the mapping is as follows:

Bits 1 0	Port
0 0	4
0 1	5
1 0	6
1 1	7

**Example:** INPPT5: MOVD A,P5            ;MOVE PORT 5 DATA TO ACC  
   ;BITS 0-3, ZERO ACC BITS 4-7

**MOVD Pp,A Move Accumulator Data to Port 4-7**

0 0 1 1	1 1 p p
---------	---------

This is a 2-cycle instruction. Data in accumulator bits 0-3 is moved (written) to 8243 port 'p'. Accumulator bits 4-7 are unaffected. (See NOTE above regarding port mapping.)

(Pp) ← (A<sub>0-3</sub>)                      p=4-7

**Example:** Move data in accumulator to ports 4 and 5.  
OUTP45: MOVD P4,A            ;MOVE ACC BITS 0-3 TO PORT 4  
                 SWAP A            ;EXCHANGE ACC BITS 0-3 AND 4-7  
                 MOVD P5,A            ;MOVE ACC BITS 0-3 TO PORT 5

**MOVP A,@A Move Current Page Data to Accumulator**

1 0 1 0	0 0 1 1
---------	---------

The contents of the program memory location addressed by the accumulator are moved to the accumulator. Only bits 0-7 of the program counter are affected, limiting the program memory reference to the current page. The program counter is restored following this operation

(PC<sub>0-7</sub>) ← (A)  
(A) ← ((PC))

Note: This is a 1-byte, 2-cycle instruction. If it appears in location 255 of a program memory page, @A addresses a location in the following page.

## INSTRUCTION SET

**Example:** MOV128: MOV A,#128 ;MOVE '128' DEC TO ACC  
                  MOVP A,@A ;CONTENTS OF 129th LOCATION  
                                  ;IN CURRENT PAGE ARE MOVED TO  
                                  ;ACC

### **MOVP3 A,@A Move Page 3 Data to Accumulator (Not in 8021)**

1 1 1 0	0 0 1 1
---------	---------

This is a 2-cycle instruction. The contents of the program memory location (within page 3) addressed by the accumulator are moved to the accumulator. The program counter is restored following this operation.

(PC<sub>0-7</sub>) ← (A)  
(PC<sub>8-11</sub>) ← 0011  
(A) ← ((PC))

**Example:** Look up ASCII equivalent of hexadecimal code in table contained at the beginning of page 3. Note that ASCII characters are designated by a 7-bit code; the eighth bit is always reset.

TABSCH: MOV A,#0B8H ;MOVE 'B8' HEX TO ACC (10111000)  
          ANL A,#7FH ;LOGICAL AND ACC TO MASK BIT  
                          ;7 (00111000)  
          MOVP3 A,@A ;MOVE CONTENTS OF LOCATION  
                          ;'38' HEX IN PAGE 3 TO ACC  
                          ;(ASCII '8')

Access contents of location in page 3 labelled TAB1.  
Assume current program location is not in page 3.

TABSCH: MOV A,#LOW TAB1 ;ISOLATE BITS 0-7 OF LABEL  
                          ;ADDRESS VALUE  
          MOVP3 A,@A ;MOVE CONTENTS OF PAGE 3  
                          ;LOCATION LABELED 'TAB1'  
                          ;TO ACC

### **MOVX A,@R<sub>r</sub> Move External-Data-Memory Contents to Accumulator**

1 0 0 0	0 0 0 r
---------	---------

(Not in 8021)

This is a 2-cycle instruction. The contents of the external data memory location addressed by register 'r' are moved to the accumulator. Register 'r' contents are unaffected.

(A) ← ((R<sub>r</sub>))                                   r=0-1

**Example:** Assume R1 contains 01110110.

MAXDM: MOVX A,@R1 ;MOVE CONTENTS OF LOCATION  
                          ;118 TO ACC



**MOVX @R<sub>r</sub>,A Move Accumulator Contents to External Data Memory**

1 0 0 1 | 0 0 0 r

(Not in 8021)

This is a 2-cycle instruction. The contents of the accumulator are moved to the external data memory location addressed by register 'r'. Register 'r' contents are unaffected.

((Rr) ← A

**Example:** Assume R0 contains 11000111.  
 MXDMA: MOVX @R0,A ;MOVE CONTENTS OF ACC TO  
 ;LOCATION 199 IN EXPANDED  
 ;DATA MEMORY

**NOP The NOP Instruction**

0 0 0 0 | 0 0 0 0

No operation is performed. Execution continues with the following instruction.

**ORL A,R<sub>r</sub> Logical OR Accumulator With Register Mask**

0 1 0 0 | 1 r r r

Data in the accumulator is logically ORed with the mask contained in working register 'r'.

(A) ← (A) OR (Rr) r=0-7

**Example:** ORREG: ORL A,R4 ;'OR' ACC CONTENTS WITH  
 ;MASK IN REG 4

**ORL A,@R<sub>r</sub> Logical OR Accumulator With Memory Mask**

0 1 0 0 | 0 0 0 r

Data in the accumulator is logically ORed with the mask contained in the resident data memory location referenced by register 'r', bits 0-5\*

(A) ← (A) OR ((Rr)) r=0-1

**Example:** ORDM: MOV R0,#3FH ;MOVE '3F' HEX TO REG 0  
 ORL A,@R0 ;'OR' ACC CONTENTS WITH MASK  
 ;IN LOCATION 63

**ORL A,#data Logical OR Accumulator With Immediate Mask**

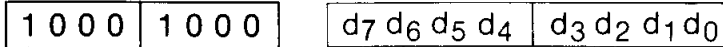
0 1 0 0 | 0 0 1 1 | d<sub>7</sub> d<sub>6</sub> d<sub>5</sub> d<sub>4</sub> | d<sub>3</sub> d<sub>2</sub> d<sub>1</sub> d<sub>0</sub>

This is a 2-cycle instruction. Data in the accumulator is logically ORed with an immediately-specified mask.

(A) ← (A) OR data

**Example:** ORID: ORL A,#'X' ;'OR' ACC CONTENTS WITH MASK  
 ;01011000 (ASCII VALUE OF 'X')

**ORL BUS,#data Logical OR BUS With Immediate Mask** (Not in 8021)

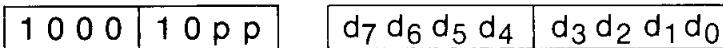


This is a 2-cycle instruction. Data on the BUS port is logically ORed with an immediately-specified mask. This instruction assumes prior specification of an 'OUTL BUS,A' instruction.

$$(BUS) \leftarrow (BUS) \text{ OR } data$$

**Example:** ORBUS: ORL BUS,#HEXMSK ;'OR' BUS CONTENTS WITH  
;MASK EQUAL VALUE OF SYMBOL  
;'HEXMSK'

**ORL Pp, #data Logical OR Port 1 or 2 With Immediate Mask** (Not in 8021)

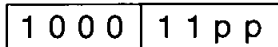


This is a 2-cycle instruction. Data on port 'p' is logically ORed with an immediately-specified mask.

$$(Pp) \leftarrow (Pp) \text{ OR } data \quad p=1-2$$

**Example:** ORP1: ORL P1, #0FFH ;'OR' PORT 1 CONTENTS WITH  
;MASK 'FF' HEX ( SET PORT 1  
;TO ALL ONES)

**ORLD Pp,A Logical OR Port 4-7 With Accumulator Mask**

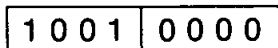


This is a 2-cycle instruction. Data on port 'p' is logically ORed with the digit mask contained in accumulator bits 0-3.

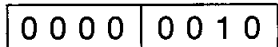
$$(Pp) \leftarrow (Pp) \text{ OR } (A_{0-3}) \quad p=4-7$$

**Example:** ORP7: ORLD P7,A ;'OR' PORT 7 CONTENTS  
;WITH ACC BITS 0-3

**OUTL P0,A Output Accumulator Data to Port 0** (8021 only)



**OUTL BUS,A Output Accumulator Data to BUS** (Not in 8021)



This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to the BUS port and latched. The latched data remains valid until altered by another OUTL instruction. Any other instruction requiring use of the BUS port (except INS) destroys the contents of the BUS latch. This includes expanded memory operations (such as the MOVX instruction). Logical operations on BUS data (AND, OR) assume the OUTL BUS,A instruction has been issued previously.

Does not apply for OUTL P0,A of 8021

$$(BUS) \leftarrow (A)$$

**Example:** OUTLBP: OUTL BUS,A ;OUTPUT ACC CONTENTS TO BUS

**OUTL Pp,A Output Accumulator Data to Port 1 or 2**

0 0 1 1	1 0 p p
---------	---------

This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to port 'p' and latched.

$$(Pp) \leftarrow (A) \qquad p=1-2$$

**Example:** OUTLP: MOV A,R7 ;MOVE REG 7 CONTENTS TO ACC  
                   OUTL P2,A ;OUTPUT ACC CONTENTS TO PORT 2  
                   MOV A,R6 ;MOVE REG 6 CONTENTS TO ACC  
                   OUTL P1,A ;OUTPUT ACC CONTENTS TO PORT 1

**RET Return Without PSW Restore**

1 0 0 0	0 0 1 1
---------	---------

This is a 2-cycle instruction. The stack pointer (PSW bits 0-2) is decremented. The program counter is then restored from the stack. PSW bits 4-7 are not restored.

$$(SP) \leftarrow (SP)-1$$

$$(PC) \leftarrow ((SP))$$
**RETR Return With PSW Restore (Not in 8021)**

1 0 0 1	0 0 1 1
---------	---------

This is a 2-cycle instruction. The stack pointer is decremented. The program counter and bits 4-7 of the PSW are then restored from the stack. Note that RETR should be used to return from an interrupt, but should not be used within the interrupt service routine as it signals the end of an interrupt routine.

$$(SP) \leftarrow (SP)-1$$

$$(PC) \leftarrow ((SP))$$

$$(PSW\ 4-7) \leftarrow ((SP))$$
**RL A Rotate Left Without Carry**

1 1 1 0	0 1 1 1
---------	---------

The contents of the accumulator are rotated left one bit. Bit 7 is rotated into the bit 0 position.

$$(AN+1) \leftarrow (An)$$

$$(A0) \leftarrow (A7) \qquad n=0-6$$

**Example:** Assume accumulator contains 10110001.  
 RLNC: RL A ;NEW ACC CONTENTS ARE 01100011.

### RLC A Rotate Left Through Carry

1	1	1	1	0	1	1	1
---	---	---	---	---	---	---	---

The contents of the accumulator are rotated left one bit. Bit 7 replaces the carry bit; the carry bit is rotated into the bit 0 position.

$$(AN+1) \leftarrow (An) \quad n=0-6$$

$$(A0) \leftarrow (C)$$

$$(C) \leftarrow (A7)$$

**Example:** Assume accumulator contains a 'signed' number; isolate sign without changing value.

```

RLTC: CLR C           ;CLEAR CARRY TO ZERO
      RLC A           ;ROTATE ACC LEFT, SIGN
                        ;BIT (7) IS PLACED IN CARRY
                        ;ROTATE ACC RIGHT — VALUE
                        ;(BITS 0-6) IS RESTORED,
                        ;CARRY UNCHANGED, BIT 7
                        ;IS ZERO
    
```

### RR A Rotate Right Without Carry

0	1	1	1	0	1	1	1
---	---	---	---	---	---	---	---

The contents of the accumulator are rotated right one bit. Bit 0 is rotated into the bit 7 position

$$(An) \leftarrow (AN+1) \quad n=0-6$$

$$(A7) \leftarrow (A0)$$

**Example:** Assume accumulator contains 10110001.

```

RRNC: RR A           ;NEW ACC CONTENTS ARE 11011000
    
```

### RRC A Rotate Right Through Carry

0	1	1	0	0	1	1	1
---	---	---	---	---	---	---	---

The contents of the accumulator are rotated right one bit. Bit 0 replaces the carry bit; the carry bit is rotated into the bit 7 position.

$$(An) \leftarrow (An+1) \quad n=0-6$$

$$(A7) \leftarrow (C)$$

$$(C) \leftarrow (A0)$$

**Example:** Assume carry is not set and accumulator contains 10110001.

```

RRTC: RRC A          ;CARRY IS SET AND ACC
                    ;CONTAINS 01011000
    
```

**SEL MBO Select Memory Bank 0** (Not in 8021)

1 1 1 0	0 1 0 1
---------	---------

PC bit 11 is set to zero on next JMP or CALL instruction.  
All references to program memory addresses fall within the range 0-2047.

(DBF) ← 0

**Example:** Assume program counter contains 834 Hex.

SEL MBO	;SELECT MEMORY BANK 0
JMP \$+20	;JUMP TO LOCATION
	;48 HEX

**SEL MB1 Select Memory Bank 1** (Not in 8021)

1 1 1 1	0 1 0 1
---------	---------

PC bit 11 is set to one on next JMP or CALL instruction.  
All references to program memory addresses fall within the range 2048-4095.

(DBF) ← 1

**SEL RB0 Select Register Bank 0** (Not in 8021)

1 1 0 0	0 1 0 1
---------	---------

PSW bit 4 is set to zero. References to working registers 0-7 address data memory locations 0-7. This is the recommended setting for normal program execution.

(BS) ← 0

**SEL RB1 Select Register Bank 1** (Not in 8021)

1 1 0 1	0 1 0 1
---------	---------

PSW bit 4 is set to one. References to working registers 0-7 address data memory locations 24-31. This is the recommended setting for interrupt service routines, since locations 0-7 are left intact. The setting of PSW bit 4 in effect at the time of an interrupt is restored by the RETR instruction when the interrupt service routine is completed.

(BS) ← 1

**Example:** Assume an external interrupt has occurred, control has passed to program memory location 3, and PSW bit 4 was zero before the interrupt.

LOC3: JN1 INIT	;JUMP TO ROUTINE 'INIT' IF
	;INTERRUPT INPUT IS ZERO

## INSTRUCTION SET

---

```
INIT: MOV R7,A           ;MOVE ACC CONTENTS TO
                        ;LOCATION 7
      SEL RB1           ;SELECT REG BANK 1
      MOV R7,#0FAH      ;MOVE 'FA' HEX TO LOCATION 31
      .
      .
      .
      SEL RB0           ;SELECT REG BANK 0
      MOV A,R7          ;RESTORE ACC FROM LOCATION 7
      RETR              ;RETURN — RESTORE PC AND PSW
```

### STOP TCNT Stop Timer/Event-Counter

0	1	1	0	0	1	0	1
---	---	---	---	---	---	---	---

This instruction is used to stop both time accumulation and event counting.

**Example:** Disable interrupt, but jump to interrupt routine after eight overflows and stop timer. Count overflows in register 7.

```
START: DIS TCNTI        ;DISABLE TIMER INTERRUPT
      CLR A             ;CLEAR ACC TO ZEROS
      MOV T,A           ;MOVE ZEROS TO TIMER
      MOV R7,A          ;MOVE ZEROS TO REG 7
      STRT T            ;START TIMER
MAIN:   JTF COUNT       ;JUMP TO ROUTINE 'COUNT'
                        ;IF TF=1 AND CLEAR TIMER FLAG
      JMP MAIN          ;CLOSE LOOP
COUNT: INC R7          ;INCREMENT REG 7
      MOV A,R7          ;MOVE REG 7 CONTENTS TO ACC
      JB3 INT           ;JUMP TO ROUTINE 'INT' IF ACC
                        ;BIT 3 IS SET (REG 7=8)
      JMP MAIN          ;OTHERWISE RETURN TO ROUTINE
                        ;MAIN
      .
      .
      .
INT:   STOP TCNT        ;STOP TIMER
      JMP 7H            ;JUMP TO LOCATION 7 (TIMER)
                        ;INTERRUPT ROUTINE
```

### **STRT CNT    Start Event Counter**

0 1 0 0	0 1 0 1
---------	---------

The test 1 (T1) pin is enabled as the event-counter input and the counter is started. The event-counter register is incremented with each high-to-low transition on the T1 pin.

**Example:** Initialize and start event counter. Assume overflow is desired with first T1 input.

```

STARTC: EN TCNTI      ;ENABLE COUNTER INTERRUPT
         MOV A,#0FFH  ;MOVE 'FF' HEX (ONES) TO
                   ;ACC
         MOV T,A      ;MOVE ONES TO COUNTER
         STRT CNT    ;ENABLE T1 AS COUNTER
                   ;INPUT AND START
    
```

### **STRT T    Start Timer**

0 1 0 1	0 1 0 1
---------	---------

Timer accumulation is initiated in the timer register. The register is incremented every 32 instruction cycles. The prescaler which counts the 32 cycles is cleared but the timer register is not.

**Example:** Initialize and start timer.

```

STARTT: CLR A        ;CLEAR ACC TO ZEROS
         MOV T,A     ;MOVE ZEROS TO TIMER
         EN TCNTI   ;ENABLE TIMER INTERRUPT
         STRT T     ;START TIMER
    
```

### **SWAP A    Swap Nibbles Within Accumulator**

0 1 0 0	0 1 1 1
---------	---------

Bits 0-3 of the accumulator are swapped with bits 4-7 of the accumulator.

$(A_{4-7}) \leftrightarrow (A_{0-3})$

**Example:** Pack bits 0-3 of locations 50-51 into location 50.

```

PCKDIG: MOV R0, #50  ;MOVE '50' DEC TO REG 0
         MOV R1, #51  ;MOVE '51' DEC TO REG 1
         XCHD A,@R0  ;EXCHANGE BITS 0-3 OF ACC
                   ;AND LOCATION 50
         SWAP A      ;SWAP BITS 0-3 AND 4-7 OF ACC
         XCHD A,@R1  ;EXCHANGE BITS 0-3 OF ACC AND
                   ;LOCATION 51
         MOV @R0,A   ;MOVE CONTENTS OF ACC TO
                   ;LOCATION 50
    
```

**XCH A,R<sub>r</sub> Exchange Accumulator-Register Contents**

0010	1rrr
------	------

The contents of the accumulator and the contents of working register 'r' are exchanged.

$$(A) \longleftrightarrow (Rr) \quad r=0-7$$

**Example:** Move PSW contents to Reg 7 without losing accumulator contents.

```
XCHAR7: XCH A,R7      ;EXCHANGE CONTENTS OF REG 7
          ;AND ACC
          MOV A, PSW   ;MOVE PSW CONTENTS TO ACC
          XCH A,R7      ;EXCHANGE CONTENTS OF REG 7
          ;AND ACC AGAIN
```

**XCH A,@R<sub>r</sub> Exchange Accumulator and Data Memory Contents**

0010	000r
------	------

The contents of the accumulator and the contents of the resident data memory location addressed by bits 0-5\*of register 'r' are exchanged. Register 'r' contents are unaffected.

$$(A) \longleftrightarrow ((Rr)) \quad r=0-1$$

**Example:** Decrement contents of location 52.

```
DEC52: MOV R0,#52     ;MOVE '52' DEC TO ADDRESS
          ;REG 0
          XCH A,@R0    ;EXCHANGE CONTENTS OF ACC
          ;AND LOCATION 52
          DEC A         ;DECREMENT ACC CONTENTS
          XCH A,@R0    ;EXCHANGE CONTENTS OF ACC
          ;AND LOCATION 52 AGAIN
```

**XCHD A,@R<sub>r</sub> Exchange Accumulator and Data Memory 4-Bit Data**

0011	000r
------	------

This instruction exchanges bits 0-3 of the accumulator with bits 0-3 of the data memory location addressed by bits 0-5\*of register 'r'. Bits 4-7 of the accumulator, bits 4-7 of the data memory location, and the contents of register 'r' are unaffected.

$$(A_{0-3}) \longleftrightarrow ((Rr_{0-3})) \quad r=0-1$$



## INSTRUCTION SET

**Example:** Assume program counter contents have been stacked in locations 22-23.

```
XCHNIB: MOV R0,#23    ;MOVE '23' DEC TO REG 0
          CLR A        ;CLEAR ACC TO ZEROS
          XCHD A,@R0   ;EXCHANGE BITS 0-3 OF ACC
                          ;AND LOCATION 23 (BITS 8-11
                          ;OF PC ARE ZEROED, ADDRESS
                          ;REFERS TO PAGE 0)
```

### **XRL A,R<sub>r</sub> Logical XOR Accumulator With Register Mask**

1	1	0	1	1	r	r	r
---	---	---	---	---	---	---	---

Data in the accumulator is EXCLUSIVE ORed with the mask contained in working register 'r'.

$(A) \leftarrow (A) \text{ XOR } (R_r) \quad r=0-7$

**Example:** XORREG: XRL A,R5 ;'XOR' ACC CONTENTS WITH  
;MASK IN REG 5

### **XRL A,@R<sub>r</sub> Logical XOR Accumulator With Memory Mask**

1	1	0	1	0	0	0	r
---	---	---	---	---	---	---	---

Data in the accumulator is EXCLUSIVE ORed with the mask contained in the data memory location addressed by register 'r', bits 0-5\*.

$(A) \leftarrow (A) \text{ XOR } ((R_r)) \quad r=0-1$

**Example:** XORDM: MOV R1, #20H ;MOVE '20' HEX TO REG 1  
XRL A,@R1 ;'XOR' ACC CONTENTS WITH MASK  
;IN LOCATION 32

### **XRL A,#data Logical XOR Accumulator With Immediate Mask**

1	1	0	1	0	0	1	1	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
---	---	---	---	---	---	---	---	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

This is a 2-cycle instruction. Data in the accumulator is EXCLUSIVE ORed with an immediately-specified mask.

$(A) \leftarrow (A) \text{ XOR } \text{data}$

**Example:** XORID: XOR A,#HEXTEN ;XOR CONTENTS OF ACC WITH  
;MASK EQUAL VALUE OF SYMBOL  
;'HEXTEN'



**PRELIMINARY**  
Notice: This is not a final specification. Some parameters are subject to change.

# 8048/8648/8748/8035

## SINGLE COMPONENT 8-BIT MICROCOMPUTER

- 8048 Mask Programmable ROM
- 8648 One-Time Factory Programmable EPROM
- 8748 User Programmable/Erasable EPROM
- 8035 External ROM or EPROM

- 8-Bit CPU, ROM, RAM, I/O in Single Package
- Interchangeable ROM and EPROM Versions
- Single 5V Supply
- 2.5  $\mu$ sec and 5.0  $\mu$ sec Cycle Versions: All Instructions 1 or 2 Cycles
- Over 90 Instructions: 70% Single Byte
- 1K x 8 ROM/EPROM
- 64 x 8 RAM
- 27 I/O Lines
- Interval Timer/Event Counter
- Easily Expandable Memory and I/O
- Compatible with 8000 Series Peripherals
- Single Level Interrupt

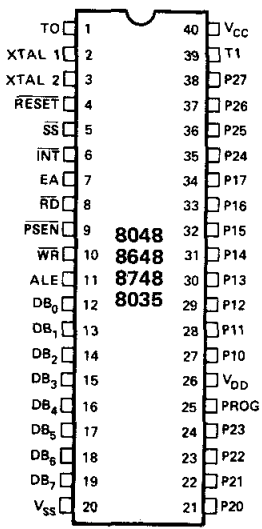
The Intel® 8048/8648/8748/8035 is a totally self-sufficient 8-bit parallel computer fabricated on a single silicon chip using Intel's N-channel silicon gate MOS process.

The 8048 contains a 1K x 8 program memory, a 64 x 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to onboard oscillator and clock circuits. For systems that require extra capability, the 8048 can be expanded using standard memories and MCS-80™ (8080A) peripherals. The 8035 is the equivalent of an 8048 without program memory. The 8035L has the RAM power down mode of the 8048 while the 8035 does not.

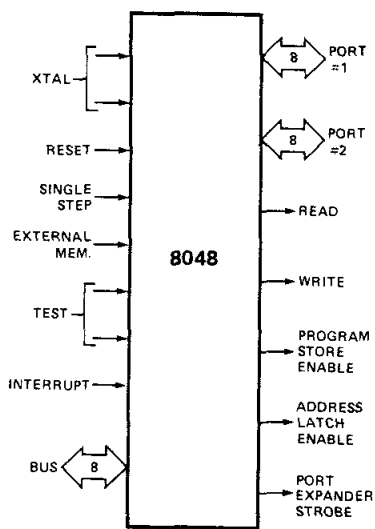
To reduce development problems to a minimum and provide maximum flexibility, three interchangeable pin-compatible versions of this single component microcomputer exist: the 8748 with user-programmable and erasable EPROM program memory for prototype and preproduction systems, the 8048 with factory-programmed mask ROM program memory for low cost, high volume production, and the 8035 without program memory for use with external program memories.

This microprocessor is designed to be an efficient controller as well as an arithmetic processor. The 8048 has extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single byte instructions and no instructions over 2 bytes in length.

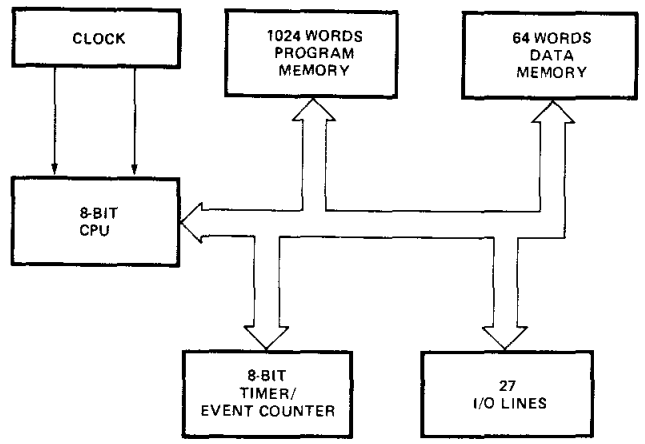
### PIN CONFIGURATION



### LOGIC SYMBOL



### BLOCK DIAGRAM



PRELIMINARY  
 Warning: This is not a final specification.  
 Parameters listed are subject to change.

## PIN DESCRIPTION

Designation	Pin #	Function	Designation	Pin #	Function
V <sub>SS</sub>	20	Circuit GND potential	$\overline{RD}$	8	Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device.  Used as a read strobe to external data memory. (Active low)
V <sub>DD</sub>	26	Programming power supply; +25V during program, +5V during operation for both ROM and PROM. Low power standby pin in 8048 ROM version.	$\overline{RESET}$	4	Input which is used to initialize the processor. Also used during PROM programming verification, and power down. (Active low) (Non TTL V <sub>IH</sub> )
V <sub>CC</sub>	40	Main power supply; +5V during operation and programming.	$\overline{WR}$	10	Output strobe during a bus write. (Active low)  Used as write strobe to external data memory.
PROG	25	Program pulse (+25V) input pin during 8748 programming.  Output strobe for 8243 I/O expander.	ALE	11	Address latch enable. This signal occurs once during each cycle and is useful as a clock output.  The negative edge of ALE strobes address into external data and program memory.
P10-P17 Port 1	27-34	8-bit quasi-bidirectional port.	$\overline{PSEN}$	9	Program store enable. This output occurs only during a fetch to external program memory. (Active low)
P20-P27 Port 2	21-24 35-38	8-bit quasi-bidirectional port.  P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.	$\overline{SS}$	5	Single step input can be used in conjunction with ALE to "single step" the processor through each instruction. (Active low)
DB <sub>0</sub> -DB <sub>7</sub> BUS	12-19	True bidirectional port which can be written or read synchronously using the $\overline{RD}$ , $\overline{WR}$ strobes. The port can also be statically latched.  Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of $\overline{PSEN}$ . Also contains the address and data during an external RAM data store instruction, under control of ALE, $\overline{RD}$ , and $\overline{WR}$ .	EA	7	External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing and program verification. (Active high)
T0	1	Input pin testable using the conditional transfer instructions JTO and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction. T0 is also used during programming.	XTAL1	2	One side of crystal input for internal oscillator. Also input for external source. (Non TTL V <sub>IH</sub> )
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.	XTAL2	3	Other side of crystal input.
$\overline{INT}$	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low)			

## INSTRUCTION SET

	Mnemonic	Description	Bytes	Cycle		Mnemonic	Description	Bytes	Cycles
Accumulator	ADD A, R	Add register to A	1	1	Subroutine	CALL	Jump to subroutine	2	2
	ADD A, @R	Add data memory to A	1	1		RET	Return	1	2
	ADD A, #data	Add immediate to A	2	2		RETR	Return and restore status	1	2
	ADDC A, R	Add register with carry	1	1					
	ADDC A, @R	Add data memory with carry	1	1		CLR C	Clear carry	1	1
	ADDC A, #data	Add immediate with carry	2	2		CPL C	Complement carry	1	1
	ANL A, R	And register to A	1	1		CLR F0	Clear flag 0	1	1
	ANL A, @R	And data memory to A	1	1		CPL F0	Complement flag 0	1	1
	ANL A, #data	And immediate to A	2	2		CLR F1	Clear flag 1	1	1
	ORL A, R	Or register to A	1	1		CPL F1	Complement flag 1	1	1
	ORL A, @R	Or data memory to A	1	1					
	ORL A, #data	Or immediate to A	2	2		MOV A, R	Move register to A	1	1
	XRL A, R	Exclusive or register to A	1	1		MOV A, @R	Move data memory to A	1	1
	XRL A, @R	Exclusive or data memory to A	1	1		MOV A, #data	Move immediate to A	2	2
XRL A, #data	Exclusive or immediate to A	2	2	MOV R, A	Move A to register	1	1		
INC A	Increment A	1	1	MOV @R, A	Move A to data memory	1	1		
DEC A	Decrement A	1	1	MOV R, #data	Move immediate to register	2	2		
CLR A	Clear A	1	1	MOV @R, #data	Move immediate to data memory	2	2		
CPL A	Complement A	1	1	MOV A, PSW	Move PSW to A	1	1		
DA A	Decimal adjust A	1	1	MOV PSW, A	Move A to PSW	1	1		
SWAP A	Swap nibbles of A	1	1	XCH A, R	Exchange A and register	1	1		
RL A	Rotate A left	1	1	XCH A, @R	Exchange A and data memory	1	1		
RLC A	Rotate A left through carry	1	1	XCHD A, @R	Exchange nibble of A and register	1	1		
RR A	Rotate A right	1	1	MOVX A, @R	Move external data memory to A	1	2		
RRC A	Rotate A right through carry	1	1	MOVX @R, A	Move A to external data memory	1	2		
				MOV P, A	Move to A from current page	1	2		
				MOV P, @A	Move to A from page 3	1	2		
Input/Output	IN A, P	Input port to A	1	2					
	OUTL P, A	Output A to port	1	2	MOV A, T	Read timer/counter	1	1	
	ANL P, #data	And immediate to port	2	2	MOV T, A	Load timer/counter	1	1	
	ORL P, #data	Or immediate to port	2	2	STRT T	Start timer	1	1	
	INS A, BUS	Input BUS to A	1	2	STRT CNT	Start counter	1	1	
	OUTL BUS, A	Output A to BUS	1	2	STOP TCNT	Stop timer/counter	1	1	
	ANL BUS, #data	And immediate to BUS	2	2	EN TCNTI	Enable timer/counter interrupt	1	1	
	ORL BUS, #data	Or immediate to BUS	2	2	DIS TCNTI	Disable timer/counter interrupt	1	1	
	MOVD A, P	Input expander port to A	1	2					
	MOVD P, A	Output A to expander port	1	2	EN I	Enable external interrupt	1	1	
ANLD P, A	And A to expander port	1	2	DIS I	Disable external interrupt	1	1		
ORLD P, A	Or A to expander port	1	2	SEL RBO	Select register bank 0	1	1		
				SEL RB1	Select register bank 1	1	1		
Registers	INC R	Increment register	1	1	SEL MBO	Select memory bank 0	1	1	
	INC @R	Increment data memory	1	1	SEL MB1	Select memory bank 1	1	1	
	DEC R	Decrement register	1	1	ENT0 CLK	Enable clock output on T0	1	1	
Branch	JMP addr	Jump unconditional	2	2					
	JMPP @A	Jump indirect	1	2	NOP	No operation	1	1	
	DJNZ R, addr	Decrement register and skip	2	2					
	JC addr	Jump on carry = 1	2	2					
	JNC addr	Jump on carry = 0	2	2					
	JZ addr	Jump on A zero	2	2					
	JNZ addr	Jump on A not zero	2	2					
	JT0 addr	Jump on T0 = 1	2	2					
	JNT0 addr	Jump on T0 = 0	2	2					
	JT1 addr	Jump on T1 = 1	2	2					
	JNT1 addr	Jump on T1 = 0	2	2					
	JF0 addr	Jump on F0 = 1	2	2					
	JF1 addr	Jump on F1 = 1	2	2					
	JTF addr	Jump on timer flag	2	2					
	JNI addr	Jump on $\overline{\text{INT}} = 0$	2	2					
	JBb addr	Jump on accumulator bit	2	2					

**ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias ..... 0°C to 70°C  
 Storage Temperature ..... -65°C to +150°C  
 Voltage On Any Pin With Respect  
 to Ground ..... -0.5V to +7V  
 Power Dissipation ..... 1.5 Watt

\*COMMENT:  
 Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

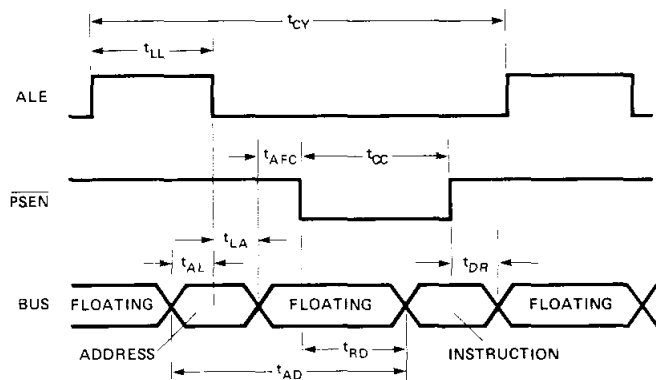
**D.C. AND OPERATING CHARACTERISTICS**  $T_A = 0^\circ\text{C to } 70^\circ\text{C}$ ,  $V_{CC} = V_{DD} = +5V \pm 10\%^*$ ,  $V_{SS} = 0V$

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.	Max.		
$V_{IL}$	Input Low Voltage	-0.5		.8	V	
$V_{IH}$	Input High Voltage (All Except XTAL1, XTAL2, $\overline{\text{RESET}}$ )	2.0		$V_{CC}$	V	
$V_{IH1}$	Input High Voltage ( $\overline{\text{RESET}}$ , X1, X2)	3.8		$V_{CC}$	V	
$V_{OL}$	Output Low Voltage (BUS, $\overline{\text{RD}}$ , $\overline{\text{WR}}$ , $\overline{\text{PSEN}}$ , ALE)			.45	V	$I_{OL} = 2.0\text{mA}$
$V_{OL1}$	Output Low Voltage (All Other Outputs Except PROG)			.45	V	$I_{OL} = 1.6\text{mA}$
$V_{OL2}$	Output Low Voltage (PROG)			.45	V	$I_{OL} = 1.0\text{mA}$
$V_{OH}$	Output High Voltage (BUS, $\overline{\text{RD}}$ , $\overline{\text{WR}}$ , $\overline{\text{PSEN}}$ , ALE)	2.4			V	$I_{OH} = -100\mu\text{A}$
$V_{OH1}$	Output High Voltage (All Other Outputs)	2.4			V	$I_{OH} = -50\mu\text{A}$
$I_{IL}$	Input Leakage Current (T1, $\overline{\text{INT}}$ )			$\pm 10$	$\mu\text{A}$	$V_{SS} \leq V_{IN} \leq V_{CC}$
$I_{OL}$	Output Leakage Current (BUS, T0) (High Impedance State)			$\pm 10$	$\mu\text{A}$	$V_{SS} + .45 \leq V_{IN} \leq V_{CC}$
$I_{DD}$	$V_{DD}$ Supply Current		10	20	mA	
$I_{DD} + I_{CC}$	Total Supply Current		65	135	mA	

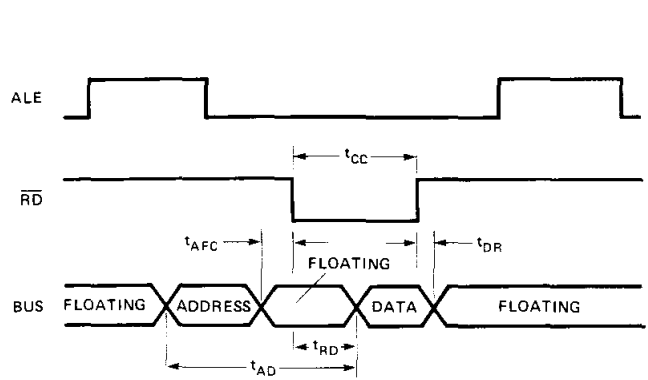
\*Standard 8748 and 8035  $\pm 5\%$ ,  $\pm 10\%$  available.

**WAVEFORMS**

**Instruction Fetch From External Program Memory**

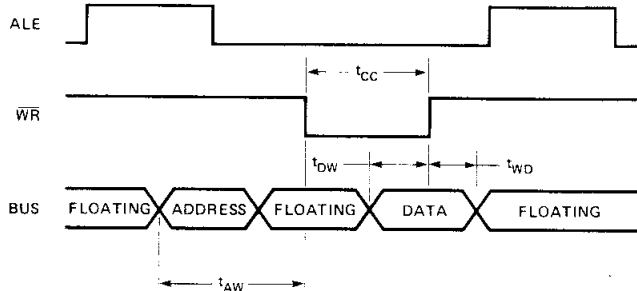


**Read From External Data Memory**



**PRELIMINARY**  
 Note: This is not a final specification. Some parameters listed are subject to change.

**Write to External Data Memory**



**A.C. CHARACTERISTICS**  $T_A = 0^\circ\text{C to } 70^\circ\text{C}$ ,  $V_{CC} = V_{DD} = +5V \pm 10\%$ ,  $V_{SS} = 0V$

Symbol	Parameter	8048 8648 (Note 2) 8748/8035/8035L		8748-8 8035-8		Unit	Conditions (Note 1)
		Min.	Max.	Min.	Max.		
t <sub>LL</sub>	ALE Pulse Width	400		600		ns	
t <sub>AL</sub>	Address Setup to ALE	150		150		ns	
t <sub>LA</sub>	Address Hold from ALE	80		80		ns	
t <sub>CC</sub>	Control Pulse Width ( $\overline{\text{PSEN}}$ , $\overline{\text{RD}}$ , $\overline{\text{WR}}$ )	700		1500		ns	
t <sub>DW</sub>	Data Setup before $\overline{\text{WR}}$	500		640		ns	
t <sub>WD</sub>	Data Hold After $\overline{\text{WR}}$	120		120		ns	C <sub>L</sub> = 20pF
t <sub>CY</sub>	Cycle Time	2.5	15.0	4.17	15.0	μs	6 MHz XTAL (3.6MHz XTAL for -8)
t <sub>DR</sub>	Data Hold	0	200	0	200	ns	
t <sub>RD</sub>	$\overline{\text{PSEN}}$ , $\overline{\text{RD}}$ to Data In		500		750	ns	
t <sub>AW</sub>	Address Setup to $\overline{\text{WR}}$	230		260		ns	
t <sub>AD</sub>	Address Setup to Data In		950		1450	ns	
t <sub>AFC</sub>	Address Float to $\overline{\text{RD}}$ , $\overline{\text{PSEN}}$	0		0		ns	

Note 1: Control outputs: C<sub>L</sub> = 80 pF  
 BUS Outputs: C<sub>L</sub> = 150 pF, t<sub>CY</sub> = 2.5μs

\*Standard 8748 and 8035 ± 5%, ± 10% available.

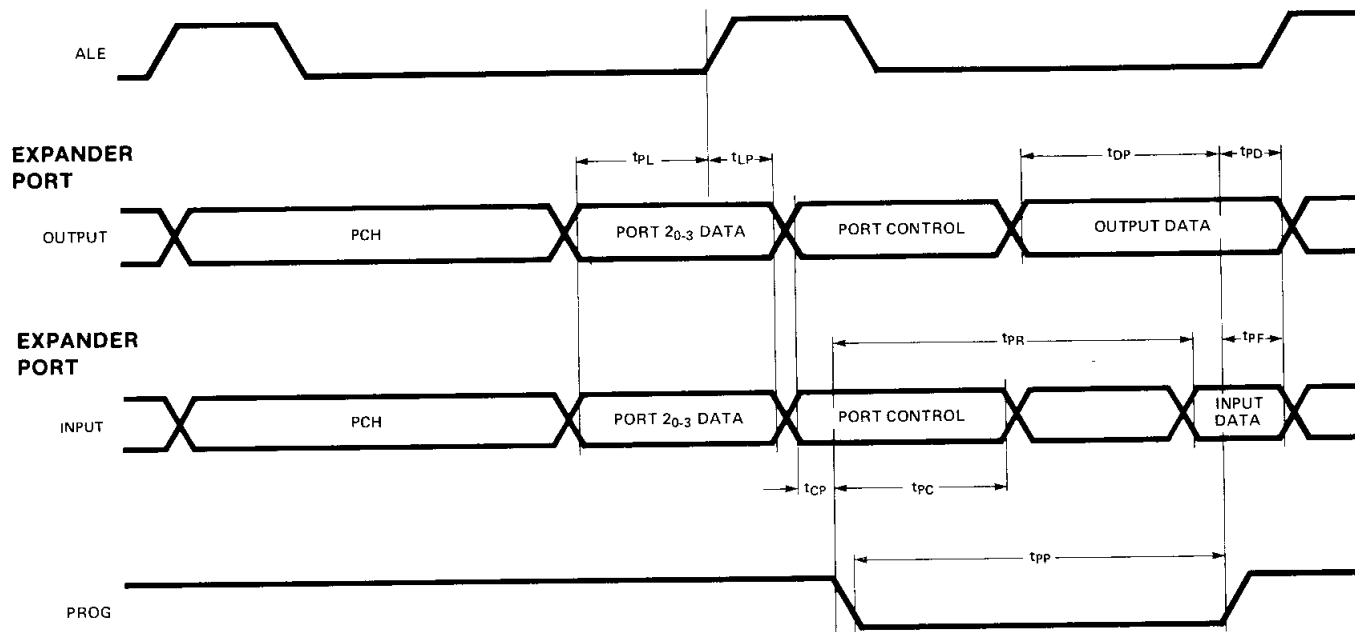
Note 2: The 8648 is a one-time programmable (at the factory) 8748 which can be ordered as the first 25 pieces of a new 8048 ROM order. The substitution of 8648's for 8048's allows for very fast turnaround for initial code verification and evaluation units. The 8648, like the 8748, is electrically and functionally interchangeable with the 8048 with the exception of the powerdown mode which the 8648 does not support and ±5% supply tolerance instead of ±10%.

**A.C. CHARACTERISTICS (PORT 2 TIMING)**

$T_A = 0^\circ\text{C to } 70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 10\%$

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t <sub>CP</sub>	Port Control Setup Before Falling Edge of $\overline{\text{PROG}}$	110		ns	
t <sub>PC</sub>	Port Control Hold After Falling Edge of $\overline{\text{PROG}}$	140		ns	
t <sub>PR</sub>	$\overline{\text{PROG}}$ to Time P2 Input Must Be Valid		810	ns	
t <sub>DP</sub>	Output Data Setup Time	220		ns	
t <sub>PD</sub>	Output Data Hold Time	65		ns	
t <sub>PF</sub>	Input Data Hold Time	0	150	ns	
t <sub>PP</sub>	$\overline{\text{PROG}}$ Pulse Width	1510		ns	
t <sub>PL</sub>	Port 2 I/O Data Setup	400		ns	
t <sub>LP</sub>	Port 2 I/O Data Hold	150		ns	

## PORT 2 TIMING



## PROGRAMMING, VERIFYING, AND ERASING THE 8748 EPROM

### Programming Verification

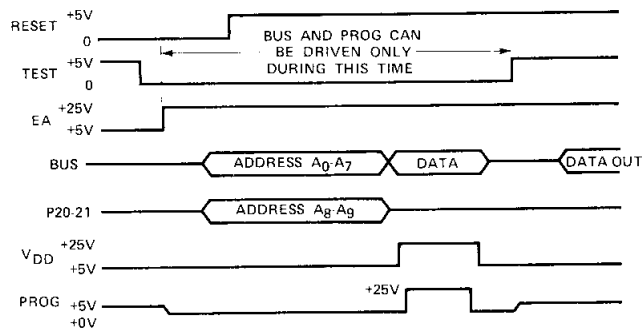
In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

Pin	Function
XTAL 1	Clock Input (1 to 6MHz)
Reset	Initialization and Address Latching
Test 0	Selection of Program or Verify Mode
EA	Activation of Program/Verify Modes
BUS	Address and Data Input Data Output During Verify
P20-1	Address Input
V <sub>DD</sub>	Programming Power Supply
PROG	Program Pulse Input

#### WARNING:

An attempt to program a missocketed 8748 will result in severe damage to the part. An indication of a properly socketed part is the appearance of the ALE clock output. The lack of this clock may be used to disable the programmer.

### Programming/Verification Sequence



### The Program/Verify sequence is:

1. V<sub>DD</sub> = 5v, Clock applied or internal oscillator operating, RESET = 0v, TEST 0 = 5v, EA = 5v, BUS and PROG floating.
2. Insert 8748 in programming socket
3. TEST 0 = 0v (select program mode)
4. EA = 25v (activate program mode)
5. Address applied to BUS and P20-1
6. RESET = 5v (latch address)
7. Data applied to BUS
8. V<sub>DD</sub> = 25v (programming power)
9. PROG = 0v followed by one 50ms pulse to 25v
10. V<sub>DD</sub> = 5v
11. TEST 0 = 5v (verify mode)
12. Read and verify data on BUS
13. TEST 0 = 0v
14. RESET = 0v and repeat from step 5
15. Programmer should be at conditions of step 1 when 8748 is removed from socket.

**PRELIMINARY**  
 NOTE: This is not a final specification. Some parameter limits are subject to change.

**AC TIMING SPECIFICATION FOR PROGRAMMING**
 $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 5\%$ ,  $V_{DD} = 25\text{V} \pm 1\text{V}$ 

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t <sub>AW</sub>	Address Setup Time to $\overline{\text{RESET}}$ ↑	4t <sub>cy</sub>			
t <sub>WA</sub>	Address Hold Time After $\overline{\text{RESET}}$ ↓	4t <sub>cy</sub>			
t <sub>DW</sub>	Data in Setup Time to PROG ↑	4t <sub>cy</sub>			
t <sub>WD</sub>	Data in Hold Time After PROG ↓	4t <sub>cy</sub>			
t <sub>PH</sub>	$\overline{\text{RESET}}$ Hold Time to Verify	4t <sub>cy</sub>			
t <sub>VDDW</sub>	V <sub>DD</sub>	4t <sub>cy</sub>			
t <sub>VDDH</sub>	V <sub>DD</sub> Hold Time After PROG ↓	0			
t <sub>PW</sub>	Program Pulse Width	50	60	MS	
t <sub>TW</sub>	Test 0 Setup Time for Program Mode	4t <sub>cy</sub>			
t <sub>TW</sub>	Test 0 Hold Time After Program Mode	4t <sub>cy</sub>			
t <sub>DO</sub>	Test 0 to Data Out Delay		4t <sub>cy</sub>		
t <sub>WW</sub>	$\overline{\text{RESET}}$ Pulse Width to Latch Address	4t <sub>cy</sub>			
t <sub>r</sub> , t <sub>f</sub>	V <sub>DD</sub> and PROG Rise and Fall Times	0.5	2.0	μs	
t <sub>CY</sub>	CPU Operation Cycle Time	5.0		μs	
t <sub>RE</sub>	$\overline{\text{RESET}}$ Setup Time Before EA ↑	4t <sub>cy</sub>			

Note: If Test 0 is high too can be triggered by  $\overline{\text{RESET}}$  ↓.

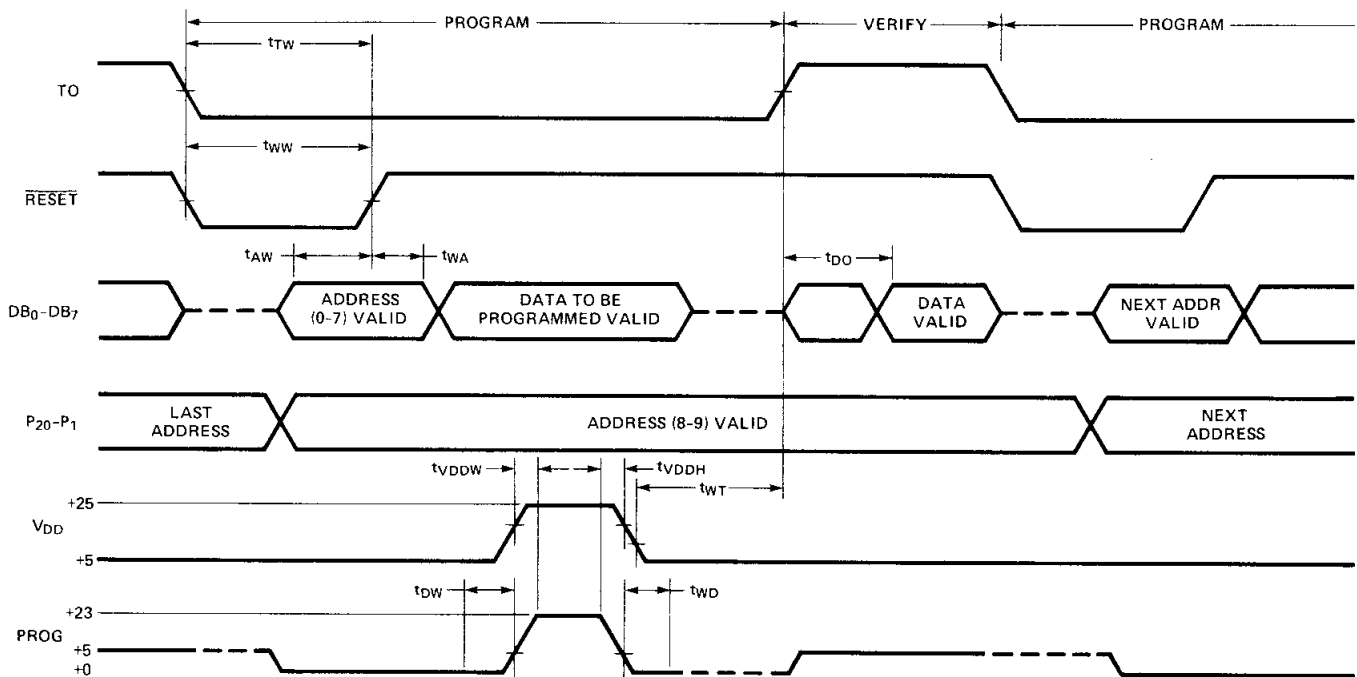
**DC SPECIFICATION FOR PROGRAMMING**
 $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 5\%$ ,  $V_{DD} = 25\text{V} \pm 1\text{V}$ 

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V <sub>DOH</sub>	V <sub>DD</sub> Program Voltage High Level	24.0	26.0	V	
V <sub>DDL</sub>	V <sub>DD</sub> Voltage Low Level	4.75	5.25	V	
V <sub>PH</sub>	PROG Program Voltage High Level	21.5	24.5	V	
V <sub>PL</sub>	PROG Voltage Low Level		0.2	V	
V <sub>EAH</sub>	EA Program or Verify Voltage High Level	21.5	24.5	V	
V <sub>EAL</sub>	EA Voltage Low Level		5.25	V	
I <sub>DD</sub>	V <sub>DD</sub> High Voltage Supply Current		30.0	mA	
I <sub>PROG</sub>	PROG High Voltage Supply Current		16.0	mA	
I <sub>EA</sub>	EA High Voltage Supply Current		1.0	mA	

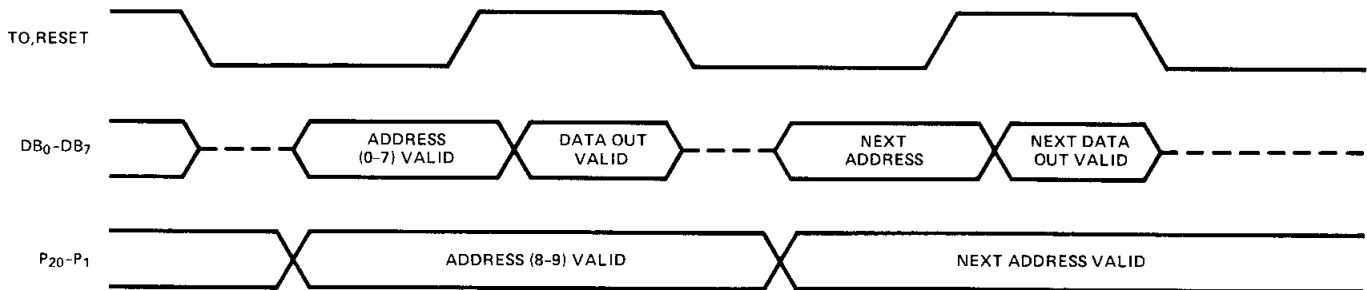


## WAVEFORMS FOR PROGRAMMING

### COMBINATION PROGRAM/VERIFY MODE (EPROM'S ONLY)



### VERIFY MODE (ROM/EPROM)



#### NOTES:

1. PROG MUST FLOAT IF EA IS LOW (i.e.,  $\neq 25V$ ), OR IF  $T0 = 5V$  FOR THE 8748. FOR THE 8048 PROG MUST ALWAYS FLOAT.
2.  $V_{EAH}$  FOR 8048 = 11.4V MIN., 12.6V MAX.
3. THE FOLLOWING CONDITIONS MUST BE MET:  
 $CS = TTL '1'$   
 $A0 = TTL '0'$   
 THIS CAN BE DONE USING 10K RESISTORS TO  $V_{CC}$ ,  $V_{SS}$  RESPECTIVELY.
4.  $X_1$  AND  $X_2$  DRIVEN BY 3 MHz CLOCK WILL GIVE  $5\mu sec t_{CY}$ . THIS IS GOOD FOR -8 PARTS AS WELL AS NON -8 PARTS.

The 8748 EPROM can be programmed by either of two Intel products:

1. PROMPT-48 Microcomputer Design Aid, or
2. Universal PROM Programmer (UPP-101 or UPP-102) peripheral of the Intellec® Development System with a UPP-848 Personality Card.

Note: See Appendix 2 for 8048 ROM ordering procedures. To minimize turnaround time on the first 25 pieces 8648 may be specified on the ROM order.



# NEW HIGH PERFORMANCE 8049/8039/8039-6 SINGLE COMPONENT 8-BIT MICROCOMPUTER

**PRELIMINARY**  
Model: This is not a final specification. Some parameters are subject to change.

- \*8049 Mask Programmable ROM
- \*8039 External ROM or EPROM
- \*New 11 MHz Operation

- 8-Bit CPU, ROM, RAM, I/O in Single Package
- Single 5V ± 10% Supply
- 1.36 μsec Cycle; All Instructions 1 or 2 Cycles
- Over 90 Instructions: 70% Single Byte
- Pin Compatible with 8048/8748
- 2K × 8 ROM
- 128 × 8 RAM
- 27 I/O Lines
- Interval Timer/Event Counter
- Easily Expandable Memory and I/O
- Compatible with MCS Memory and I/O
- Single Level Interrupt

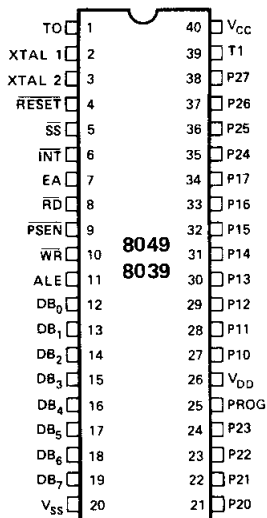
The Intel® 8049/8039/8039-6 is a totally self-sufficient 8-bit parallel computer fabricated on a single silicon chip using Intel's N-channel silicon gate MOS process.

The 8049 contains a 2K × 8 program memory, a 128 × 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to on board oscillator and clock circuits. For systems that require extra capability, the 8049 can be expanded using standard memories and MCS-80™/MCS-85™ peripherals. The 8039 is the equivalent to an 8049 without program memory. The 8039-6 is a lower speed (6MHz) version of the 8039.

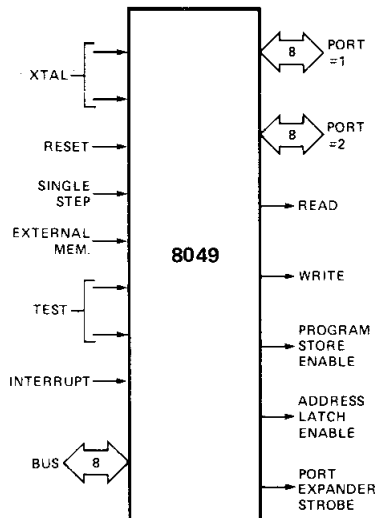
To reduce development problems to a minimum and provide maximum flexibility, two interchangeable pin-compatible versions of this single component microcomputer exist: the 8049 with factory-programmed mask ROM program memory for low-cost high volume production, and the 8039 without program memory for use with external program memories in prototype and preproduction systems.

This microprocessor is designed to be an efficient controller as well as an arithmetic processor. The 8049 has extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single byte instructions and no instructions over two bytes in length.

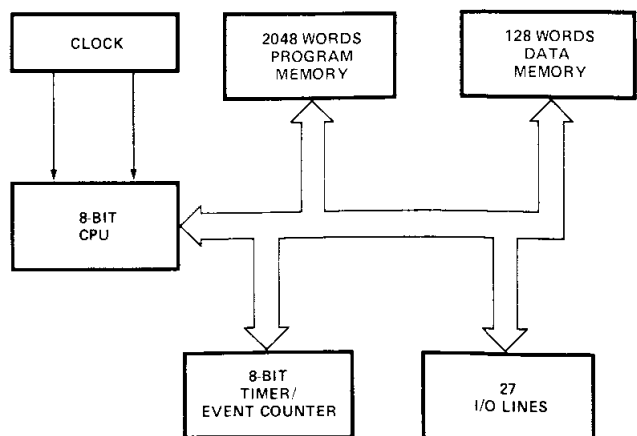
## PIN CONFIGURATION



## LOGIC SYMBOL



## BLOCK DIAGRAM



**ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias ..... 0°C to 70°C  
 Storage Temperature ..... -65°C to +150°C  
 Voltage on Any Pin With  
 Respect to Ground ..... -0.5V to +7V  
 Power Dissipation ..... 1.5 Watt

\*COMMENT: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

**D.C. AND OPERATING CHARACTERISTICS**  $T_A = 0^\circ\text{C to } 70^\circ\text{C}$ ,  $V_{CC} = V_{DD} = +5V \pm 10\%$ ,  $V_{SS} = 0V$ 

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.	Max.		
$V_{IL}$	Input Low Voltage	-0.5		0.8	V	
$V_{IH}$	Input High Voltage (All Except XTAL1, XTAL2, $\overline{\text{RESET}}$ )	2.0		$V_{CC}$	V	
$V_{IH1}$	Input High Voltage ( $\overline{\text{RESET}}$ , X1, X2)	3.8		$V_{CC}$	V	
$V_{OL}$	Output Low Voltage (BUS, $\overline{\text{RD}}$ , $\overline{\text{WR}}$ , $\overline{\text{PSEN}}$ , ALE)			0.45	V	$I_{OL} = 2.0\text{mA}$
$V_{OL1}$	Output Low Voltage (All Other Outputs Except PROG)			0.45	V	$I_{OL} = 1.6\text{mA}$
$V_{OL2}$	Output Low Voltage (PROG)			0.45	V	$I_{OL} = 1.0\text{mA}$
$V_{OH}$	Output High Voltage (BUS, $\overline{\text{RD}}$ , $\overline{\text{WR}}$ , $\overline{\text{PSEN}}$ , ALE)	2.4			V	$I_{OH} = -100\mu\text{A}$
$V_{OH1}$	Output High Voltage (All Other Outputs)	2.4			V	$I_{OH} = -50\mu\text{A}$
$I_{IL}$	Input Leakage Current (T1, $\overline{\text{INT}}$ )			$\pm 10$	$\mu\text{A}$	$V_{SS} \leq V_{IN} \leq V_{CC}$
$I_{OL}$	Output Leakage Current (Bus, T0) (High Impedance State)			$\pm 10$	$\mu\text{A}$	$V_{SS} + 0.45 \leq V_{IN} \leq V_{CC}$
$I_{DD}$	Power Down Supply Current		25	50	mA	$T_A = 25^\circ\text{C}$
$I_{DD} + I_{CC}$	Total Supply Current		100	170	mA	$T_A = 25^\circ\text{C}$

**A.C. CHARACTERISTICS**  $T_A = 0^\circ\text{C to } 70^\circ\text{C}$ ,  $V_{CC} = V_{DD} = +5V \pm 10\%$ ,  $V_{SS} = 0V$ 

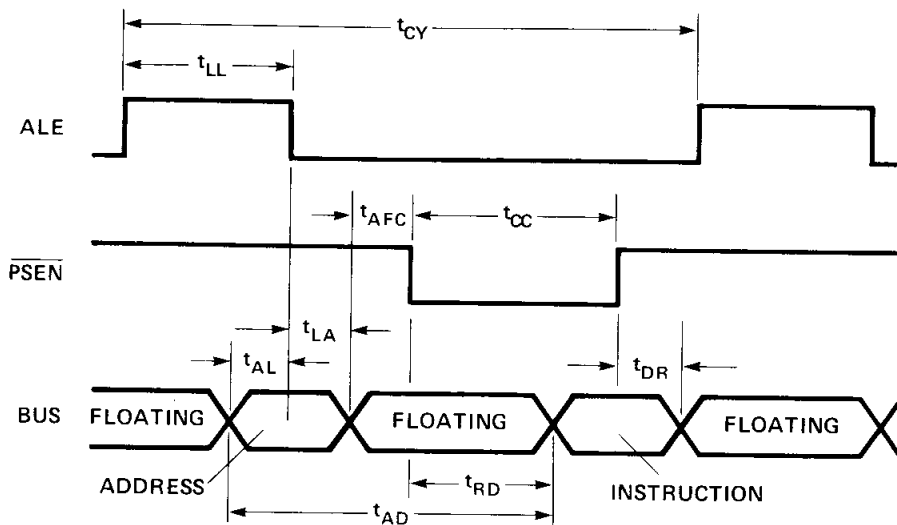
Symbol	Parameter	8049/8039 (Note 1)		8039-6		Unit	Conditions (Note 2)
		Min.	Max.	Min.	Max.		
$t_{LL}$	ALE Pulse Width	150		400		ns	
$t_{AL}$	Address Setup to ALE	70		150		ns	
$t_{LA}$	Address Hold from ALE	50		80		ns	
$t_{CC}$	Control Pulse Width (PSEN, RD, WR)	300		700		ns	
$t_{DW}$	Data Set-Up Before WR	250		500		ns	
$t_{WD}$	Data Hold After WR	40		120		ns	$C_L = 20\text{pF}$
$t_{CY}$	Cycle Time	1.36	15.0	2.5	15.0	$\mu\text{s}$	11MHz XTAL (6MHz XTAL for -6)
$t_{DR}$	Data Hold	0	100	0	200	ns	
$t_{RD}$	PSEN, RD to Data In		200		500	ns	
$t_{AW}$	Address Setup to WR	200		230		ns	
$t_{AD}$	Address Setup to Data In		400		950	ns	
$t_{AFC}$	Address Float to RD, PSEN	-10		0		ns	

Notes: 1. 8039-6 specifications are also valid for 8049/8039 operating at 6MHz.

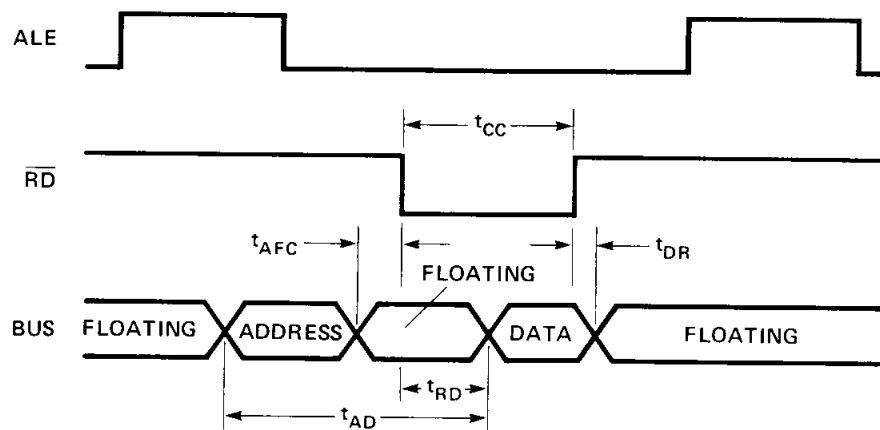
2. Control Outputs:  $C_L = 80\text{pF}$   
 BUS Outputs:  $C_L = 150\text{pF}$

## WAVEFORMS

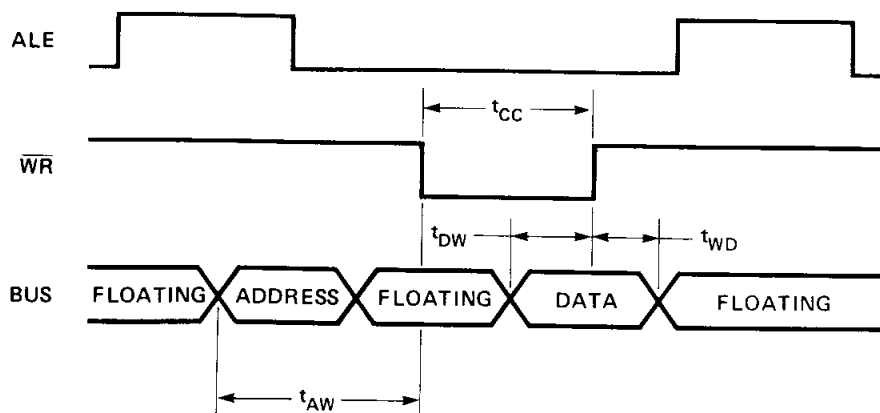
## INSTRUCTION FETCH FROM EXTERNAL PROGRAM MEMORY



## READ FROM EXTERNAL DATA MEMORY



## WRITE TO EXTERNAL DATA MEMORY



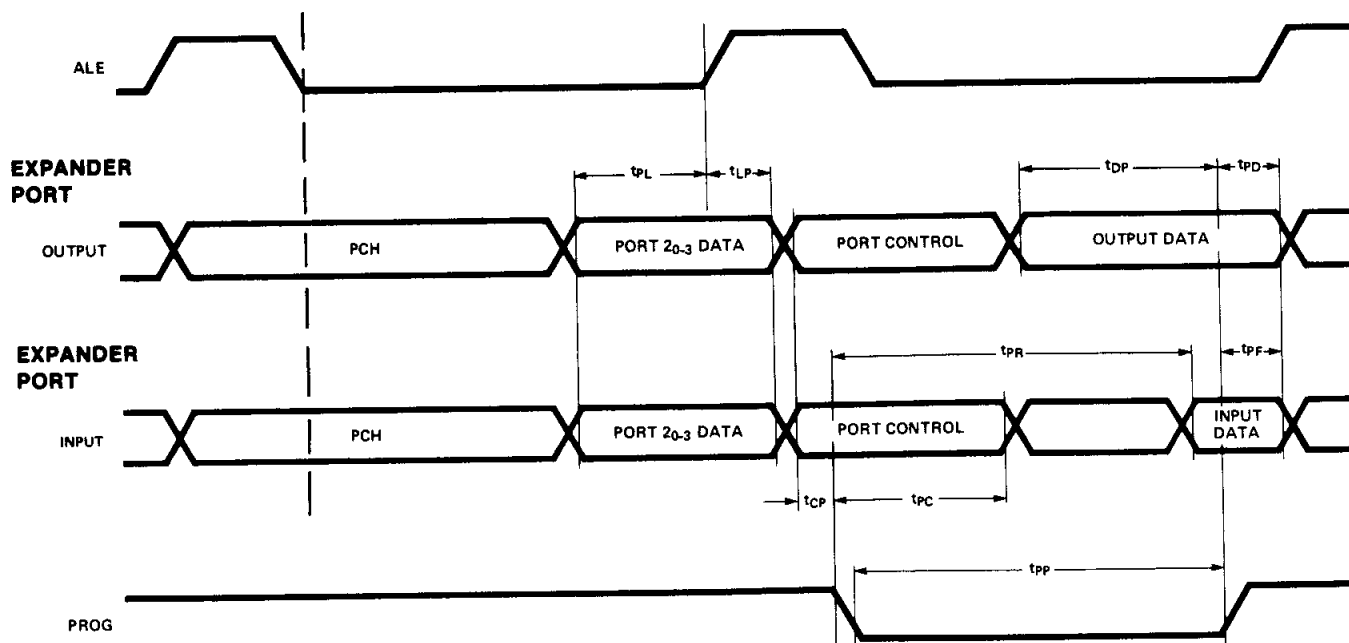
## A.C. CHARACTERISTICS

$T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 10\%$

Symbol	Parameter	8049/8039		8039-6		Unit	Conditions (Note 2)
		Min.	Max.	Min.	Max.		
$t_{CP}$	Port Control Setup Before Falling Edge of PROG	100		110		ns	
$t_{PC}$	Port Control Hold After Falling Edge of PROG	60		140		ns	
$t_{PR}$	PROG to Time P2 Input Must Be Valid		650		810	ns	
$t_{DP}$	Output Data Setup Time	200		220		ns	
$t_{PD}$	Output Data Hold Time	20		65		ns	
$t_{PF}$	Input Data Hold Time	0	150	0	150	ns	
$t_{PP}$	PROG Pulse Width	700		1510		ns	
$t_{PL}$	Port 2 I/O Data Setup	150		400		ns	
$t_{LP}$	Port 2 I/O Data Hold	20		150		ns	

## WAVEFORMS

### PORT 2 TIMING



## PIN DESCRIPTION

Designation	Pin #	Function	Designation	Pin #	Function
V <sub>SS</sub>	20	Circuit GND potential	$\overline{RD}$	8	Output strobe activated during a BUS read. Can be used to enable data onto the BUS from an external device.  Used as a Read Strobe to External Data Memory. (Active low)
V <sub>DD</sub>	26	+5V during operation. Low power standby pin.	$\overline{RESET}$	4	Input which is used to initialize the processor. Also used during verification, and power down. (Active low) (Non TTL V <sub>IH</sub> )
V <sub>CC</sub>	40	Main power supply; +5V during operation.	$\overline{WR}$	10	Output strobe during a BUS write. (Active low)  Used as write strobe to External Data Memory.
PROG	25	Output strobe for 8243 I/O expander.	ALE	11	Address Latch Enable. This signal occurs once during each cycle and is useful as a clock output.  The negative edge of ALE strobes address into external data and program memory.
P10-P17 Port 1	27-34	8-bit quasi-bidirectional port.	$\overline{PSEN}$	9	Program Store Enable. This output occurs only during a fetch to external program memory. (Active low)
P20-P27 Port 2	21-24 35-38	8-bit quasi-bidirectional port. P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243	$\overline{SS}$	5	Single step input can be used in conjunction with ALE to "single step" the processor through each instruction. (Active low)
D0-D7 BUS	12-19	True bidirectional port which can be written or read synchronously using the $\overline{RD}$ , $\overline{WR}$ strobes. The port can also be statically latched. Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, $\overline{RD}$ , and $\overline{WR}$ .	EA	7	External Access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing and program verification. (Active high)
T0	1	Input pin testable using the conditional transfer instructions JTO and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction.	XTAL1	2	One side of crystal input for internal oscillator. Also input for external source. (Not TTL Compatible)
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.	XTAL2	3	Other side of crystal input.
$\overline{INT}$	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low)			

## INSTRUCTION SET

	Mnemonic	Description	Bytes	Cycle		Mnemonic	Description	Bytes	Cycles	
Accumulator	ADD A, R	Add register to A	1	1	Subroutine	CALL	Jump to subroutine	2	2	
	ADD A, @R	Add data memory to A	1	1		RET	Return	1	2	
	ADD A, #data	Add immediate to A	2	2		RETR	Return and restore status	1	2	
	ADDC A, R	Add register with carry	1	1	Flags	CLR C	Clear Carry	1	1	
	ADDC A, @R	Add data memory with carry	1	1		CPL C	Complement Carry	1	1	
	ADDC A, #data	Add immediate with carry	2	2		CLR F0	Clear Flag 0	1	1	
	ANL A, R	And register to A	1	1		CPL F0	Complement Flag 0	1	1	
	ANL A, @R	And data memory to A	1	1		CLR F1	Clear Flag 1	1	1	
	ANL A, #data	And immediate to A	2	2		CPL F1	Complement Flag 1	1	1	
	ORL A, R	Or register to A	1	1		Data Moves	MOV A, R	Move register to A	1	1
	ORL A, @R	Or data memory to A	1	1	MOV A, @R		Move data memory to A	1	1	
	ORL A, #data	Or immediate to A	2	2	MOV A, #data		Move immediate to A	2	2	
	XRL A, R	Exclusive Or register to A	1	1	MOV R, A		Move A to register	1	1	
	XRL A, @R	Exclusive or data memory to A	1	1	MOV @R, A		Move A to data memory	1	1	
	XRL A, #data	Exclusive or immediate to A	2	2	MOV R, #data		Move immediate to register	2	2	
	INC A	Increment A	1	1	MOV @R, #data		Move immediate to data memory	2	2	
	DEC A	Decrement A	1	1	Move A, PSW		Move PSW to A	1	1	
	CLR A	Clear A	1	1	MOV PSW, A		Move A to PSW	1	1	
	CPL A	Complement A	1	1	XCH A, R		Exchange A and register	1	1	
	DA A	Decimal Adjust A	1	1	XCHA, @R	Exchange A and data memory	1	1		
	SWAP A	Swap nibbles of A	1	1	XCHD A, @R	Exchange nibble of A and register	1	1		
	RL A	Rotate A left	1	1	MOVX A, @R	Move external data memory to A	1	2		
	RLC A	Rotate A left through carry	1	1	MOVX @R, A	Move A to external data memory	1	2		
RR A	Rotate A right	1	1	MOVP A, @A	Move to A from current page	1	2			
RRC A	Rotate A right through carry	1	1	MOVP3 A, @A	Move to A from Page 3	1	2			
Input/Output	IN A, P	Input port to A	1	2	Timer/Counter	MOV A, T	Read Timer/Counter	1	1	
	OUTL P, A	Output A to port	1	2		MOV T, A	Load Timer/Counter	1	1	
	ANL P, #data	And immediate to port	2	2		STRT T	Start Timer	1	1	
	ORL P, #data	Or immediate to port	2	2		STRT CNT	Start Counter	1	1	
	INS A, BUS	Input BUS to A	1	2		STOP TCNT	Stop Timer/Counter	1	1	
	OUTL BUS, A	Output A to BUS	1	2		EN TCNTI	Enable Timer/Counter Interrupt	1	1	
	ANL BUS, #data	And immediate to BUS	2	2		DIS TCNTI	Disable Timer/Counter Interrupt	1	1	
	ORL BUS, #data	Or immediate to BUS	2	2		Control	EN I	Enable external interrupt	1	1
	MOVD A, P	Input Expander port to A	1	2	DIS I		Disable external interrupt	1	1	
	MOVD P, A	Output A to Expander port	1	2	SEL RB0		Select register bank 0	1	1	
ANLD P, A	And A to Expander port	1	2	SEL RB1	Select register bank 1		1	1		
ORLD P, A	Or A to Expander port	1	2	SEL MB0	Select memory bank 0		1	1		
Registers	INC R	Increment register	1	1	SEL MB1	Select memory bank 1	1	1		
	INC @R	Increment data memory	1	1	ENT0 CLK	Enable Clock output on T0	1	1		
	DEC R	Decrement register	1	1	NOP	NOP	No Operation	1	1	
Branch	JMP addr	Jump unconditional	2	2						
	JMPP @A	Jump indirect	1	2						
	DJNZ R, addr	Decrement register and skip	2	2						
	JC addr	Jump on Carry = 1	2	2						
	JNC addr	Jump on Carry = 0	2	2						
	JZ addr	Jump on A Zero	2	2						
	JNZ addr	Jump on A not Zero	2	2						
	JT0 addr	Jump on T0 = 1	2	2						
	JNT0 addr	Jump on T0 = 0	2	2						
	JT1 addr	Jump on T1 = 1	2	2						
	JNT1 addr	Jump on T1 = 0	2	2						
	JF0 addr	Jump on F0 = 1	2	2						
	JF1 addr	Jump on F1 = 1	2	2						
	JTF addr	Jump on timer flag	2	2						
	JNI addr	Jump on $\overline{INT} = 0$	2	2						
	JBb addr	Jump on Accumulator Bit	2	2						

Mnemonics copyright Intel Corporation 1976, 1977, 1978

# 8021

## SINGLE COMPONENT 8-BIT MICROCOMPUTER

- 8-Bit CPU, ROM, RAM, I/O in Single 28-Pin Package
- Single 5V Supply (+ 4.5V to 6.5V)
- 10  $\mu$ sec Cycle; All Instructions 1 or 2 Cycles
- Instructions —8748 Subset
- High Current Drive Capability—2 Pins
- 1K  $\times$  8 ROM
- 64  $\times$  8 RAM
- 21 I/O Lines
- Interval Timer/Event Counter
- Clock Generated With Single Resistor or Inductor
- Zero-Cross Detection Capability
- Easily Expandable I/O

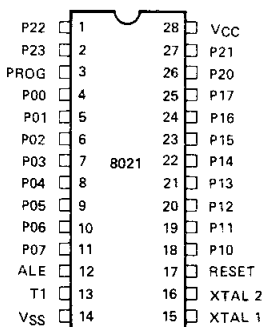
The Intel<sup>®</sup> 8021 is a totally self-sufficient 8-bit parallel computer fabricated on a single silicon chip using Intel's N-channel silicon gate MOS process. The features of the 8021 include a subset of the 8048 optimized for low cost, high volume applications, plus additional I/O flexibility and power.

The 8021 contains a 1K  $\times$  8 program memory, a 64  $\times$  8 data memory, 21 I/O lines, and an 8-bit timer/event counter, in addition to on-board oscillator and clock circuits. For systems that require extra I/O capability, the 8021 can be expanded using the 8243 or discrete logic.

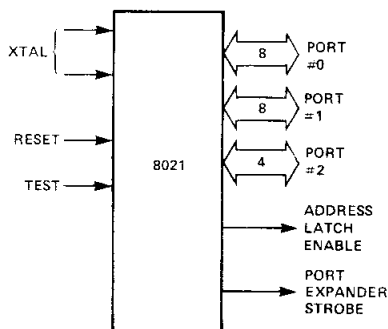
This microprocessor is designed to be an efficient controller as well as an arithmetic processor. The 8021 has bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single byte instructions and no instructions over two bytes in length.

To minimize development problems and maximize flexibility, an 8021 system can be easily designed using the 8021 emulation board, EMB-21. The EMB-21 contains a 40-pin socket which can accommodate either the 8748 shipped with the board or an ICE-48 plug. Also, the necessary discrete logic to reproduce the 8021's additional I/O features is included.

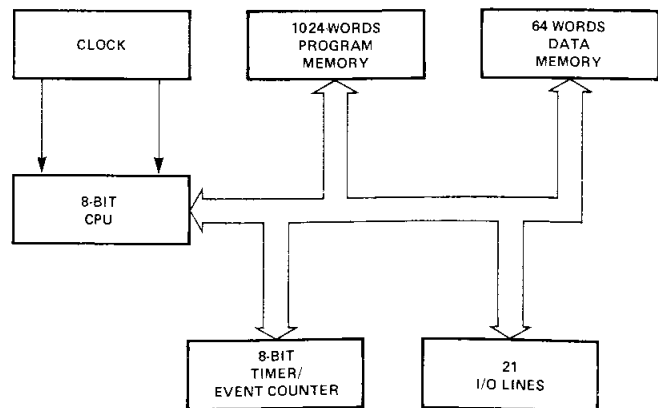
### PIN CONFIGURATION



### LOGIC SYMBOL



### BLOCK DIAGRAM







PRELIMINARY  
Notice: This is not a final specification. Some  
nomenclature items are subject to change.

# 8041/8741 UNIVERSAL PERIPHERAL INTERFACE 8-BIT MICROCOMPUTER

- Fully Compatible with MCS-80™, MCS-85™ and MCS-48™ Microprocessor Families
- Single Level Interrupt
- 8-Bit CPU plus ROM, RAM, I/O, Timer and Clock in a Single Package
- Single 5V Supply
- Alternative to Custom LSI
- Pin Compatible ROM and EPROM Versions
- 1K x 8 ROM/EPROM, 64 x 8 RAM, 18 Programmable I/O Pins
- Asynchronous Data Register for Interface to Master Processor
- Expandable I/O

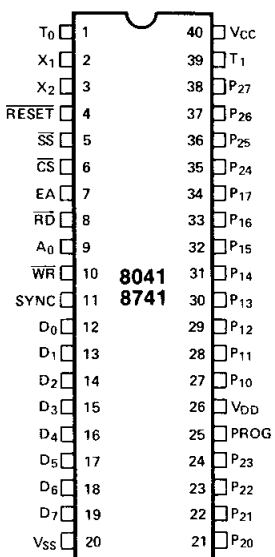
The Intel® 8041/8741 is a general purpose, programmable interface device designed for use with a variety of 8-bit microprocessor systems. It contains a low cost microcomputer with program memory, data memory, 8-bit CPU, I/O ports, timer/counter, and clock in a single 40-pin package. Interface registers are included to enable the UPI device to function as a peripheral controller in MCS-80™, MCS-85™, MCS-48™, and other 8-bit systems.

The UPI-41™ has 1K words of program memory and 64 words of data memory on-chip. To allow full user flexibility the program memory is available as ROM in the 8041 version or as UV-erasable EPROM in the 8741 version. The 8741 and the 8041 are fully pin compatible for easy transition from prototype to production level designs.

The device has two 8-bit, TTL compatible I/O ports and two test inputs. Individual port lines can function as either inputs or outputs under software control. I/O can be expanded with the 8243 device which is directly compatible and has 16 I/O lines. An 8-bit programmable timer/counter is included in the UPI device for generating timing sequences or counting external inputs. Additional UPI features include: single 5V supply, low power standby mode (in the 8041), single-step mode for debug (in the 8741), single level interrupt, and dual working register banks.

Because it's a complete microcomputer, the UPI provides more flexibility for the designer than conventional LSI interface devices. It is designed to be an efficient controller as well as an arithmetic processor. Applications include keyboard scanning, printer control, display multiplexing and similar functions which involve interfacing peripheral devices to microprocessor systems.

## PIN CONFIGURATION



## BLOCK DIAGRAM

