Course Code: 20EC0416

R20

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

NOTES

MICROPROCESSORS AND MICROCONTROLLERS

Course Code: 20EC0416

R20

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

UNIT –I
MICROPROCESSORS, MICROCOMPUTERS AND ASSEMBLY LANGUAGE

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

MICROPROCESSOR-BASED SYSTEM :

Organization of a Microprocessor-Based System Figure 1 shows a simplified but formal structure of a microprocessor-based system or a product. Since a microcomputer is one among many microprocessor-based systems. it will have the same structure as shown in Figure 1. It includes three components microprocessor, I/O (input/output), and memory (read/write memory and read-only memory). These components are organized around a common communication path called a bus. The entire group of components is also referred to as a system or a microcomputer system, and the components themselves are referred to as sub-systems. The microprocessor is one component of the microcomputer. On the other hand, the microcomputer is a complete computer similar to any other computer, except that CPU functions of the microcomputer are performed by the microprocessor. Similarly, the term peripheral is used for input/output devices. The various components of a microprocessor-based product or a microcomputer are shown in Figure 1 and their functions are described in this section

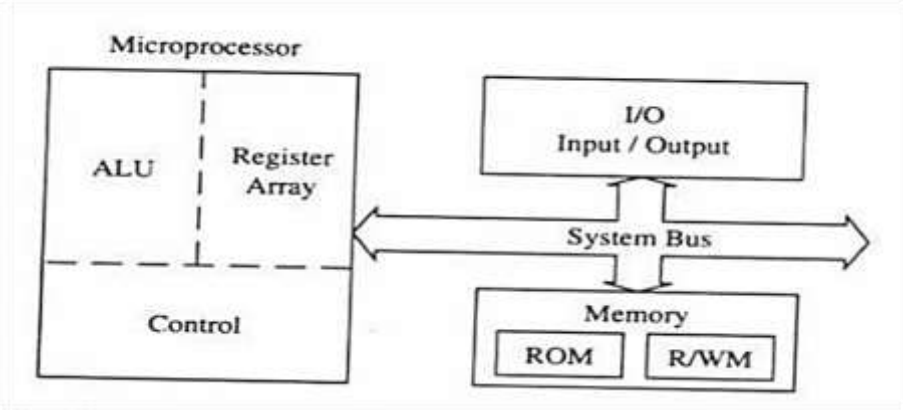


Fig 1 Microprocessor based System

MICROPROCESSOR

The microprocessor is a clock-driven semiconductor device consisting of electronic logic circuits manufactured by using either a large-scale integration (LSI) or very-large-scale integration (VLSI) technique. The microprocessor is capable of performing various computing functions and making decisions to change the sequence of program execution. In large computers, a CPU implemented on one or more circuit boards performs these computing functions. The microprocessor is in many ways similar to the CPU, but includes all the logic circuitry, including the control unit, on one chip. The microprocessor can be divided into three segments for the sake of clarity, as shown in Figure 1.: arithmetic/logic unit (ALU), register array, and control unit.

Arithmetic/Logic Unit : This is the area of the microprocessor where various computing functions are performed on data. The ALU unit performs such arithmetic operations as addition and subtraction, and such logic operations as AND, OR, and exclusive OR.

Siddartha Institute of Science and Technology::Puttur
(Autonomous)
(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

Register Array: This area of the microprocessor consists of various registers identified by letters such as B, C, D, E, H. and L. These registers are primarily used to store data temporarily during the execution of a program and are accessible to the user through instructions.

Control Unit: The control unit provides the necessary timing and control signals to all the operations in the microcomputer. It controls the flow of data between the micro- processor and memory and peripherals. To understand the control unit, let us consider one example. In early childhood, we learn a word, 'sit', and physical movememnts needed for the action are embedded in our brain. When we hear the word 'sit' our brain motions activates a series of actions for our muscles and bones and we sit down. In this analogy. the word "sit" is like as instruction in a microprocessor, and actions initiated by our brain are like microprograms. The bit patterns required to initiate these microprogram operations are given to the programmer in the form of the instruction set of the microprocessor. The programmer selects appropriate bit patterns from the set for a given task and enters them sequentially in memory through an input device. When the CPU reads these bit patterns one at a time, it initiates appropriate microprograms through the control unit, and performs the task specified in the instructions.

MEMORY: Memory stores such binary information as instructions and data, and provides that information to the microprocessor whenever necessary. To execute programs, the microprocessor reads instructions and data from memory and performs the computing operations in its ALU section. Results are either transferred to the output section for display or stored in memory for later use. The memory block shown in Figure 1. has two sections: Read-Only memory (ROM) and Read/Write memory (R/WM), popularly known as Random-Access memory (RAM).

The ROM is used to store programs that do not need alterations. The monitor program of a single-board microcomputer is generally stored in the ROM. This program interprets the information entered through a keyboard and provides equivalent binary digits to the microprocessor. Programs stored in the ROM can only be read; they cannot be altered

The Read/Write memory (R/WM) is also known as user memory. It is used to store user programs and data. In single-board microcomputers, the monitor program monitors the Hex keys and stores those instructions and data in the R/W memory. The information stored in this memory can be easily read and altered.

I/O (INPUT/OUTPUT) or Peripherals

The component of a microprocessor-based system is I/O (input/output); it communicates with the outside world. I/O includes two types of devices: input and output; these I/O devices are also known as peripherals.

The input devices such as a keyboard, switches, and an analog-to-digital (A/D) converter transfer binary information (data and instructions) from the outside world to the microprocessor. Typically, a hexadecimal keyboard or an ASCII keyboard as an input device. The hexadecimal (Hex) keyboard has 16 data keys (0 to 9 and A to F) and some additional function keys to perform such operations as storing data and executing programs. The ASCII keyboard is similar to a typewriter keyboard, and it is used to enter programs in an English-like language.

Siddartha Institute of Science and Technology::Puttur
(Autonomous)
(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

The output devices transfer data from the microprocessor to the outside world. They include devices such as light emitting diodes (LEDs), a cathode-ray tube (CRT) or video screen, a printer, X-Y plotter, a magnetic tape, and digital-to-analog (D/A) converter.

SYSTEM BUS: The system bus is a communication path between the microprocessor and peripherals; it nothing bit a group of wires to carry bits. In fact, there are several buses in the system. All peripherals (and memory) share the same bus; however, the microprocesor communicates with only one peripheral at a time. The timing is provided by the control unit of the microprocessor.

.HOW DOES THE MICROPROCESSOR WORK?

To understand the Mircoprocessor working, let us assume that a program and data are already entered in the R/W memory. The program includes binary instructions to add given data and to display the answer at the seven-segment LEDs. When the microprocessor is given a command to execute the program, it reads and executes one instruction at a time and finally sends the result to the seven-segment LEDs for display

This process of program execution can best be described by comparing it to the process of assembling a radio kit. The instructions for assembling the radio are printed in a sequence on a sheet of paper. One reads the first instruction, then picks up the necessary components of the radio and performs the task. The sequence of the process is read, interpret, and perform. The microprocessor works the same way. The instructions restored sequentially in the memory. The microprocessor fetches the instruction from its memory sheet, decodes it, and executes that instruction. The sequence of fetch. decode, and execute is continued until the microprocessor comes across an instruction to stops.

During the entire process, the microprocessor uses the system bus to fetch the bi nary instructions and data from the memory. It uses registers from the register section to store data temporarily, and it performs the computing function in the ALU section. Finally, it sends out the result in binary, using the same bus lines, to the seven-segment LEDs.

MICROPROCESSOR INSTRUCTION SET AND COMPUTER LANGUAGES

Microprocessors recognize and operate in binary numbers. However, each microprocessor has its own binary words, meanings, and language. The words are formed by combining a number of bits for a given machine. The word (or word length) is defined as the number of hits the microprocessor recognizes and processes at a time. The word length ranges from four bits for small, microprocessor-based systems to 64 bits for high-speed large computers. Another term commonly used to express word length is byte. *A byte is defined as a group of eight bits*. For example, a 16-bit microprocessor has a word length equal to two bytes. *The term nibble, which stands for a group of four bits,* is found also in popular computer magazines and books. A byte has two nibbles.

Each machine has its own set of instructions based on the design of its CPU or of its Microprocessor. To communicate with the computer one give instructions in binary Language.

Different computer languages:

A programming language is a notation designed to connect instructions to a machine or a computer. Programming languages are mainly used to control the performance of a machine or to express

Siddartha Institute of Science and Technology::Puttur
(Autonomous)
(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

algorithms

Generally there are three types of computer languages

I. Machine Level Language
II. Assembly Level language
III. High Level Language

I.) **Machine Level Language:** Machine language is the language understood by a computer. It is very difficult to understand, but it is the only thing that the computer can work with. All programs and programming languages eventually generate or run programs in machine language. Machine language is made up of instructions and data that are all binary numbers.

Machine language is normally displayed in hexadecimal form so that it is a little bit easier to read.

The number of bits in a word for a given machine is fixed, and words are formed through various combinations of these bits For example, a machine with a word length of eight hits can have 256 (2) combinations of eight bits-thus a language of 256 words However, not all of these words need to be used in the machine. The microprocessor design engineer selects combinations of bit patterns and gives a specific meaning to each combination by using electronic logic circuits; this is called an instruction. Instructions are made up of one word or several words. The set of instructions designed into the ma chine makes up its machine language-a binary language, composed of 0's and I's -that is specific to each computer.

8085 Machine Language

The 8085 is a microprocessor with 8-bit word length: its instruction set (or language) is designed by using various combinations of these eight bits. The 8085 is an improved version of the earlier processor 8080A. An instruction is a binary pattern entered through an input device in memory to command the microprocessor to perform that specific function.

For example:

0011 1100 is an instruction that increments the number in the register called the accumulator by one.

1000 0000 is an instruction that adds the number in the register called B to the number in the accumulator, and keeps the sum in the accumulator.

The 8085 microprocessor has 246 such bit patterns, amounting to 74 different instructions for performing various operations. These 74 different instructions are called its instruction set. This binary language with a predetermined instruction set is called the 8085 machine language.

Because it is tedious and error-inducive for people to recognize and write instructions in binary language, these instructions are, for convenience, written in hexadecimal code and entered in a single-board microcomputer by using Hex keys. For example, the binary instruction 0011 1100 (mentioned previously) is equivalent to 3C in hexadecimal. This instruction can be entered in a single-board microcomputer system with a Hex key- board by pressing two keys: 3 and C. The monitor program of the system translates these keys into their equivalent binary pattern

II)Assembly level language: Assembly language is almost the same as machine language, except that the instructions, variables and addresses have names instead of just hex numbers. it acts as the intermediate language between machine language and high-level programming

languages.

8085 Assembly Language: Even though the instructions can be written in hexadecimal code, it is still difficult to un derstand a program written in hexadecimal numbers. Therefore, each manufacturer of a microprocessor has devised a symbolic code for each instruction, called a mnemonic. (The word mnemonic is based on the Greek word meaning mindful, that is, a memory aid) The mnemonic for a particular instruction consists of letters that suggest the opera- tion to be performed by that instruction.

For example, the binary code 0011 1100 (3C) or 3CH in hexadecimal) of the 8085 microprocessor is represented by the mnemonic INR A.

INR A: INR stands for increment, and A represents the accumulator. This symbol suggests the operation of incrementing the accumulator contents by one,

Similarly, the binary code 1000 0000 (80, or 80H) is represented as
ADD B: ADD stands for addition, and B represents the contents in register B. This symbol suggests the addition of the contents in register B and the contents in the accumulator
Although these symbols do not specify the complete operations, they suggest its significant part. The complete description of each instruction must be supplied by the manufacturer. The complete set of 8085 mnemonics is called the 8085 assembly language, and a program written in these mnemonics is called an assembly language program.
An assembly language program written for one microprocessor is not transferable to a computer with another microprocessor unless the two microprocessors are compatible in their machine codes.

Machine language and assembly language are microprocessor-specific and are both considered low-level languages. The machine language is in binary, and the assembly language is in English-like words; however, the microprocessor understands only the binary. The mnemonics can be written by hand on paper and communicated manually in hexadecimal code, called hand assembly, Similarly, the mnemonics can be written electronically on a com per nga program called an Editor in the ASCII code and mastered into binary code by using the program called an assembler.

III)High Level Language

In comparison to machine language, assembly language is easier to write and use;
A high-level language (HLL) is a programming language such as C, FORTRAN, or Pascal that enables a programmer to write
programs that are less dependent on machine .
e.g. High level program

```
class Triangle {
…
float surface()
return b*h/2;
}
```

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

ASSEMBLE LANGUAGE PROGRAM PROCEDURE:

A program is a sequence of instructions written to tell a computer to perform a specific function. The instructions are selected from the instruction set of the microprocessor. To write a program, divide a given problem in small steps in terms of the operations the 8085 can perform, then translate these steps into instructions .

The Assembly Process is shown in block diagram Figure 2

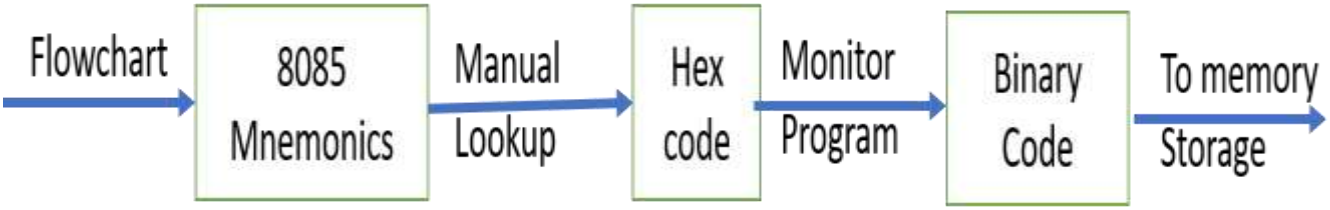


Figure 2 : Manual Assembly Process

To explain this procedure consider example of writing a simple program of adding two numbers in the 8085 language is illustrated below:

Program: Adding Two Hexadecimal Numbers

PROBLEM STATEMENT :We instructions to load the two hexadecimal numbers 32H and 48H in registers A and B, respectively, Add the numbers, and display the sum at the LED output port PORT 1 Even though this is a simple problem, it is necessary to divide the problem into small seps to examine the process of writing programs. The wording of the problem provides sufficient clues for the necessary steps. They are as follows:

PROBLEM ANALYSIS

1. Load the numbers in the registers.
2. Add the numbers
3. Display the sum at the output port PORTI

FLOWCHART

The steps listed in the problem analysis and the sequence can be represented in a block diagram, called a flowchart. Figure.3 shows such a flowchart representing the above steps.

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

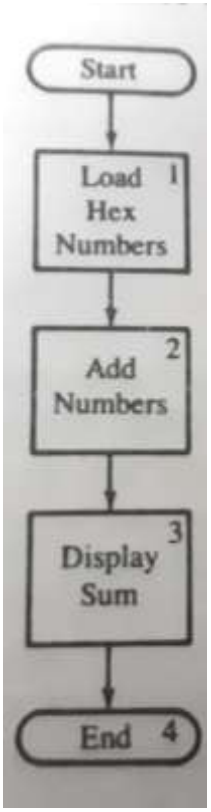


Figure.3: Flowchart

ASSEMBLY LANGUAGE PROGRAM

To write an assembly language program: we need to translate the blocks shown in the flowchart into 8085 operations and then, subsequently, into mnemonics. By examining the blocks, we can classify them into three types of operations: Blocks 1 and 3 are copy operations. Block 2 is an arithmetic operation and Block 4 is a machine control operation. To translate these steps into assembly and machine language, each block are rewritten with mnemonics and comments.

Block1 : MVI A, 32H Load register A with 32H
MVI B, 48H Load register B with 48 H
Block 2: ADD B Add two bytes and save the sum in A
Block 3: OUT 01H Display Accumulator contents at port 01H
Block 4: HALT End

FROM ASSEMBLY LANGUAGE TO HEX CODE

To convert the mnemonics into Hex code, we need to look up the code in the 8085 in- struction set; this is called either manual or hand assembly.

Mnemonics	Hex Code	
MVI A.32H	3E 32	2-byte instruction
MVI B.48H	06 48	2-byte instruction
ADD B	80	1-byte instruction
OUT 01	D3	2-byte instruction

Siddartha Institute of Science and Technology::Puttur
(Autonomous)
(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

	01	
HALT	76	1-byte instruction

STORING IN MEMORY AND CONVERTING FROM HEX CODE TO BINARY CODE

To store the program in R/W memory of a single-board microcomputer and display the output, we need to know the memory addresses and the output port address. Let us assume that R/W memory ranges from 2000H to 20FFH. and the system has an LED output port with the address 01H. Now, to enter the program:

1. Reset the system by pushing the RESET key.

2. Enter the first memory address using Hex keys where the program should be stored. Let us assume it is 2000H.
3. Enter each machine code by pushing Hex keys. For example, to enter the first machine code, push the 3, E, and STORE keys. (The STORE key may be labeled differently indifferent systems.) When you push the STORE key, the program will store the machine code in memory location 2000H and upgrade the memory address to 2001H.
4. Repeat Step 3 until the last machine code, 76H.
5. Reset the system.

Now the question is: How does the Hex code get converted into binary code? The an- swer lies with the Monitor program stored in Read-Only memory (or EPROM) of the micro-

EXECUTING THE PROGRAM

To execute the program, we need to tell the microprocessor where the program begins by entering the memory address 2000H. Now, we can push the Execute key (or the key with a similar label) to begin the execution. As soon as the Execute function key is pushed, the microprocessor loads 2000H in the program counter, and the program control is trans- ferred from the Monitor program to our program.

The microprocessor begins to read one machine code at a time, and when it fetches the complete instruction, it executes that instruction. For example, it will fetch the ma- chine codes stored in memory locations 2000H and 2001H and execute the instruction MVI A.32H; thus it will load 32H in register A. The ADD instruction will add the two numbers, and the OUT instruction will display the answer 7AH (32H+ 48H=7AH) at the LED port. It continues to execute instructions until it fetches the HLT instruction.

CLASSIFICATION OF COMPUTERS FROM LARGE COMPUTERS TO SINGLE CHIP MICROCONTROLLERS:

Computers are classified into mainly three types

i)Large Computers ii). Medium size Computers iii). Microcomputers

i)Large computers

Large computers are multipurpose, multiuser, multitasking computers. These are used in computer scientific and engineering calculations, handle large records for large corporations and government agencies. Large computers are two types

1) Main frames and 2) Super computers

Main frames

A **mainframe computer** (as storage for large database and serve as maximum number of users

simultaneously) is a computer used primarily by large organizations for critical applications, bulk data processing such as the census and industry and consumer statistics, enterprise resource planning, and large-scale transaction processing.

A mainframe computer is larger and has more processing power than other classes of computers, such as minicomputers, servers, workstations, and personal computers.

Most of the large-scale computer- system architectures were established in the 1960s, but they continue to evolve. Mainframe computers are often used as servers

Super computers

Super computers (large and complex mathematical Computations)are fastest high-performance systems available at any given time. Such computers have been used primarily for scientific and engineering work requiring exceedingly high- speed computations.

Common applications for supercomputers include testing mathematical models for complex physical phenomena or designs, such as climate and weather, evolution of the cosmos, nuclear weapons and reactors, new chemical compounds (especially for pharmaceutical purposes), and cryptology.

ii)Medium size computers

(Minicomputers support more than 100 users at a time with multi terminal and time sharing)

Medium-size computer systems provide faster operating speeds and larger storage capacities than minicomputer systems. They can support a large number of high-speed input/output devices and several disk drives can be used to provide online access to large data files as required for direct access processing and their operating systems also support both multiprogramming and virtual storage.

This allows the running of variety of programs concurrently. A medium-size computer can support a management information system and can therefore serve the needs of a large bank, insurance company or University

iii)Microcomputers

A microcomputer is a complete computer on a small scale, designed for use by one person at a time. microcomputer is now primarily called a personal computer (PC), or a device based on a single-chip microprocessor. Common microcomputers include laptops and desktops.

The 4-bit and 8-bit microprocessors became available in the mid-1970s, and initial appli cations were primarily in the areas of machine control and instrumentation. As the price of the microprocessors and memory began to decline, the applications mushroomed in al- most all areas, such as video games, word processing, and small-business applications Early microcomputers were designed around 8-bit microprocessors. Since then, 16-, 32- and 64-bit microprocessors, such as the Intel 8086/88, 80386/486, Pentium, Pro-Pentium, Pentium 4. Motorola 68000, and the Power PC series have been introduced, and recent microcomputers are being designed around these microprocessors.

1) Present-day micro- computers can be classified in four groups:
2) personal (or business) computers (PC),
3) work- stations,
4) single-board, and
5) single-chip microcomputers (microcontrollers).

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

PERSONAL COMPUTERS (PC)

These microcomputers are single-user systems and are used for a variety of purposes, such as payroll, business accounts, word processing, legal and medical record keeping, personal finance, accessing Internet resources (e-mail. Web search, newsgroup), and instruction. They are also known as personal computers (PC) or desktop computers. Typically, the price ranges from $500 to $5000 for a single-user system. Examples include such microcomput- ers as the IBM Personal Computer (Aptiva series), the Hewlett-Packard Pavilion series, and the Apple Macintosh series. At the low end of the microcomputer spectrum, a typical configuration includes a 32- bit (or 64-bit) microprocessor, 32 to 256 MB (megabytes) of system memory, a video screen (monitor), a 3" high-density floppy disk, a hard disk with storage capacity of more than 10 gigabytes, a CD-ROM, and a Zip disk. The floppy disk is a magnetic medium similar to a cassette tape except that it is round in shape, like a record. Information recorded on these disks can be accessed randomly using disk drives. Conversely, information stored on a cassette tape is accessed serially. In order to read information at the end of the tape, the user must run the entire tape through the machine. The hard disk is similar to the floppy disk except that the magnetic material is coated on a rigid aluminum base that is enclosed in a sealed container and permanently installed in a microcomputer. The hard disk and the floppy disks are used to store programs semipermanently, i.e., the binary information does not disappear when the power is turned off. However, the microprocessor does not have direct access to this information; it must copy this information (programs) into system memory to modify or ex- these programs. The hard disk has a large storage capacity; therefore, large and frequently used programs such as compilers, interpreters, system programs, and application programs are stored on this disk. The floppy disk is generally used for user programs and to make backup copies.

The microcomputers are further classified according to their size, weight, and portability. They are called laptop and notebook.

The laptop computer is a portable microcom- puter that has a flat screen, a hard disk, and a 3" floppy disk, and usually weighs around ten pounds. These computers can be battery operated or use AC power and are carried eas- ily from place to place. These are called laptop (instead of desktop) because the size is small enough to place them in one's lap (if necessary). The notebook computer is a portable microcomputer of a notebook size 11"x2") and weighs around five pounds. A mi- crocomputer smaller than the notebook computers, called a subnotebook, is also available.

WORKSTATIONS

These are high-performance cousins of the personal computers. They are used in engi- neering and scientific applications such as computer-aided design (CAD), computer-aided engineering (CAE), and computer-aided manufacturing (CAM). They generally include system memory and storage (hard disk) memory in gigabytes, and a high-resolution screen

The workstations are designed around RISC (reduced instruction set computing) processors. The RISC processors tend to be faster and more efficient than the processors used in personal computers. Some of the workstations have better performance than that of the low-end large computers.

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

SINGLE-BOARD MICROCOMPUTERS

These microcomputers are primarily used in college laboratories and industries for instructional purposes or to evaluate the performance of a given microprocessor. They can also be part of some larger systems. Typically, these microcomputers include an 8 or 16-bit microprocessor, from 256 bytes to 8K bytes of ser memory, a Hes keyboard, and seven-segment LEDs as display. The interaction between the microprocessor, memory, and I/Os in these small systems is looked after by a program called a system monitor program, which is generally small in size, stored in less than 2K bytes of ROM. When a single-board microcomputer is turned on, the monitor program is in charge of the system; it monitors the keyboard inputs, interprets these keys, stores programs in memory, sends system displays to the LEDs, and enables the execution of the user programs. The function of the monitor program in a small system is similar to that of the operating system in a large system. The prices of these single-board computers range from $100 to $800, the average price being around $300.

Examples of these computers include such systems as the Intel SDK 85. SDK 86. and the EMAC Primer . These are generally used to write and execute as- sembly language programs and to perform interfacing experiments.

SINGLE-CHIP MICROCOMPUTERS (MICROCONTROLLERS)

These microcomputers are designed on a single chip, which typically includes a micro- processor, 256 bytes of R/W memory, from IK to 8K bytes of ROM, and several signal limes to connect I/Os. These are complete microcomputers on a chip: they are also known as microcontrollers. They are used primarily for such functions as controlling appliances and traffic lights. Typical examples of these microcomputers include such chips as the Zilog 28. Intel MCS 51 series, Motorola 68HC11, and the Microchip Technology PIC family.

APPLICATION: MICROPROCESSOR CONTROLLED TEMPERATURE SYSTEM

A Temperature Controlled System is a type of control system that automatically controls the temperature of an object or an area.This system sets two point upper limit point and lower limit point. This system is excepted to read the temperature in room ,display the temperature at LCD panel turn on a fan if the temperature is above upper limit point and turn on a heater if the temperature below lower limit point. This system has input and output ports for the connection of input and output devices like sensor , fan, heater and LCD etc.

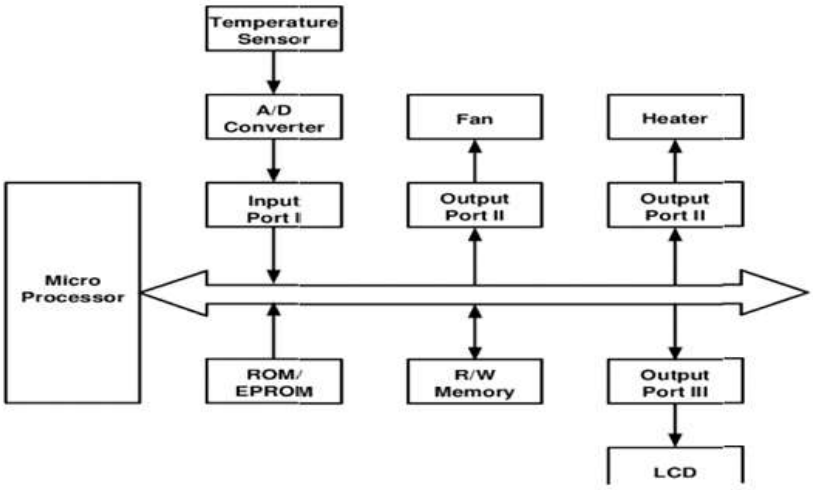


Figure 4 block diagram of microprocessor controlled temperature system

Siddartha Institute of Science and Technology::Puttur
(Autonomous)
(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

System hardware
1.Microprocess
2.Memory
3.Input devices
4.Output devices

MICROPROCESSOR

Figure 1.8 shows an 8085 processor with a system bus; we will expand this bus in vari- ous buses in the following chapter. The processor will read the binary instructions from memory and execute those instructions continuously. It will read the temperature, display it at the LCD display panel, and turn on/off the fan and the heater based on the temperature.

MEMORY

The system includes two types of memory. ROM (read-only memory) will be used to store the program, called the monitor program, that is responsible for providing the nec- essary instructions to the processor to monitor the system. This will be a permanent pro- gram stored in ROM and will not be altered. The R/W (read-write) memory is needed for temporary storage of data; the need for this memory will be explained in a later chapter.

INPUT

In this system, we need a device that can translate temperature (measurement of heat) into an equivalent electrical signal; a device that translates one form of energy into another form is called transducer. For example, a microphone is a transducer that converts sound electrical signal, and a thermocouple is a transducer that converts heat into an electrical signals. A temperature sensor is a three terminal semiconductor electronic device ta chip similar way that generates a stage signal that is proportional to the temperature. However this is an analog signal and our processor is capable of handling only hinary Therefore, this signal must be converted into digital bits. The analog-to-digital A converter performs that function. The A/D converter, shown in Figure, is also electronics semiconductor chip that converts an input analog signal into the equivalent binary output signals. In microprocessor-based systems, devices that provide binary input (data) are connected to the processor using devices such as buffers called input pes. In our system, this A/D converter is an input port, and it will be assigned a binary number called an address. The microprocessor reads this digital signal from the input port

OUTPUT

Figure shows three output devices: fan, heater and liquid crystal display (LCD). These devices are connected to the processor using latches called output ports.

Fan This is an output device, identified as Portl, that is turned on by the processor when the temperature reaches a set higher limit.

Heater This is also an output device, identified as Port2, that is turned on by the processor when the temperature reaches a set lower limit.

Liquid Crystal Display (LCD) This display is made of crystal material placed between two plates in the form of a dot matrix or segments. It can display letters, decimal digits, or graphic characters. The LCD in Figure 1.8 will be used to display temperatures,

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

MICROPROCESSOR ARCHITECTURE AND ITS OPERATION

The microprocessor is a programmable digital device. designed with registers, flip-flops, and timing elements. The microprocessor has a set of instructions, designed internally, to manipulate data and communicate with peripherals. This process of data manipulation and communication is determined by the logic design of the microprocessor, called the architecture. The microprocessor can be programmed to perform functions on given data by selecting necessary instructions from its set. These instructions are given to the microprocessor by writing them into its memory. Writing (or entering) instructions and data is done through an input device such as a keyboard. The microprocessor reads or transfer one instruction at a time, matches it with its instruction set, and performs the data manipulation indicated by the instruction. The result can be stored in memory or sent to such output devices as LEDs or a CRT terminal. In addition, the microprocessor can respond to external signals. It can be interrupted, reset, or asked to wait to synchronize with slower peripherals. All the various functions performed by the microprocessor can be classified in three general categories:

1)Microprocessor-initiated operations
2) Internal operations
3) Peripheral (or externally initiated) operations

To perform these functions, the microprocessor requires a group of logic circuits and a set of signals called control signals. However, early processors did not have the necessary circuitry on one chip: the complete units were made up of more than one chip. Therefore, the term microprocessing unit (MPU) is defined here as a group of devices that can perform these functions with the necessary set of control signals. This term is similar to the term central processing unit (CPU). However, later microprocessors include most of the necessary circuitry to perform these operations on a single chip. Therefore, the terms MPU and microprocessor often are used synonymously. The microprocessor functions listed above are explained here in relation to the 8085 MPU but without the details of the MPUs. However, the general concepts discussed here are applicable to any microprocessor. The devices necessary to make up the 8085 MPUs will be discussed in the next chapter.

1) Microprocessor-Initiated Operations and 8085 Organization

The MPU performs primarily four operations

1. Memory Read: Reads data (or instructions) from memory

2. Memory Write: Writes data (or instructions) into memory.

3. I/O Read: Accepts data from input devices.

4. I/O Write: Sends data to output devices.

All these operations are part of the communication process between the MPU and peripheral devices (including memory). To communicate with a peripheral (or a memory location), the MPU needs to perform the following steps:

Siddartha Institute of Science and Technology::Puttur
(Autonomous)
(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

Step 1: Identify the peripheral or the memory location (with its address).
Step 2: Transfer binary information (data and instructions).
Step 3: Provide timing or synchronization signals.

The 8085 MPU performs these functions using three sets of communication lines called buses: the address bus, the data bus, and the control bus (Figure.5). These buses are shown as one group, called the system bus.

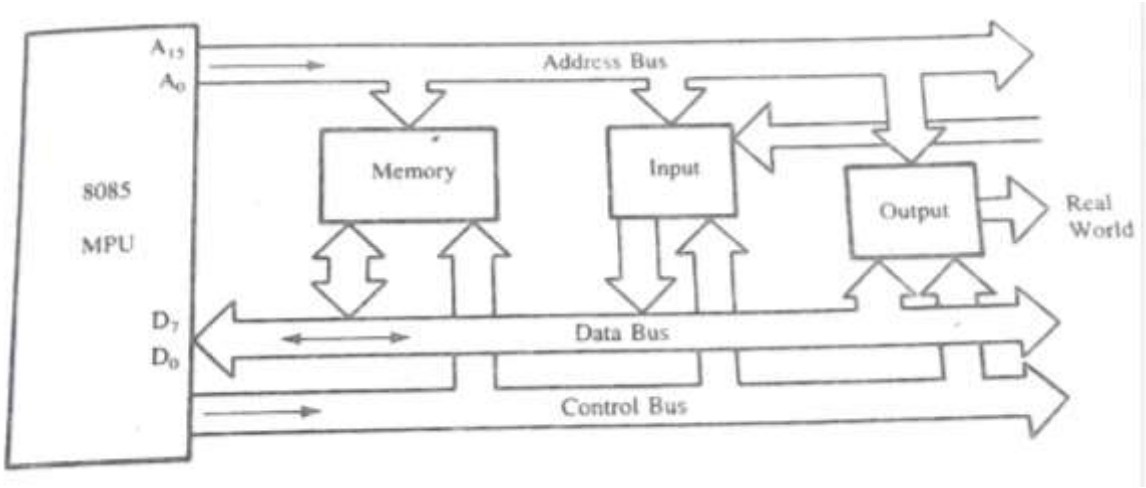


Fig.5 The 8085 Bus Architecture

ADDRESS BUS

The address bus is a group of 16 lines generally identified as A15 to A0. The address bus is unidirectional: hits flow in one direction-from the MPU to peripheral devices. The MPU uses the address bus to perform the first function: identifying a peripheral or a memory location (Step 1). In a computer system, each peripheral or memory location is identified by a binary called an address, and the address bus is used to carry a 16-bit address. This is similar to the postal address of a house A house can be identified by various number schemes For example, the forty-fifth house in a lane can be identified by the two-digit number 45 or the fine-digit number 0045 The two-digit numbering scheme can identify only a hundred from 0 to 99 On the other hand, the four-digit scheme can identify ten thousand fb 0000 to 9999 Similarly, the number of address lines of the MPU determines its its different memory locations for peripherals). The 8085 MPU with its 16 address lines is capable of addressing 265.536 (generally known as 64K) memory locations. 1K memory is determined by rounding off 1024 to the nearest that similarly 65.536 is rounded off to 64,000 as a multiple of IK.
Most 8-bit microprocessor have 16 address lines. This may explain why microcomputer systems based on 8-bit microprocessors have 64K memory. However, not every mi computer system has 64K memory. In fact, most single-board microcomputers have less dan 4K of memory even if the MPU is capable of addressing 64K memory. The number of address lines is arbitrary, it is determined by the designer of a microprocessor based on such considerations as availability of pins and intended applications of the processor. For example, the Intel 8088 processor has 20 and the Pentium processor has 32 address lines.

DATA BUS
The data bus is a group of eight lines used for data flow (Figure 5). These lines are bidirectional

Siddartha Institute of Science and Technology::Puttur
(Autonomous)
(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

data flow in both directions between the MPU and memory and peripheral devices. The MPU uses the data bus to perform the second function: transferring binary information (Step 2).
The eight data lines enable the MPU to manipulate 8 bit data ranging from 00 to FF (2^8=256 numbers). The largest number that can appear on the data bus is 11111111 (255_{10}) The 8085 is known as an 8-bit microprocessor Microprocessors such as the Intel 8086, Zilog Z8000, and Motorola 68000 have 16 data lines; thus they are known as 16- bit microprocessors. The Intel 80386/486 have 32 data lines; thus they are classified as 32-bit microprocessors

CONTROL BUS

The control bus is comprised of various single lines that carry synchronization signals. The MPU uses such lines to perform the third function: providing timing signals (Step 3). The term bus, in relation to the control signals, is somewhat confusing. These are not groups of lines like address or data buses, but individual lines that provide a pulse to indicate an MPU operation. The MPU generates specific control signals for every operation (such as Memory Read or I/O Write) it performs. These signals are used to identify a device type with which the MPU intends to communicate.

To communicate with a memory-for example, to read an instruction from a memory location the MPU places the 16-bit address on the address bus (Figure 6). The address on the bus is decoded by an external logic circuit, which will be explained later, and the memory location is identified. The MPU sends a pulse called Memory Read as the control signal. The pulse activates the memory chip, and the contents of the memory location (8-bit data) are placed on the data bus and brought inside the microprocessor.

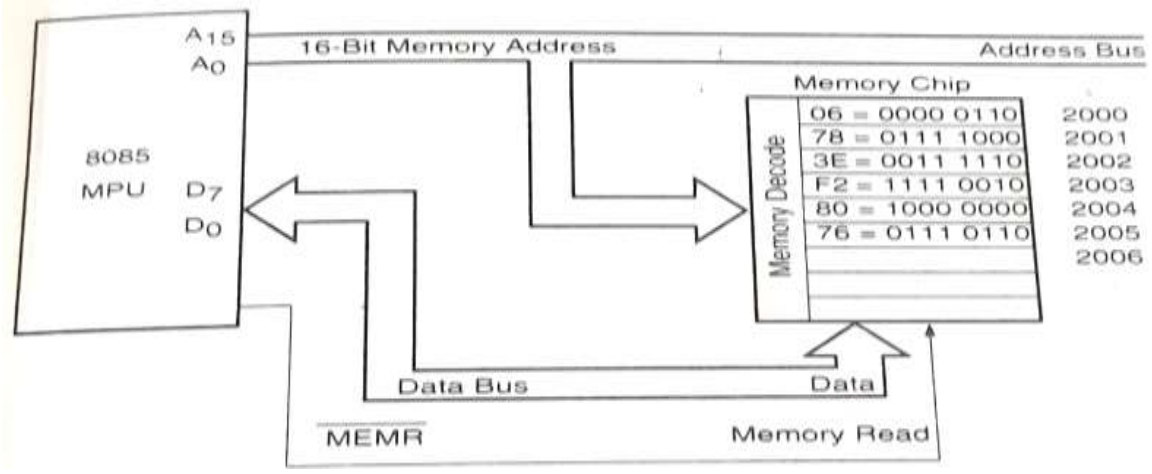


FIGURE 6 Memory Read Operation

2) Internal Data Operations and the 8085 Registers

The internal architecture of the 8085 uP determines how and what operations can be performed with the data.. These operations are:

1. Store 8-bit

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

2. Perform arithmetic and logical operations

3. Test for conditions

4. Sequence the execution of instructions

5. Store data temporarily during execution in the defined R/W memory location called the stack.

To perform theses operations the microprocessor requires registers, an arithmetic/logic unit (ALU) and control logic and internal buses.

Functions of Registers : Take an Program to understand the functions of registers

The Hex code for Memory read operation is as follows:

2000 06 MVI B, 76H

2001 78

2002 3E MVI A, F2H

2003 F2

2004 80 ADD B

2005 76 HLT

When the user enters the memory address 2000H and pushes the execute key of the trainer, the processor places the address 2000H in the program counter (PC).

1. The program counter is a 16-bit register that performs the fourth operation in the list sequencing the execution of the instructions. When the processor begins execution, it places the address 2000H on the address bus and increments the address in the PC to 2001 for the next operation. It brings the code 06, interprets the code, places the ad- dress 2001H on the address bus, and then gets byte 78H and increments the address in PC to 2002H. The processor repeats the same process for the next instruction, MVIA, F2H

2. When the processor executes the first two instructions, it uses register B to store 78H and A to store F2H in binary (Operation 1).

3. When the processor executes the instruction ADD B in the ALU (Operation 2), it adds 78H to F2H, resulting in the sum 16AH (78H+ F2H = 16AH). It replaces F2H by 6AH in A and sets the Carry flag as described next.

4. In our example, the addition operation generates a carry because the sum is larger than the size of the accumulator (8 bits). To indicate the carry, the processor sets the flip- flop called Carry (CY flag) to 1 and places logic 1 in the flag register at the designated bit position for the carry.

5. The fifth operation deals with the concept of the stack. The stack pointer is a 16-bit register used as a memory pointer to identify the stack, part of the R/W memory de- fined and used by

Siddartha Institute of Science and Technology::Puttur
(Autonomous)
(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

the processor for temporary storage of data during the execution.

3 Peripheral or Externally Initiated Operations

External devices (or signals) can initiate the following operations, for which individual pins on the microprocessor chip are assigned: Reset, Interrupt. Ready, Hold.
Reset: When the reset pin is activated by an external key (also called a reset key), all internal operations are suspended and the program counter is cleared (it holds 0000H) Now the program execution can again begin at the zero memory address. The microprocessor can be interrupted from the normal execution of instructions and asked to execute some other instructions called a service routine (for example, emergency procedures). The microprocessor resumes its operation after completing the service routine.
Ready: The 8085 has a pin called READY. If the signal at this READY pin is low, the microprocessor enters into a Wait state. This signal is used primarily to synchronize slower peripherals with the microprocessor. For example, if uP is communicated with slow responding peripheral, the UP read or write cycles delays until the peripheral device will send READY signal to uP. When this signal goes low the uP waits for an integral number of clock cycles until it goes high.

Hold: When the HOLD pin is activated by an external signal, the microprocessor re- linquishes control of buses and allows the external peripheral to use them. For example, the HOLD signal is used in Direct Memory Access (DMA) data transfer. It is an input signal.
HLDA: When the uP receives HOLD input then, Up respond HOLD request signal by sending acknowledgement signal called HLDA. It is a output signal.

MEMORY

Memory is an essential component of a microcomputer system; it stores binary instruc- tions and data for the microprocessor. There are various types of memory, which can be classified in two groups: prime (or main) memory and storage memory. In the last chap- ter, we discussed briefly two examples of prime memory: Read/Write memory (R/WM) and Read-Only memory (ROM). Magnetic tapes or disks can be cited as examples of stor- age memory. First, we will focus on prime memory and then, briefly discuss storage memory when we examine various types of memory.

The R/W memory is made of registers, and each register has a group of flip-flops or field-effect transistors that store bits of information; these flip-flops are called memory cells. The number of bits stored in a register is called a memory word; memory devices (chips) are available in various word sizes. The user can use this memory to hold programs and store data. On the other hand, the ROM stores information permanently in the form of diodes; the group of diodes can be viewed as a register. In a memory chip. all registers are arranged in a sequence and identified by binary numbers called memory addresses. To communicate with memory, the MPU should be able to select the chip, identify the register, and read from or write into the register.

The MPU uses its address bus to send the address of a memory register and uses the data bus and control lines to read from (as shown in Figure 3.2) or write into that re- gister. In the following sections, we will examine the basic concepts related to memory: its structure, its addressing, and its requirements to communicate with the MPU and build a model for R/W memory. However,

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

except for slight differences in Read/Write control signals the discussion is equally applicable to ROM.

Tri-State Devices

In-state logic devices have three states: logic 1, logic 0, and high impedance. The term Tri-State is a trademark of National Semiconductor and is used to represent three logic states. A tri-state logic device has a third line called Enable, as shown in Figure 4. When this line is activated, the tri-state device functions the same way as ordinary logic devices. When the third line is disabled, the logic device goes into the high impedance state-as if it were disconnected from the system. Ordinarily, current is required to drive a device in logic 0 and logic 1 states. In the high impedance state, practically no current is drawn from the system. Figure 4(a) shows a tri-state inverter. When the Enable is high, the circuit functions as an ordinary inverter; when the Enable line is low, the inverter stays in the high impedance state. Figure 4(b) also shows a tri-state inverter with active low Enable line-notice the bubble. When the Enable line is high, the inverter stays in the high impedance state.

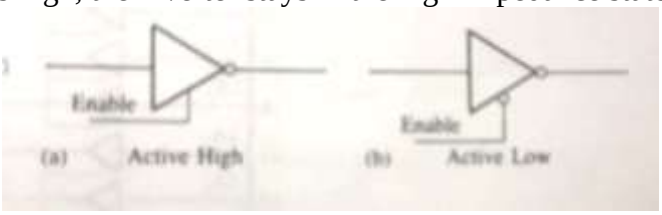


Figure 4 : Tristate Inverter a) with enable b) without enable

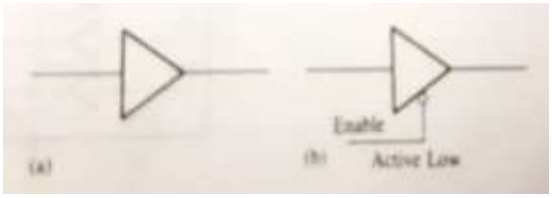


Figure 5 : Tristate buffer a) without enable b) with enable

In microcomputer systems; peripherals are connected in parallel between the address bus and the data bus. However, because of the tri-state interfacing devices, peripherals do not load the system buses. The microprocessor communicates with one device at a time by enabling the tri-state line of the interfacing device. Tri-state logic is critical to proper functioning of the microcomputer.

Decoder

The decoder is a logic circuit that identifies each combination of the signals present at its input. For example, if the input to a decoder has two binary lines, the decoder will have four output lines (Figure 6). The two lines can assume four combinations of in- put signals-00, 01, 10, 11-with each combination identified by the output lines 0 to 3. the input is 11, the output line 3 will be at logic 1, and the others will remain at logic 0. This is decoding. Figure 6(a) shows a symbolic representation for a hypothetical 2:4 decoder. It is also called a 1-out-of-4 decoder. Various types of decoders are available; for example, 3-to-8, 4-to-16 (to decode binary inputs), and 4-to-10 (to decode BCD input). In general, decoders have active low output line as well as Enable lines, as

Figure 6: 2 to 4 decoder a) without enable b) with enable and active low output

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

Flip-Flop or Latch as a Storage Element

What is memory? It is a circuit that can store bits-high or low, generally voltage levels or capacitive charges representing 1 and 0. A flip-flop or a latch is a basic element of memory To write or store a bit in the latch, we need an input data bit (DIN) and an enable signal (EN), as shown in Figure 3a. In this latch, the stored bit is always avail- able on the output line Dour. To avoid unintentional change in the input and control the availability of the output, we can use two tri-state buffers on the latch, as shown in Figure 3b). Now we can write into the latch by enabling the input buffer and read from it by enabling the output buffer. Figure 3b) shows the Write signal as WR and the Read signal as RD: these are active low signals indicated by the bar.

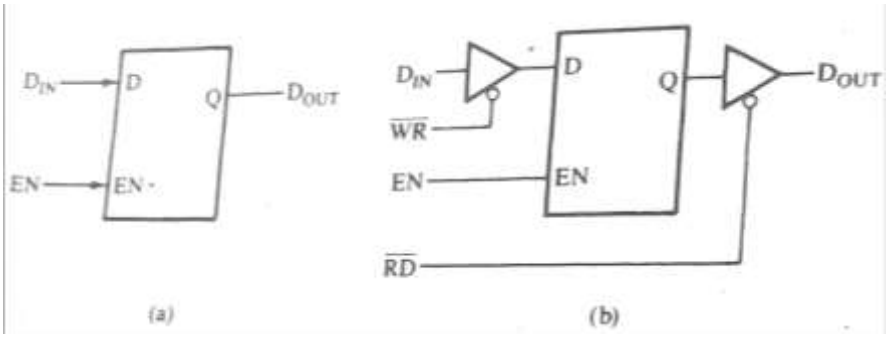


Fig. 3 Latches as storage element a) Latch and b) Latch with two tristate Buffers

This latch, which can store one binary bit, is called a memory cell.
Figure 4a shows four such cells or latches grouped together, this is a register, which has four input lines and four output lines and can store four bits: thus the size of the memory word is four bits. The size of this register is specified either as 4-bit or 1 x 4-bit, which indicates one register with four cells or four 1/0 lines. Figures 4b and 4(c) show simplified block diagrams of the 4-bit register.

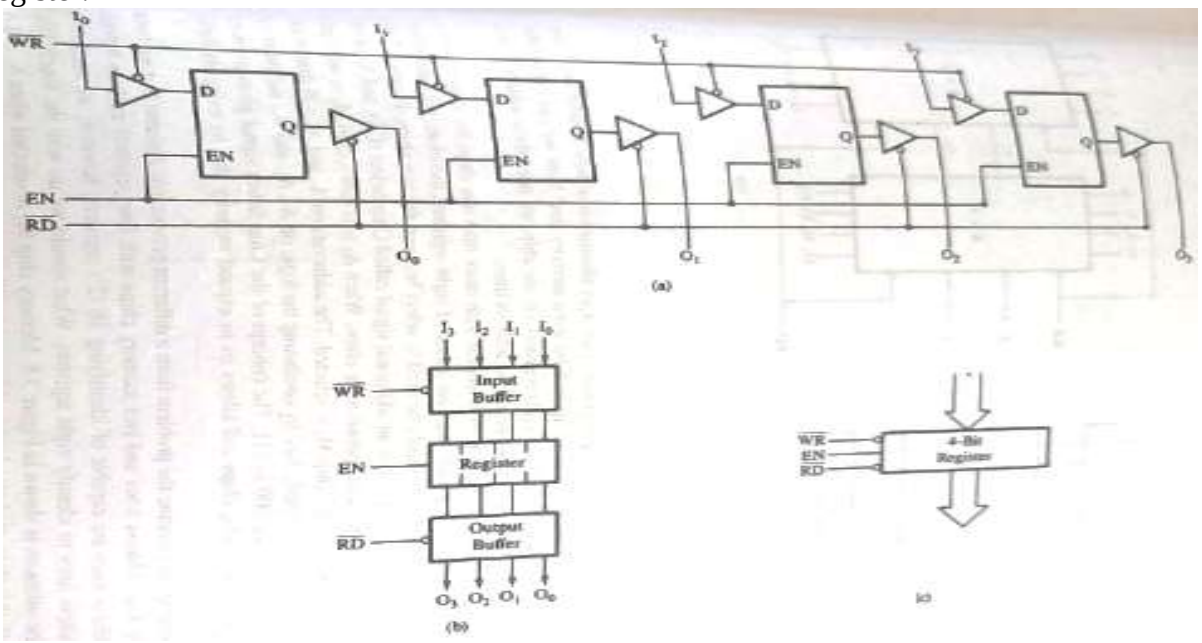


Fig 4 a) Four latches as 4-bit Register b) and c) Block diagram of a 4-bit Register

In Figure 5. four registers with eight cells (or an 8-bit memory word) are arranged in a sequence.

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

To write into or read from any one of the registers, a specific register should be identified or enabled. This is a simple decoding function; a 2-to-4 decoder can perform that function. However, two more input lines A1, and A0, called address lines, are required to the decoder.

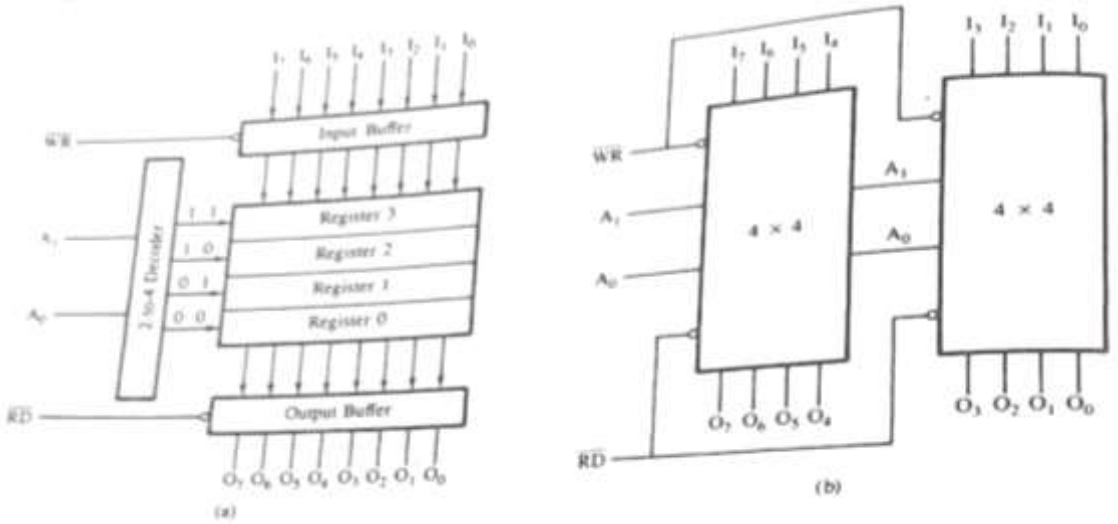


Fig. 5 4X8 bit Registers

input lines can have four different bit combinations (00, 01, 10, 11), and each combination can identify or enable one of the registers named as Register 0 through Register 3. Thus the Enable signal of the flip-flops in Figure 4 is replaced by two address lines in Figure 5.

Figure 5(a) has 8-bit registers and Figure 5(b) has two chips with 4-bit registers. This is an illustration of how smaller word size chips can be connected to make up an 8-bit word memory size. Now we can expand the number of registers. If we have eight registers on one chip, we need three address lines, and if we have 16 registers, we need four address lines.

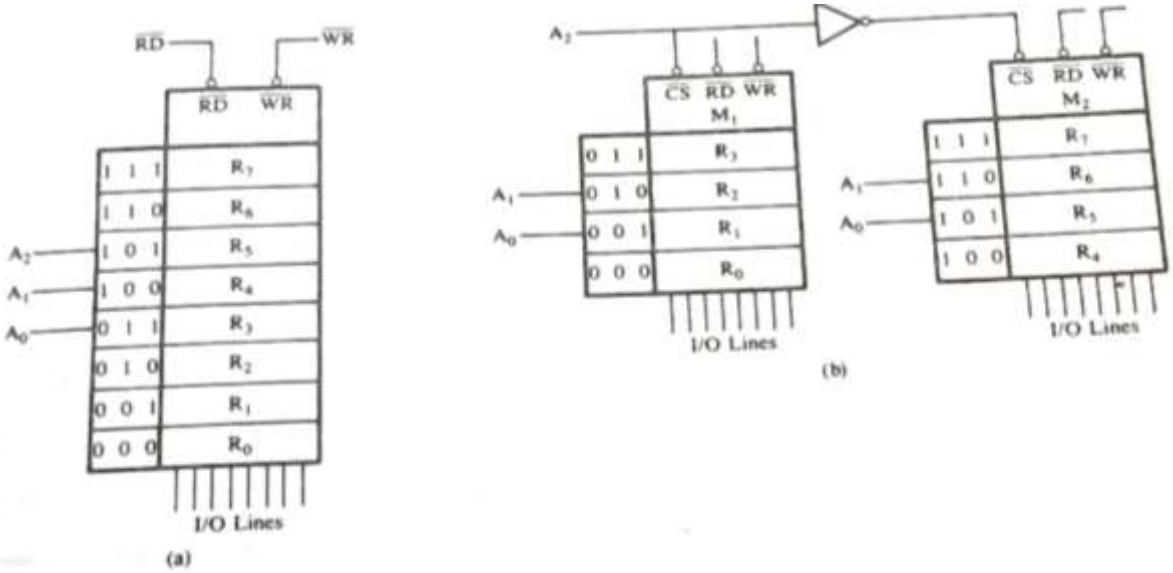


Fig 6 Two memory chips with four registers each and chip select

An interesting problem is how to deal with more than one chip: for example, two chips with four registers each. We have a total of eight registers; therefore, we need three address lines, but one

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

line should be used to select between the two chips. Figure 6(b) shows two memory chips, with an additional signal called Chip Select (CS'), and A2, (with an inverter) is used to select between the chips. When A2 is 0 (low), chip M1, is selected. and when A2, is 1 (high), chip M2, is selected. The addresses on A1 and A0, will determine the registers to be selected; thus, by combining the logic on A2, A1 and A0. the memory addresses range from 000 to 111. The concept of the Chip Select signal gives us more flexibility in designing chips and allows us to expand memory size by using multiple chips.

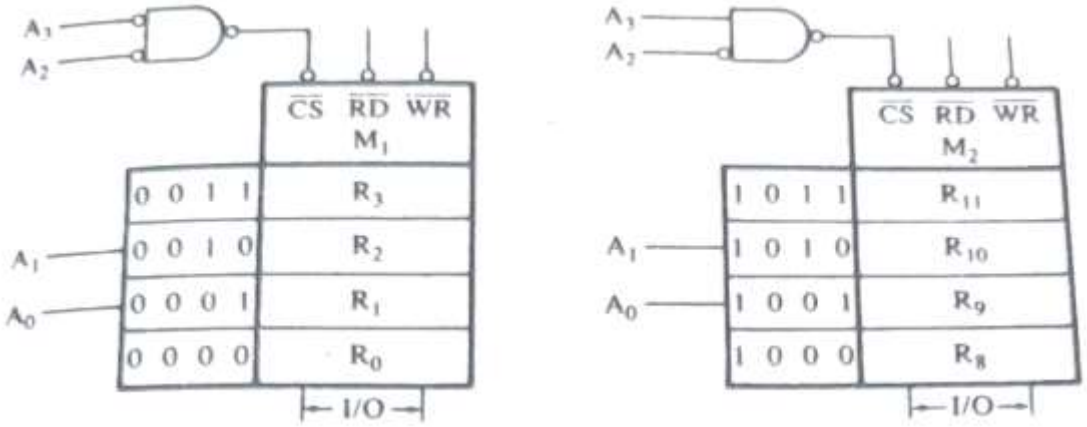


Fig 7 Addressing Eight Registers with Four Address Lines

Now let us examine the problem from a different perspective. Assume that we have available four address lines and two memory chips with four registers each as before. Four address lines are capable of identifying 16 (2^4) registers; however, we need only three address lines to identify eight registers. What should we do with the fourth line? One of the solutions is shown in Figure 7. Memory chip M1, is selected when A3, and A2 are both 0; therefore, registers in this chip are identified with the addresses ranging from 0000 to 0011 (0 to 3). Similarly, the addresses of memory chip M2 range from 1000 to 1011 (8 to B); this chip is selected only when A3, is 1 and A2, is 0. In this example, we need three lines to identify eight registers: two for registers and one for Chip Select. we used the fourth line for Chip Select also. This is called complete or absolute decoding.

After reviewing the above explanation, we can summarize the requirements of a memory chip, then build a model and match the requirements with the microprocessor bus concepts

1. A memory chip requires address lines to identify a memory register. The number of address lines required is determined by the number of registers in a chip (2^n = Number of registers where n is the number of address lines). The 8085 microprocessor has 16 address lines. Of these 16 lines, the address lines necessary for the memory chip must be connected to the memory chip.

2. A memory chip requires a Chip Select (CS') signal to enable the chip. The remaining address lines (from Step 1) of the microprocessor can be connected to the CS' signal through an interfacing logic.

3. The address lines connected to CS' select the chip, and the address lines connected to the address lines of the memory chip select the register. Thus the memory address of a register is determined by the logic levels (0/1) of all the address lines (including the address lines used for

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

CS').

4. The control signal Read (RD') enables the output buffer, and data from the selected register are made available on the output lines. Similarly, the control signal Write (WR) enables the input buffer, and data on the input lines are written into memory cells. The microprocessor can use its Memory Read and Memory Write control signals to enable the buffers and the data bus to transport the contents of the selected register between the microprocessor and memory.

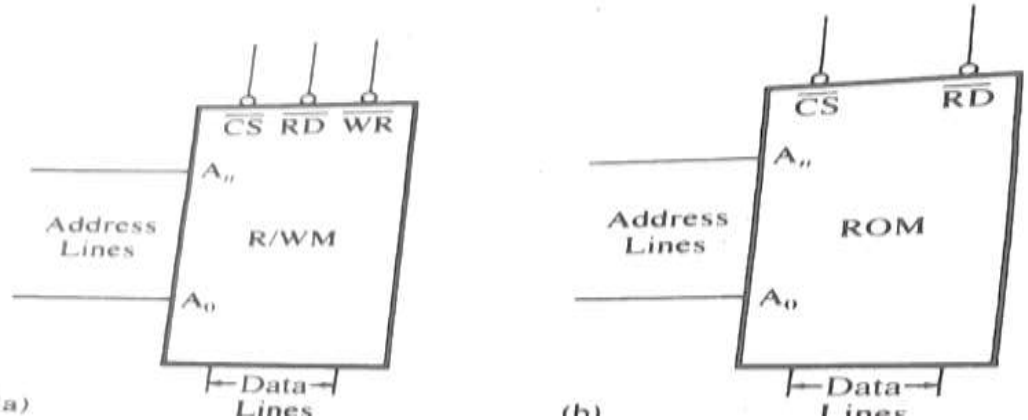


Fig.8 a) R/W Memory model b) ROM model

A model of a typical memory chip representing the above requirements is shown in Figure 8. Figure 8(a) represents the R/W memory and Figure 8(b) represents the Read-Only memory; the only difference between the two as far as addressing is concerned is that ROM does not need the WR' signal. Internally, the memory cells are arranged in a matrix format-in rows and columns; as the size increases, the internal decoding scheme we discussed becomes impractical. For example, a memory chip with 1024 registers would require a 10-to-1024 decoder. If the cells are arranged in six and four columns, the internal decoding circuitry can be designed with two decoders, one for selecting a and the other for selecting a column. However, we will not be concerned about the internal row and column arrangement because it does not affect our external interfacing logic, which is explained in the next section.

Memory Map and Addresses

Typically, in an 8-bit microprocessor system, 16 address lines are available for memory This means it is a numbering system of 16 binary bits and is capable of identifying 2^{16} (65,536) memory registers, each register with a 16-bit address. The entire memory addresses can range from 0000 to FFFF in Hex. A memory map is a pictorial representation in which memory devices are located in the entire range of addresses. Memory addresses provide the locations of various memory devices in the system, and the interfacing logic defines the range of memory addresses for each memory device. The concept of memory map and memory addresses can be illustrated with an analogy of identical houses built in sequence and their postal addresses, or numbers.

Let us assume that houses are given three-digit decimal numbers, which will enable us to number one thousand houses from 000 to 999. Because it is cumbersome to direct someone to houses with large numbers, the numbering scheme can be devised with the concept of a row or block. Each block will have a hundred houses to be numbered with the last two digits from 00 to 99. Similarly, the blocks are also identified by the first decimal digit. For example, a house with the

Siddartha Institute of Science and Technology::Puttur
(Autonomous)
(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

number 247 is house number 47 in block 2. With this scheme, all the houses in block 0 will be identified 000 to 099, in block 2 from 200 to 299, and in block 9 from 900 to 999. This numbering scheme with three decimal digits is capable of giving addresses to one thousand houses from 000 to 999 (10) blocks of 100 houses each). Let us also assume that all houses are identical and have eight rooms. The example of numbering the houses is directly applicable to assigning addresses to memory registers. In the binary number system. 16 binary digits can have 65,536 (2) different combinations. In the hexadecimal number system, 16 binary bits are equivalent to four Hex digits that can be used to assign addresses to 65,536 (0000H to FFFFH) memory registers in various memory chips. In our analogy, a memory chip is similar to a block in a housing development and a register can be viewed as a house with eight identical rooms.

Let us assume that we have a memory chip with 256 registers. Therefore, we need only 256 numbers (out of 65,536) that require eight address lines (2^8= 256). Now the question is what we should do with the remaining eight address lines of the microprocessor. We can find a clue in our housing analogy. Let us assume that we have only 100 houses in block five. They will be numbered as 500 to 599; the first digit 5 remains constant and the next two digits vary from 00 to 99. Similarly, we can use the remaining eight address lines to assign fixed logic to generate a constant (fixed) number. This can be accomplished by using the remaining eight lines for the Chip Select through appropriate logic gates, as shown in Example 3.1.

As mentioned previously, in computer systems, we define 1024 as 1K; therefore a IK-byte memory chip has 1024 registers with 8 bits each. Similarly, a group of 256 registers is defined as one page and each register is viewed as a line to write on. This is analogous to a notebook containing various pages, with each page having a certain number of lines With this analogy, we can view IK-byte memory as a chip with four pages (1024/256=4) with each page having 256 registers. With two Hex digits, 256 registers can be numbered from 00 to FFH: 1024 registers can be numbered with four digits from 0000 to 03FF. If we examine the high-order digits of IK-byte memory, we find that they range from 00 to 03 representing four pages (00, 01, 02, and 03). In 8-bit microprocessor systems, this page concept is used frequently. In 16- and 32-bit microprocessor systems, the page concept (256 registers) defined here is not applicable; it is defined differently. based on the microprocessor used in a system.

So far we have been using the term addresses or address range for a given memory chip. The term memory map is used generally for the entire address ranges of the memory chips in a given system. The relationship of a row of houses to the road map is similar to the relationship of memory addresses to the memory map. However, these terms are also used synonymously.

Example 3.1

Illustrate the memory address range of the chip with 256 bytes of memory, shown in Figure 9(a), and explain how the range can be changed by modifying the hardware of the Chip Select CS line in Figure 9(b).

Solution

Figure 9(a) shows a memory chip with 256 registers with eight I/O lines: the memory size of the chip is expressed as 256 x 8. It has eight address lines (A7-A0), one Chip Select signal (CS) (active low), and two control signals Read (RD') and Write (WR'). The eight address lines (A7-A0) of the microprocessor are required to identify 256 memory registers. The remaining eight lines (A15- A8) are connected to the Chip Select (CS') line through inverters and the NAND

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

gate. The memory chip is enabled or selected when CS' goes low. Therefore, to select the chip, the address lines A15-A8 should be at logic 0. which will cause the output of the NAND gate to go low. No other logic levels on the lines A15-A8, can select the chip. Once the chip is selected (enabled), the remaining ad- dress lines A7-A0, can assume any combination from 00H to FFH and identify any of the 256 memory registers through the decoder. Therefore, the memory addresses of the chip in Figure 9(a) will range from 0000H to 00FFH, as shown below.

A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 = 0000H

1 1 1 1 1 1 1 1 = 00FFH

Chip Enable or Chip Select

Register Select

The address lines A15-A8, which are used to select the chip, must have fixed logic levels, and these lines are called high-order address lines. The address lines A7-A0, which

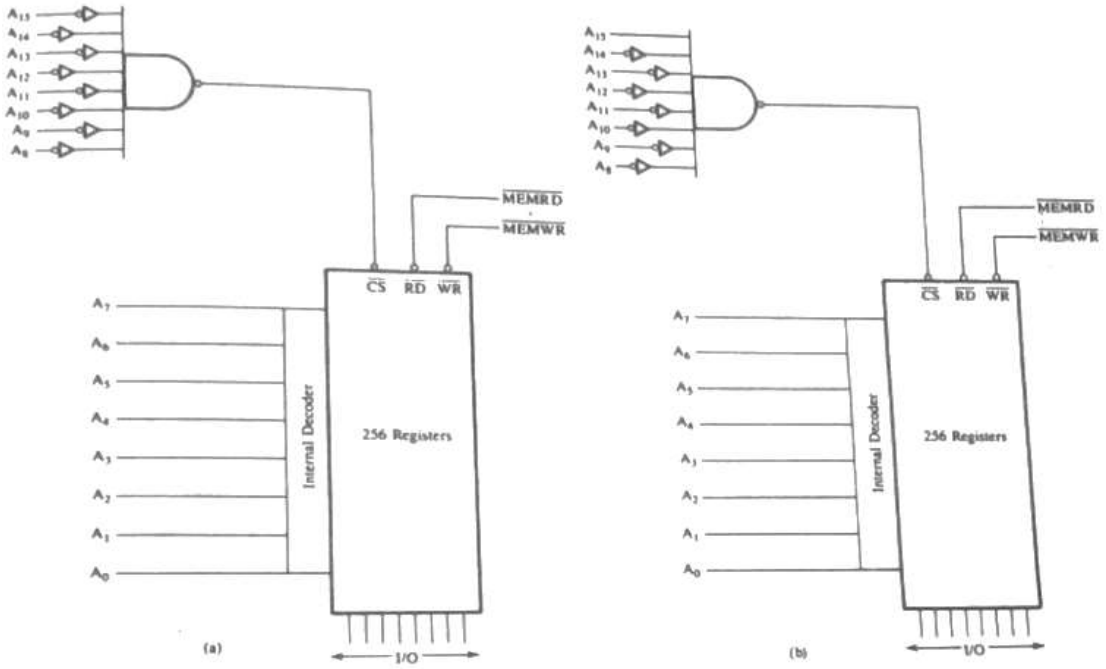


FIGURE 9 Memory Maps: 256 Bytes of Memory with range a) from 0000H to 00FFH b)from 8000H to 80FFH

are used to select a register, are called low-order address lines, and they can be assigned logic levels from all 0's to all Is and any in-between combination. For example, when the address lines A7-A0, are all 0's, the register number 0 is selected, and when they are all 1's the register number 255 (FFH) is selected. The Chip Select addresses are determined by the hardware (the inverters and NAND gate); therefore, the memory addresses of the chip can be changed by modifying the hardware. For example, if the inverter on line Ass is removed, as shown in Figure 9b, the address required on A15-A8, to enable the chip will be as follows:

A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 = 8000H

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

The memory address range in Figure 9(b) will be 8000H to 80FFH. The memory chips in Figures. 9(a) and (b) are the same chips. However, by changing the hardware of the Chip Select logic, the location of the memory in the map can be changed, and memory can be assigned addresses in various locations over the entire map of 0000H to FFFFH.

After reviewing the example and the previous explanation of the memory map, we can summarize the following points.

1. In a numbering system, the number of digits used determines the maximum addressing capacity of the system. Sixteen address lines (16 bits) of the 8085 can address 65536 memory registers; this is similar to three decimal digits providing microprocessor the postal addresses for one thousand houses.
2. For a given memory chip, the number of address lines required to identify the registers is determined by the number of registers in the chip. The remaining address lines can be used for selecting the chip.

3. The address lines that are used to select registers in memory, called low-order address lines, can be assigned any logic levels (0 or 1) depending on the register being selected. The address lines that are used to select a chip, called high-order address lines, must have a fixed logic for a given address range.
4. The memory address range of a given chip can be changed by changing the hardware of the Chip Select (C'S) line. This line is also known as the Chip Enable (CE') line.

Memory Range of a 1K Memory Chip: In the previous example, the high-order and the low-order address lines were equally divided, eight each of the 16 address lines. However, if a chip includes more than 256 registers (e.g. 512 or 1024 registers), the number of low-order address lines will be higher than that of the high-order address lines, as shown in the following example.

Example :3.2 Explain the memory address range of IK (1024 x 8) memory shown in Figure 10 and explain the changes in the addresses if the hardware of the CS line is modified.

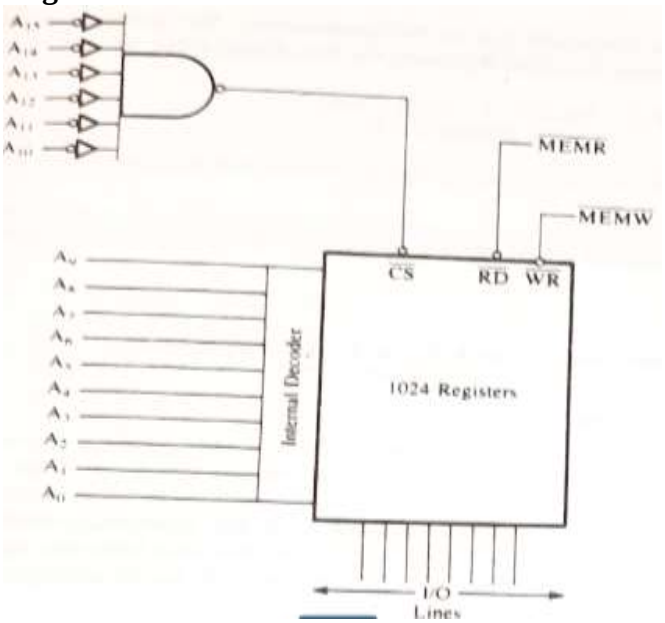


Fig. 10 Memory Address range: 1024 Bytes of Memory

Siddartha Institute of Science and Technology::Puttur
(Autonomous)
(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

The memory chip has 1024 registers; therefore 10 address lines (A9-A0) are required to identify the registers. The remaining six address lines (A15-A8) of the microprocessor are used for the Chip Select (CS) signal. In Figure 10, the memory chip is enabled when the address lines A15-A10 are at logic 0. The address lines Ag-A, can assume any address of the 1024 registers, starting from all 0's to all 1s, as shown next.

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= 0000H
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	= 03FFH

The memory addresses range from 0000H to 03FFH. By combining the high-order and the low-order address lines, we can specify the complete memory address range of a given chip. As explained in the previous example, the memory addresses of the 1K chip in Figure 10 can be changed to any other location by changing the hardware of the CS line. For example, if A, is connected to the NAND gate without an inverter, the memory addresses will range from 8000H to 83FFH.

Calculation of Memory Address Lines

In the last two examples, the address lines of the memory chips were given. The number of address lines necessary for a given chip can be obtained from data sheets. However, we need to know the relationship between the number of registers in a memory chip and the number of address lines. For a chip with 256 registers, we need 256 binary numbers to identify each register. Each address line can assume only two logic states (0 and 1); therefore, we need to find the power of 2 that will give us 256 combinations. The problem can be restated as follows: Find where 2256. By taking the log of both sides, we get:

log 2^x = log 256 -> x log 2 = log 256
x = log 256/log 2 = 8

Here x represents the number of address lines needed to obtain 256 binary numbers,

Example 3.3 Calculate the address lines required for an 8K-byte (1024x8 = 8192 registers) memory chip.

Solution : Number of address lines = log 8192/log 2 = 13 address lines

Memory Word Size

Memory devices (chips) are available in various word sizes (1. 4. and 8) and the size of a memory chip is generally specified in terms of the total number of bits it can store. On the other hand, the memory size in a given system is generally specified in terms of bytes Therefore, it is necessary to design a byte-size memory word. For example, a memory chip of size 1024 x 4 has 1024 registers and each register can store four bits; thus it can store a total of 4096 (1024 x 4 = 4096) bits. To design 1K-byte (1024 x 8) memory, we will need two chips, each chip will provide four data lines.

Siddartha Institute of Science and Technology::Puttur
(Autonomous)
(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

Example 3.4 Calculate the number of memory chips needed to design 8K-byte memory if the memory chip size is 1024 x 1.

Solution: The chip 1024 x 1 has 1024 (1K) registers and each register can store one bit with one data line. We need eight data lines for byte-size memory. Therefore, eight chips are necessary for IK-byte memory. For 8K-byte memory, we will need 64 chips. We can arrive at the same answer by dividing 8K-byte by IK x 1 as follows:

8192 x 8/ 1024 × 1 = 64

Memory and Instruction Fetch Operation

The primary function of memory is to store instructions and data and to provide that in- formation to the MPU whenever the MPU requests it. The MPU requests the information
by sending the address of a specific memory register on the address bus and enables the data flow by sending the control signal, as illustrated in the next example.

Example
The instruction code 0100 1111 (4FH) is stored in memory location 2005H. Illustrate the data flow and list the sequence of events when the instruction code is fetched by the MPU.

To fetch the instruction located in memory location 2005H, the following steps are per- formed:

1. The program counter places the 16-bit address 2005H of the memory location on the

address bus (Figure 11). 2. The control unit sends the Memory Read control signal (MEMR, active low) to enable the output buffer of the memory chip.

3. The instruction (4FH) stored in the memory location is placed on the data bus and transferred (copied) to the instruction decoder of the microprocessor.
4. The instruction is decoded and executed according to the binary pattern of the instruction.
Figure 11 shows how the 8085 MPU fetches the instruction using the address, the data, and the control buses. Figure 11 is similar to Figure 3 Memory Read operation except that Figure 11 shows additional details.

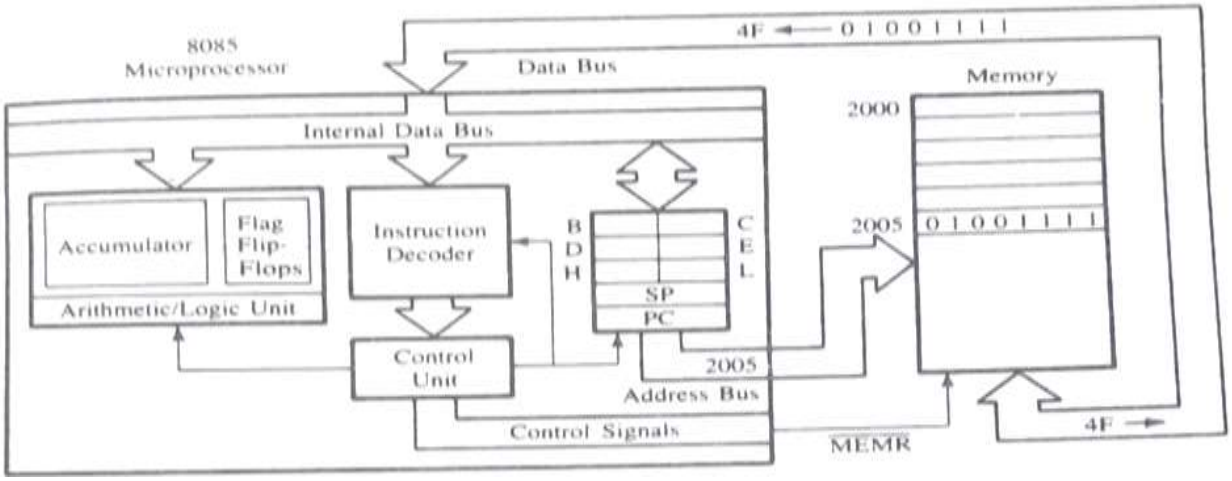


Figure 11 Instruction Fetch Operation

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

Memory Classification

Memory can be classified into two groups: primary (system or main) memory and storage memory. The R/WM and ROM are examples of prime memory: this memory As mentioned earlier, memory can be classified into two groups: prime (system or main) is the memory the microprocessor uses in executing and storing programs. This t should be able to respond fast enough to keep up with the execution speed of the micro- processor. Therefore, it should be random access memory, meaning that the microprocessor should be able to access information from any register with the same speed (independent of its place in the chip). The size of a memory chip is specified in terms of bits. For example, a IK memory chip means it can store IK (1024) bits (not bytes). On the other hand, memory in a system such as a PC is specified in bytes. For example, 4M memory in a PC means it has 4 megabytes of memory.

The other group is the storage memory, such as magnetic disks and Figure 12). This memory is used to store programs and results after the completion of program execution Information stored in these memories is nonvolatile, meaning information remains intact even if the system is turned off. The microprocessor can directly execute or process programs stored in these devices; programs need to be copied into the RW prime memory first. Therefore, the vine of the prime memory, ch 512K or SM megabytes), determines how large a pengram the system can process. The size of the storage memory is unlimited when i dick tape is full, the next one can be used

Figure 12 shows two groups in storage secondary storage and backup range. The secondary storage is similar to what you put on a shelf in your study, and the backup is similar to what you on your attic. The secondary storage and the backup age include devices such as disks, magnetic tapes, magnetic bubble memory, and charged-coupled devices, as shown in Figure11. The primary features of these devices are high capacity, low cost, and low access. A disk is similar to a record the access to the stored information in the disk is semirandom. The remaining devices shown in Figure 11 are serial, meaning if information is stored in the middle of the tape, it can be accessed after running half the tape.

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

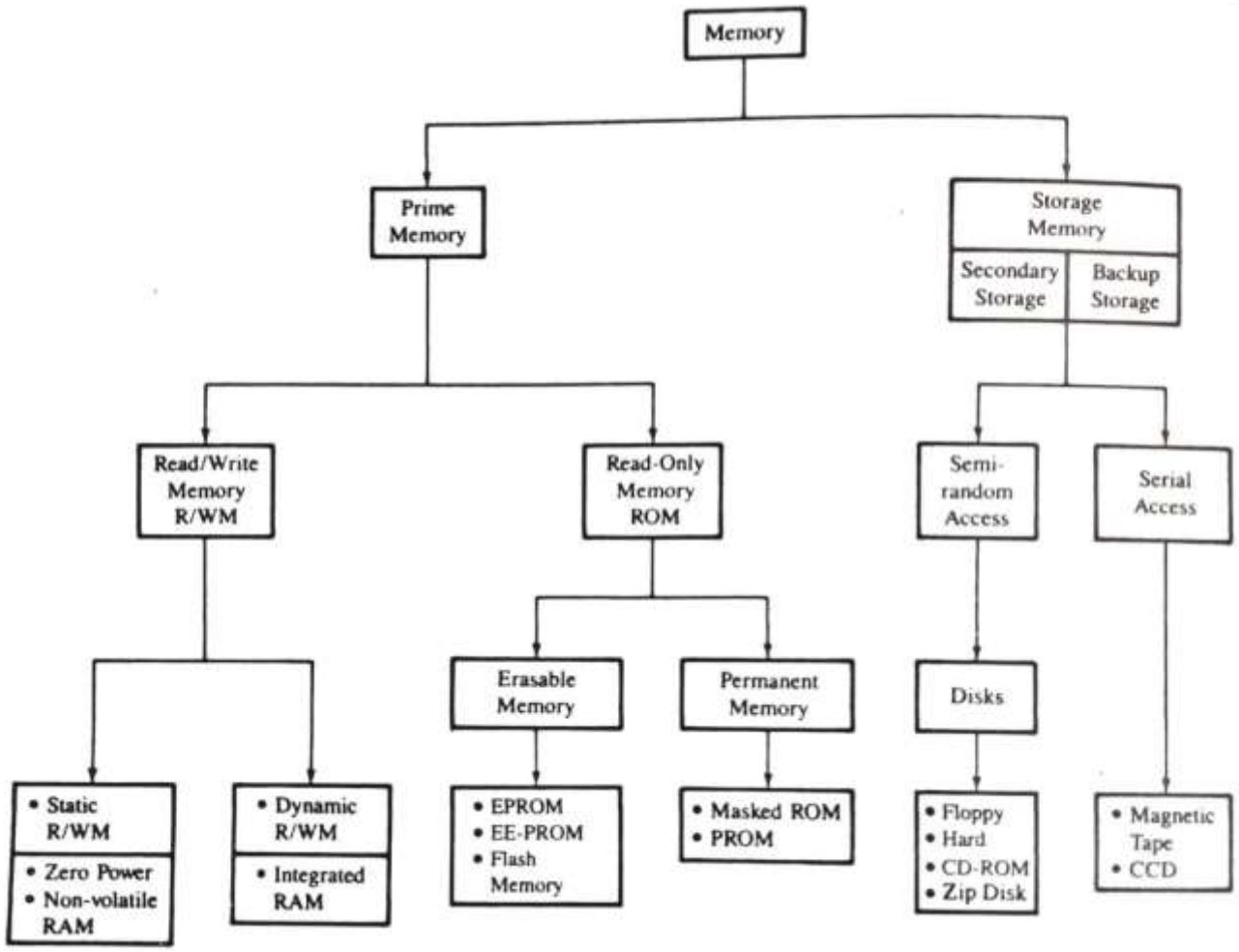


Figure 11 Memory Classification

Figure 11 shows that the prime (system) memory is divided into two r proups: Read/Write memory (R/WM) and Read-Only memory (ROM): each group includes sev eral different types of memory, as discussed below

R/WM (READ/WRITE MEMORY)

As the name suggests, the microprocessor can write into or read from this memory, it is popularly known as Random Access memory (RAM). It is used primarily for information that is likely to be altered, such as writing programs or receiving data. This memory is volatile, meaning that when the power is turned off, all the contents are destroyed. Two types of R/W memories static and dynamic-are available.

Static Memory (SRAM) This memory is made up of flip-flops, and it stores the hit as a voltage. Each memory cell requires six transistors; therefore, the memory chip has low density but high speed. This memory is more expensive and consumes more power than the dynamic memory described in the next paragraph. In high-speed processors (such as Intel 486 and Pentium). SRAM known as cache memory is included on the processor chip. In addition, high-speed cache memory is also included external to the processor to improve the performance of a system

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

Dynamic Memory (DRAM) This memory is made up of MOS transistor gates, and it stores the hit as a charge. The advantages of dynamic memory are that it has high density and low power consumption and is cheaper than static memory. The disadvantage is that the charge thit information) leaks; therefore, stored information needs to be read and writ ten again every few milliseconds. This is called refreshing the memory, and it requires ex- tra circuitry, adding to the cost of the system. It is generally economical to use dynamic memory when the system memory size is at least 8K: for small systems, the static memory is appropriate. However, in recent years, the processor speed has reached beyond 200 MH, and 1000 MHz processors are in the design stage. In comparison to the processor speed the DRAM is too slow. To increase the speed of DRAM various techniques are be ing used. These techniques have resulted in high-speed memory chips such as EDO (Extended Data Out), SDRAM (Synchronous DRAM), and RDRAM (Rambus DRAM)

ROM (READ-ONLY MEMORY) The ROM is nonvolatile memory; it retains stored information even if the power is turned off. This memory is used for programs and data that need not be altered. As the name suggests, the information can be read only, which means once a bit pattern is stored, it is permanent or at least semi permanent. The permanent group includes two types of memory: masked ROM and PROM. The semipermanent group also includes two types of memory: EPROM and EE-PROM, as shown in Figure 11. The concept underlying the ROM can be explained with the diodes arranged in a matrix format. The horizontal lines are connected to vertical lines only through the diodes: they are not connected where they appear to cross in the diagram. Each of the eight horizontal rows can be viewed as a register with binary addresses ranging from 000 to 111: information is stored by the diodes in the register as Os or Is. The presence of a diode stores 1. and its absence stores 0. When a register is selected, the voltage of that line goes high, and the output lines, where diodes are connected, go high. For example, when the memory register 111 is selected, the data byte 0111 1000 (78H) can be read at the data lines D7-D0

The diode representation is a simplified version of the actual MOSFET memory cell. The manufacturer of the ROM designs the MOSFET matrix according to the information to be stored; therefore, information is permanently recorded in the ROM. as a song is recorded on a record. Five types of ROM-masked ROM, PROM, EPROM, EE- PROM, and Flash Memory-are described in the following paragraphs. masking and Masked ROM In this ROM, a bit pattern is permanently recorded by the metalization process. Memory manufacturers are generally equipped to do this process. It is an expensive and specialized process, but economical for large production quantities.

PROM (Programmable Read-Only Memory) This memory has nichrome or poly- silicon wires arranged in a matrix; these wires can be functionally viewed as diodes or fuses. This memory can be programmed by the user with a special PROM programmer that selectively burns the fuses according to the bit pattern to be stored. The process is known as "burning the PROM," and the information stored is permanent.

EPROM (Erasable Programmable Read-Only Memory) This memory stores a bit by charging the floating gate of an FET. Information is stored by using an EPROM pro- grammer, which applies high voltages to charge the gate. All the information can be erased by exposing the chip to ultraviolet light through its quartz window, and the chip can be reprogrammed. Because the chip can be reused many times, this memory is ide- ally suited for product development, experimental projects, and college laboratories. The disadvantages of EPROM are (1) it must be

Siddartha Institute of Science and Technology::Puttur
(Autonomous)
(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

taken out of the circuit to erase it, (2) the en- tire chip must be erased, and (3) the erasing process takes 15 to 20 minutes.

EE-PROM (Electrically Erasable PROM) This memory is functionally similar to EPROM, except that information can be altered by using electrical signals at the register level rather than erasing all the information. This has an advantage in field and remote control applications. In microprocessor systems, software update is a common occur- rence. If EE-PROMs are used in the systems, they can be updated from a central com- puter by using a remote link via telephone lines. Similarly, in a process control where tim- ing information needs to be changed, it can be changed by sending electrical signals from a central place. This memory also includes a Chip Erase mode, whereby the entire chip can be erased in 10 ms vs. 15 to 20 min. to erase an EPROM. However, this memory is expensive compared to EPROM or flash memory (described in the next paragraph).

Flash Memory: This is a of EE-PROM that is becoming popular. The major difference between the flash memory and EE-PROM is in the erasure procedure: The EE- PROM can be erased at a register level, but the flash memory must be erased either in its entirety or at the sector (block) level. These memory chips can be erased and programmed at least a times. The power supply requirement for programming these chips was around 12 V, but now chips are available that can be programmed using a power supply as low as 1.8 V. Therefore, this memory is ideally suited for low-power systems..

In a microprocessor-based product, programs are generally written in ROM, and data that are likely to vary are stored, in R/WM. For example, in a microprocessor- controlled oven, programs that run the oven are permanently stored in ROM, and data such as baking period, starting time, and temperature are entered in R/W memory through the keyboard. On the other hand, when microcomputers are used for developing software or for learning purposes, programs are first written in R/W memory, and then stored on a storage memory such as a cassette tape or floppy disk.

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

INPUT AND OUTPUT (I/O) DEVICES

Input/output devices are the means through which the MPU communicates with "the out- side world." The MPU accepts binary data as input from devices such as keyboards and A/D converters and sends data to output devices such as LEDS or printers. There are two different methods by which I/O devices can be identified; one uses an 8-bit address and the other uses a 16-bit address. These methods are described briefly in the following sections.

a) I/Os with 8-Bit Addresses (Peripheral-Mapped I/O)

In this type of I/O, the MPU uses eight address lines to identify an input or an output de- vice; this is known as peripheral-mapped I/O (also known as I/O-mapped I/O). This is an 8-bit numbering system for I/Os used in conjunction with Input and Output instructions. This is also known as I/O space, separate from memory space, which is a 16-bit numbering system. The eight address lines can have 256 (2 combinations) addresses; thus, the MPU can identify 256 input devices and 256 output devices with addresses ranging from 00H to FFH. The input and output devices are differentiated by the control signals; the MPU uses the I/O Read control signal for input devices and the I/O Write control signal for output devices. The entire range of I/O addresses from 00 to FF is known as an I/O map, and individual addresses are referred to as I/O device addresses or I/O port numbers. If we use LEDs as output or switches as input, we need to resolve two issues: how to assign addresses and how to connect these I/O devices to the data bus. In a bus architecture, these devices cannot be connected directly to the data bus or the address bus; all connections must be made through tri-state interfacing devices so they will be enabled and connected to the buses only when the MPU chooses to communicate with them. In the case of memory, we did not have to be concerned with these problems because of the internal address decoding. Read/Write buffers, and availability of CS and control signals of the memory chip. In the case of I/O devices, we need to use external interfacing devices. The steps in communicating with an I/O device are similar to those in communicating with memory and can be summarized as follows:

1. The MPU places an 8-bit address on the address bus, which is decoded by external decode logic
2. The MPU sends a control signal (1/O Read or I/O Write) and enables the I/O device.
3. Data are transferred using the data bus.

b) **I/Os with 16-Bit Addresses (Memory-Mapped I/O):** In this type of I/O, the MPU uses 16 address lines to identify an I/O device: an I/O is connected as if it is a memory register. This is known as memory-mapped I/O. The MPU uses the same control signal (Memory Read or Memory Write) and instructions as those of memory. In some microprocessors, such as the Motorola 6800, all 1/Os have 16-bit addresses; I/Os and memory share the same memory map (64K). In memory-mapped I/O. the MPU follows the same steps as if it is accessing a memory register.

EXAMPLE OF A MICROCOMPUTER SYSTEM

On the basis of the discussion in the previous sections, we can expand the microcomputer system shown in Figure 1 to include additional details. Figure 12 illustrates such a system. It shows the

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

8085 MPU, two types of memory (EPROM and R/WM), input and output, and the buses linking all peripherals (memory and I/Os) to the MPU.

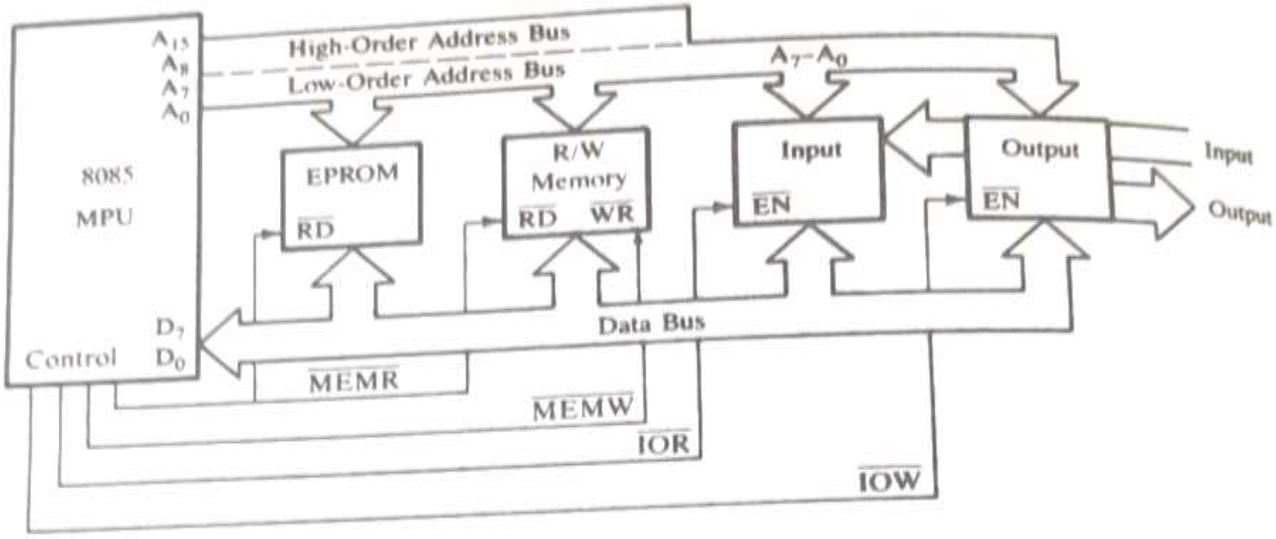


Figure 12 Example of Microcomputer System

The address lines A15-A0, are used to address memory, and the low-order address bus A7-A0, is used to identify the input and the output. The data bus D7-D0, is bidirectional and common to all the devices. The four control signals generated by the MPU are connected to different peripheral devices, as shown in Figure 12.

The MPU communicates with only one peripheral at a time by enabling the peripheral through its control signal. For example, to send data to the output device, the MPU places the device address (output port number) on the address bus, data on the data bus, and enables the output device using the control signal IOW (I/O Write). The output device latches and displays data if the output device happens to be LEDs. The other peripherals that are not enabled remain in a high impedance state called tri-state , similar to being disconnected from the system. Figure12 is a simplified block diagram of the system; it does not show such details as data latching and tri-state devices. Figure 13 shows an expanded version of the output section and the buses of Figure 12. The block diagram includes tri-state bus drivers, a decoder, and a latch.

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

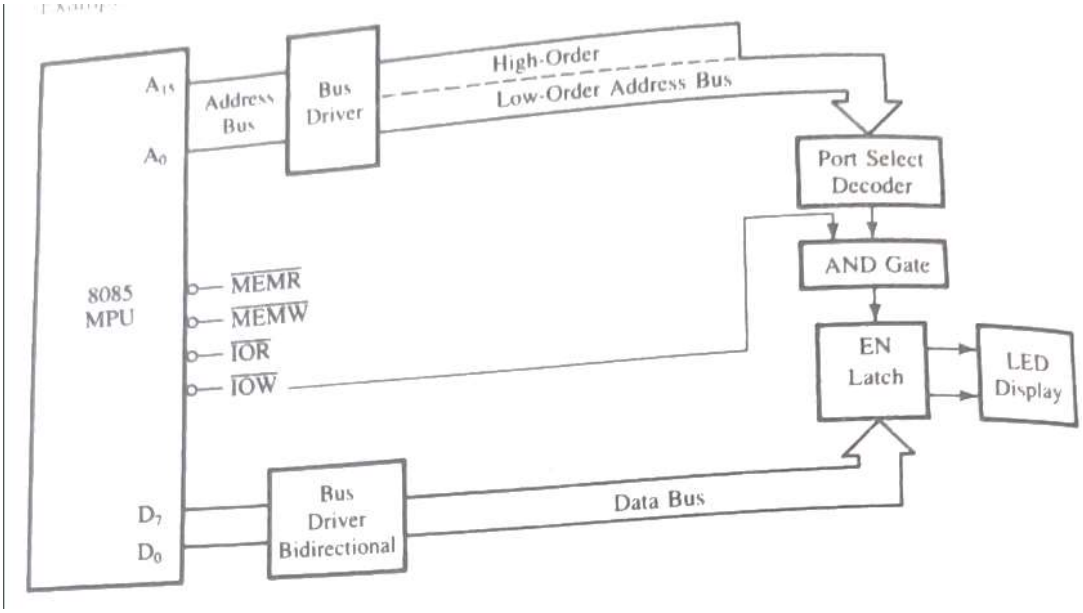


FIGURE 13 The Output Section of the Microcomputer System Illustrated in Figure 3.15

The bus drivers increase the current driving capacity of the buses, the decoder decodes the address to identify the output port, and the latch holds data output for display. These devices are called interfacing devices. The interfacing devices are semiconductor chips that are needed to connect peripherals to the bus system. Before we discuss interfacing concepts, we will review these interfacing devices

Course Code: 20EC0416

R20

Siddartha Institute of Science and Technology::Puttur
(Autonomous)
(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

UNIT-II
8085 Microprocessor Architecture:

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

The 8085 MPU

The term microprocessor unit (MPU) is similar to the term central processing unit (CPU) used in traditional computers. We define the MPU as a device or a group of de- vices (as a unit) that can communicate with peripherals, provide timing signals, direct data flow, and perform computing tasks as specified by the instructions in memory. The unit will have the necessary lines for the address bus, the data bus, and the control signals, and would require only a power supply and a crystal (or equivalent frequency. determining components) to be completely functional. Using this description, the 8085 microprocessor can almost qualify as an MPU, but with the following two limitations.

1. The low-order address bus of the 8085 microprocessor is multiplexed (time-shared) with the data bus. The buses need to be demultiplexed.
2. Appropriate control signals need to be generated to interface memory and I/O with the 8085. (Intel has some specialized memory and I/O devices that do not require such control signals.)

This section shows how to the bus and generate the control signals after describing the 8085 microprocessor and illustrates the bus timings.

The 8085 Microprocessor

The 8085A (commonly known as the 8085) is an 8-bit general-purpose microprocessor capable of addressing 64K of memory. The device has forty pins, requires a +5 V single power supply, and can operate with a 3-MHz single-phase clock. The 8085A-2 version can operate at the maximum frequency of 5 MHz. The 8085 is an enhanced version of its predecessor, the 8080A; its instruction set is upward-compatible with that of the 8080A, meaning that the 8085 instruction set includes all the 8080A instructions plus some additional ones. Figure.1 shows the logic pinout of the 8085 microprocessor. All the signals can be classified into six groups:
(1) address bus,
(2) data bus,
(3) control and status signals,
(4) power supply and frequency signals.
(5) externally initiated signals, and
(6) serial I/O ports

ADDRESS BUS

The 8085 has 16 signal lines (pins) that are used as the address bus; however, these lines are split into two segments: A15-A8, and AD7-AD,0 The eight signal lines, A15-A8, a unidirectional and used for the most significant bits, called the high-order address of a 16-bit address. The signal lines AD7-AD0, are used for a dual purpose, as explained in the next section

MULTIPLEXED ADDRESS/DATA BUS

The signal lines AD7-AD0, are bidirectional: they serve a dual purpose. They are used as the low-order address bus as well as the data bus. In executing an instruction, during the earlier part of the cycle, these lines are used as the low-order address bus. During the later part of the cycle, these lines are used as the data bus. (This is also known as multiplexing the bus) However, the low-order address bus can be separated from these signals by using a latch

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

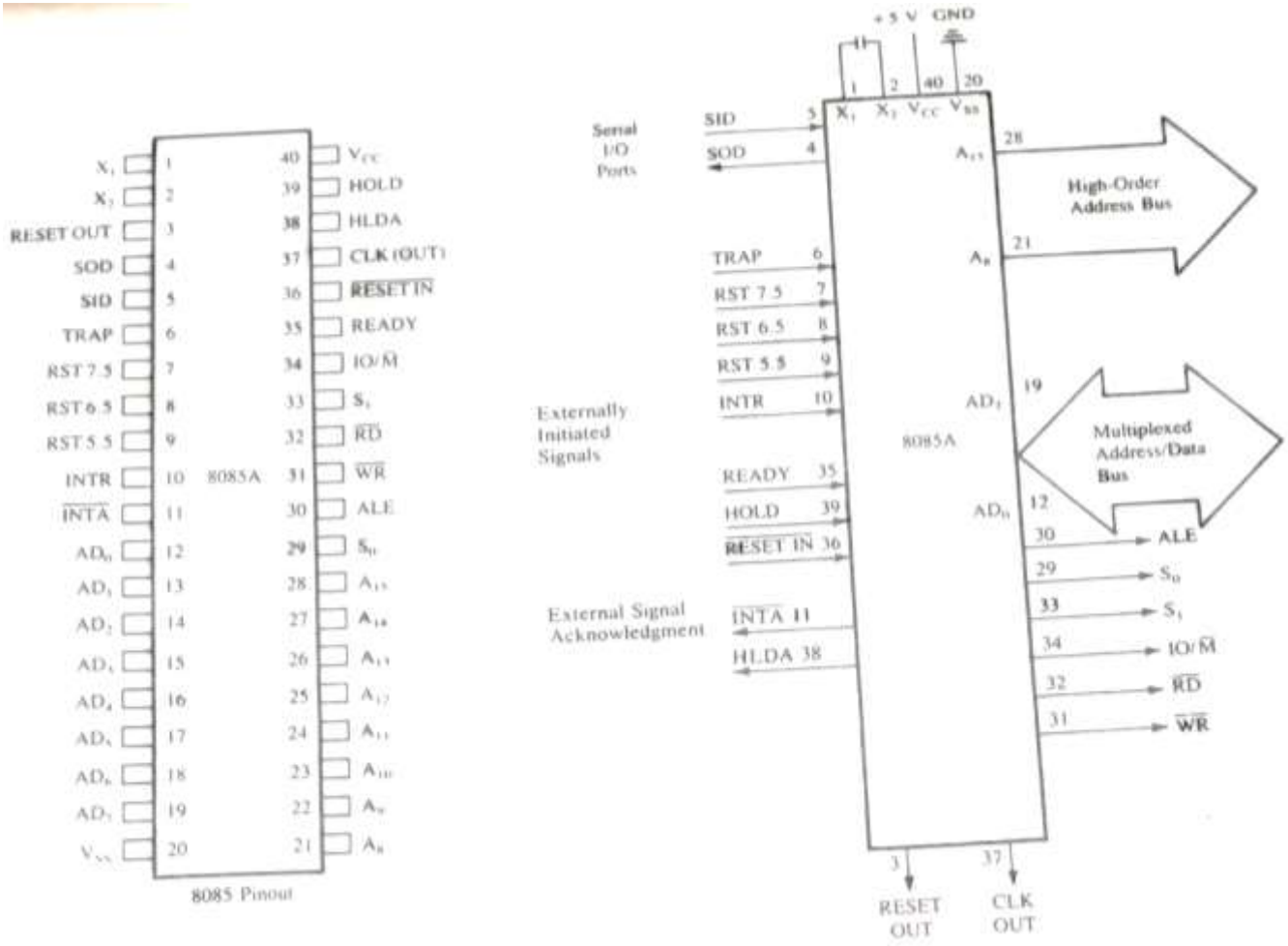


Figure 1 Microprocessor pinout and signals

CONTROL AND STATUS SIGNALS

This group of signals includes two control signals (RD' and WR'), three status signals (1O/M', S1. and S0 to identify the nature of the operation, and one special signal (ALE) to indicate the beginning of the operation. These signals are as follows:

1) ***ALE-Address Latch Enable:*** This is a positive going pulse generated every time the 8085 begins an operation (machine cycle); it indicates that the bits on AD7-AD0, are address bits. This signal is used primarily to latch the low-order address from the multiplexed bus and generate a separate set of eight address lines, A7-A0.
2) ***RD'-Read:*** This is a Read control signal (active low). This signal indicates that the selected 1/O or memory device is to be read and data are available on the data bus.

3) **WR'-Write:** This is a Write control signal (active low). This signal indicates that the data on the data bus are to be written into a selected memory or I/O location.

4) **IO/M':** This is a status signal used to differentiate between I/O and memory operations. When it is high, it indicates an I/O operation; when it is low, it indicates a memory operation. This signal is combined with RD' (Read) and WR' (Write) to generate I/O and memory control signals.

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

5) ***S1, and S0:*** These status signals, similar to IO/M', can identify various operations, but they are rarely used in small systems. (All the operations and their associated status signals are listed in **Table1** for reference.)

POWER SUPPLY AND CLOCK FREQUENCY The power supply and frequency signals are as follows:

b) Vcc: +5V power supply.
c) Vss: Ground Reference.
d) X1, X2: A crystal (or RC, LC network) is connected at these two pins. The frequency is internally divided by two; therefore, to operate a system at 3 MHz, the crystal should have a frequency of 6 MHz.
e) **CLK (OUT)-**Clock Output: This signal can be used as the system clock devices.

TABLE 1: Microprocessor Machine cycle and control signals

Machine Cycle	Status			Control Signals
	IO/$\overline{M}$	S_1	S_0	
Opcode Fetch	0	1	1	$\overline{RD} = 0$
Memory Read	0	1	0	$\overline{RD} = 0$
Memory Write	0	0	1	$\overline{WR} = 0$
I/O Read	1	1	0	$\overline{RD} = 0$
I/O Write	1	0	1	$\overline{WR} = 0$
Interrupt Acknowledge	1	1	1	$\overline{INTA} = 0$
Halt	Z	0	0	$\overline{RD}$, $\overline{WR}$ = Z and $\overline{INT}$
Hold	Z	X	X	
Reset	Z	X	X	

NOTE: Z = Tri-state (high impedance)
X = Unspecified

EXTERNALLY INITIATED SIGNALS, INCLUDING INTERRUPTS

The 8085 has five interrupt signals (see Table 4.2) that can be used to interrupt a program execution. One of the signals. INTR (Interrupt Request), is identical to the 8080A microprocessor interrupt signal (INT); the others are enhancements to the 8080A. The microprocessor acknowledges an interrupt request by the INTA (Interrupt Acknowledge) signal. In addition to the interrupts, three pins-RESET. HOLD, and READY-accept the externally initiated signals as inputs. To respond to the HOLD request, the 8085 has one signal called HLDA (Hold Acknowledge).

RESET IN: When the signal on this pin goes low, the program counter is set to zero, the buses are tri-stated, and the MPU is reset.

RESET OUT: This signal indicates that the MPU is being reset. The signal can be used to reset other devices.

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

SERIAL I/O PORTS

The 8085 has two signals to implement the serial transmission: SID (Serial Input Data) and SOD (Serial Output Data). In serial transmission, data bits are sent over a single line, one bit at a time, such as the transmission over telephone lines. This will be discussed in 16 on serial I/O.

8085 Interrupts and Externally initiated Signals

INTR (Input); Interrupt Request: This is used as a general-purpose interrupt. it is similar to the INT signal of the 8080A.

INTA' (Output) :Interrupt Acknowledge: This is used to acknowledge an interrupt.

RST 7.5 (Inputs)RST6.5, RST5.5 : Restart Interrupts: These are vectored interrupts that transfer the program control to specific memory locations. They have higher priorities than the INTR interrupt. Among these three, the priority order is 7.5. 6.5, and 5.5.

HOLD (Input): This signal indicates that a peripheral such as a DMA (Direct Memory Access) controller is requesting the use of the address and data buses.

HLDA (Output): Hold Acknowledge: This signal acknowledges the HOLD request.

READY (Input): This signal is used to delay the microprocessor Read or Write cycles until a slow-responding peripheral is ready to send or accept data. When this signal goes low, the microprocessor waits for an integral number of clock cycles until it goes high.

MICROPROCESSOR COMMUNICATION AND BUS TIMING

To understand the functions of various signals of the 8085, we will examine the process of communication (reading or writing into memory) between the microprocessor and memory and the timings of the signals in relation with the system clock. The first step in the communication process is reading from memory or fetching an instruction. This can be easily understood by an analogy of how a package is picked from your house by shipping company such as DTDC express.

The steps are as follows:

1. A courier gets the address from the office; he or she drives the pickup van, finds the street, and looks for your house number.
2. The courier rings the bell.
3. Somebody in the house opens the door and gives the package to the courier, and the courier returns to the office with the package.
4. The internal office staff disposes the package according to the instructions given by the customer Now let us examine the steps in the following example of how the microprocessor fetches or gets a machine code from memory.

Example: Illustrate the steps and the timing of data flow when the instruction code 0100 1111 (4FH-MOV C.A), stored in location 2005H, is being fetched.

To fetch the byte (4FH), the MPU needs to identify the memory location 2005H and enable the data flow from memory. This is called the Fetch cycle. The data flow is shown in Figure 2, and the timings are explained below.

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

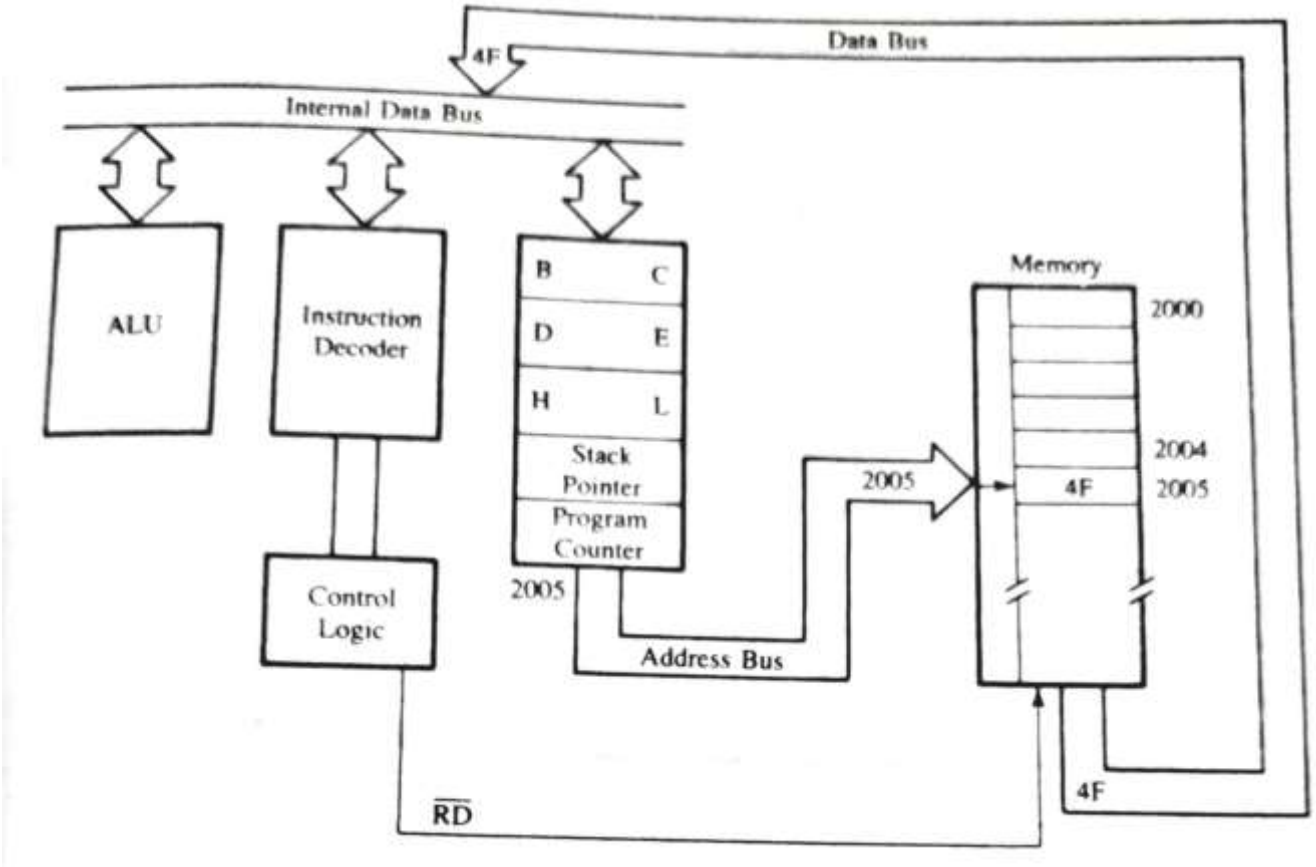


FIGURE. 2 : Data Flow from Memory to the MPU

Figure 3 shows the timing of how a data byte is transferred from memory to the MPU: it shows five different groups of signals in relation to the system clock. The address bus and data bus are shown as two parallel lines. This is a commonly used practice to represent logic levels of groups of lines; some lines are high and others are low. The crossover of the lines indicates that a new byte (information) is placed on the bus, and a dashed straight line indicates the high impedance state.

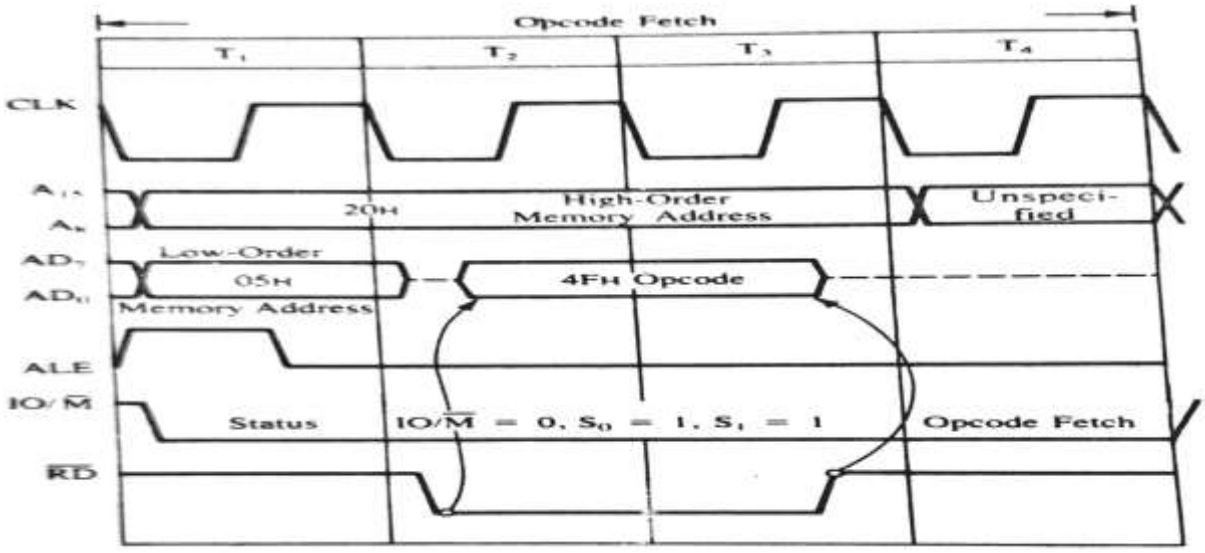


Figure 3 : Timing:transfer byte from memory to MPU

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

To fetch the byte, the MPU per- forms the following steps:

Step 1: The microprocessor places the 16-bit memory address from the program counter (PC) on the address bus (Figure 2). In our analogy, this is the equivalent of our courier getting on the road to find the address.

Figure 3 shows that at T1, the high-order memory address 20H is placed on the address lines A15-A8, the low-order memory address 05H is placed on the bus AD7-AD0 and the ALE signal goes high. Similarly, the status signal IO/M' goes low, indicating that this is a memory-related operation. (For the sake of clarity, the other two status signals, S1 and S0. are not shown in Figure 3;

Step 2: The control unit sends the control signal RD to enable the memory chip (Figure 2). This is similar to ringing the doorbell in our analogy of a package pickup. The control signal RD' is sent out during the clock period T2, thus enabling the memory chip (Figure 3). The RD' signal is active during two clock periods.

Step 3: The byte from the memory location is placed on the data bus.

When the memory is enabled, the instruction byte (4FH) is placed on the bus AD7-AD0 and transferred to the microprocessor. The RD' signal causes 4FH to be placed on bus AD7-AD0, (shown by the arrow), and when RD' goes high, it causes the bus to go into high impedance.

Step 4: The byte is placed in the instruction decoder of the microprocessor, and the task is carried out according to the instruction. The machine code or the byte (4FH) is decoded by the instruction decoder, and the contents of the accumulator are copied into register C. This task is performed during the period T4, in Figure 3.

The above four steps are similar to the steps listed in our analogy of the package pickup.

DEMULTIPLEXING THE BUS AD7-AD0

The need for demultiplexing the bus AD7-AD0, becomes easier to understand after examining Figure 3. This figure shows that the address on the high-order bus (20H) remains on the bus for three clock periods. However, the low-order address (05H) is lost after the first clock period. This address needs to be latched and used for identify7-ing the memory If the bus AD7-AD0 is used to identify the memory location (2005), the address will change to 204FH after the first clock period. Figure 4 shows a schematic that uses a latch and the ALE signal to demultiplex the bus. The bus AD7- AD0, is connected as the input to the latch 74LS373. The ALE signal is connected to the Enable (G) pin of the latch, and the Output control (OC') signal of the latch is grounded.
Figure 3 shows that the ALE goes high during T1. When the ALE is high, the latch is transparent; this means that the output changes according to input data. During T1 the output of the latch is 05H. When the ALE goes low, the data byte 05H is latched until the next ALE, and the output of the latch represents the low-order address bus A7-A0 after the latching operation.

After carefully examining Figure 3, we can make the following observations the ALE signal.
1. The machine code 4FH (0100 1000) is a one-byte instruction that copies the contents of the accumulator into register C.

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

2. The 8085 microprocessor requires one external operation-fetching a machine code from memory location 2005H.
3. . The entire operation-fetching, decoding, and executing requires four clock periods
Now we can define three terms-instruction cycle, machine cycle, and T-state-and use these terms later for examining timings of various 8085 operations .
Instruction cycle is defined as the time required to complete the execution of an instruction.
The 8085 instruction cycle consists of one to six machine cycles or one to six operations.

Machine cycle is defined as the time required to complete one operation of accessing memory, I/O, or acknowledging an external request. This cycle may consist of three to six T-states.
In Figure 3, the instruction cycle and the machine cycle are the same.
T-state is defined as one subdivision of the operation performed in one clock period. These subdivisions are internal states synchronized with the system clock, and each T-state is precisely equal to one clock period. The terms T-state and clock period are often used synonymously.

GENERATING CONTROL SIGNALS

Figure 3 shows the RD' (Read) as a control signal. Because this signal is used both for reading memory and for reading an input device, it is necessary to generate two different Read signals: one for memory and another for input. Similarly, two separate Write signals must be generated.

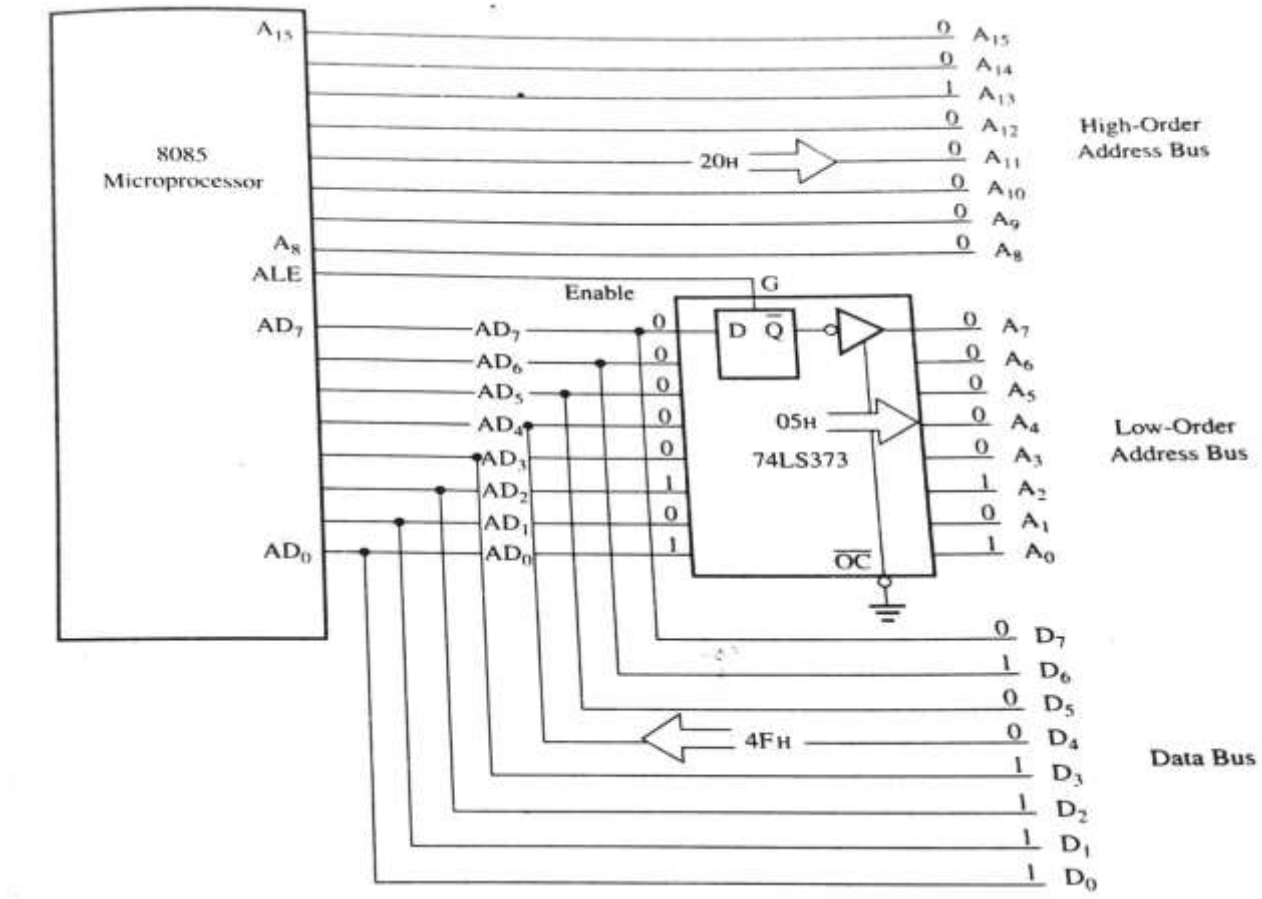


Figure 4 Schematic of Latching Low-Order Address Bus

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

Figure 5 shows that four different control signals are generated by combining the signals RD', WR', and IO/M'. The signal IO/M' goes low for the memory operation. This signal is ANDed with RD and WR signals by using the 74LS32 quadruple two-input OR gates, as shown in Figure 5. The OR gates are functionally connected as negative NAND gates. When both input signals go low, the outputs of the gates go low and generate MEMR' (Memory Read) and MEMW' (Memory Write) control signals. When the IO/M' signal goes high, it indicates the peripheral I/O operation. Figure 5 shows that this signal is complemented using the Hex inverter 74LS04 and ANDed with the RD' and WR' signals to generate IOR (I/O Read) and IOW (I/O Write) control signals.

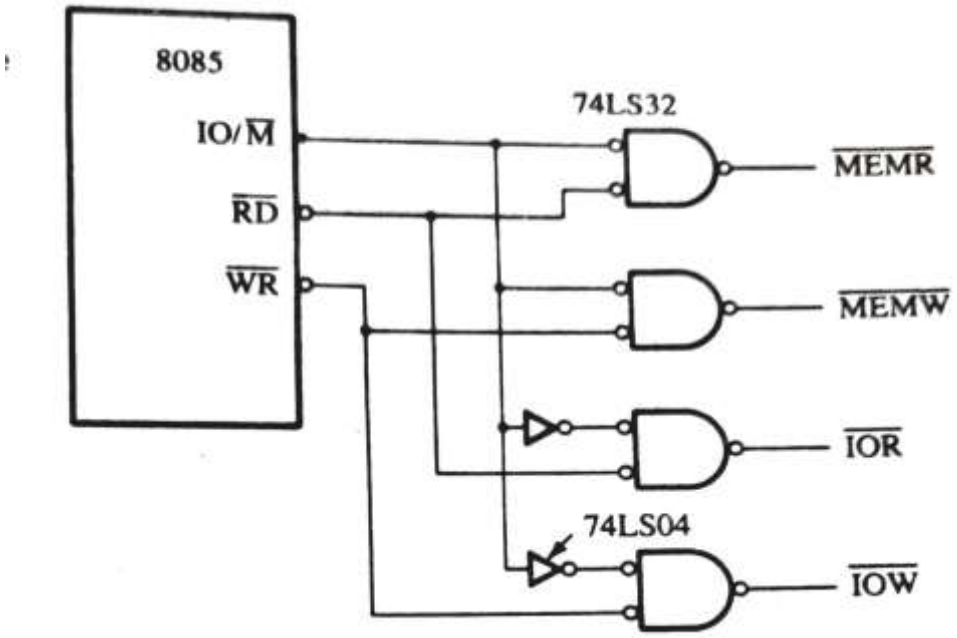


Figure. 5 Schematic to Generate Read/Write Control Signals for Mmeory and I/O

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

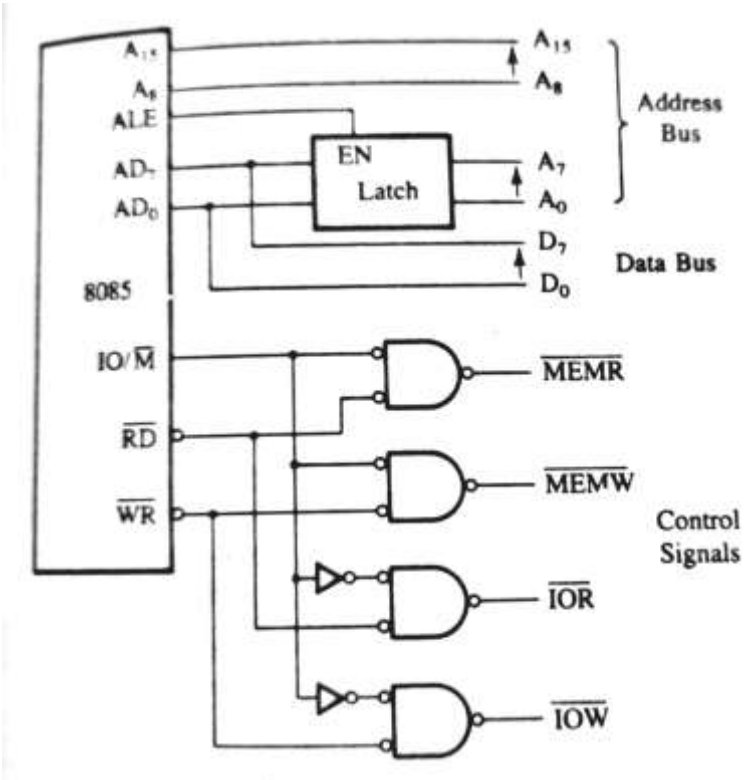


FIGURE 6: 8085 Demultiplexed Address and Data Bus with Control Signals

To demultiplex the bus and to generate the necessary control signals, the 8085 microprocessor requires a latch and logic gates to build the MPU, as shown in Figure 6. This MPU can be interfaced with any memory or I/O.

A DETAILED LOOK AT THE 8085 MPU AND ITS ARCHITECTURE

Figure 7 shows the internal architecture of the 8085 beyond the programmable registers we discussed previously. It includes the ALU (Arithmetic/Logic Unit). Timing and Control Unit. Instruction Register and Decoder, Register Array, Interrupt Control, and Serial I/O Control.

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

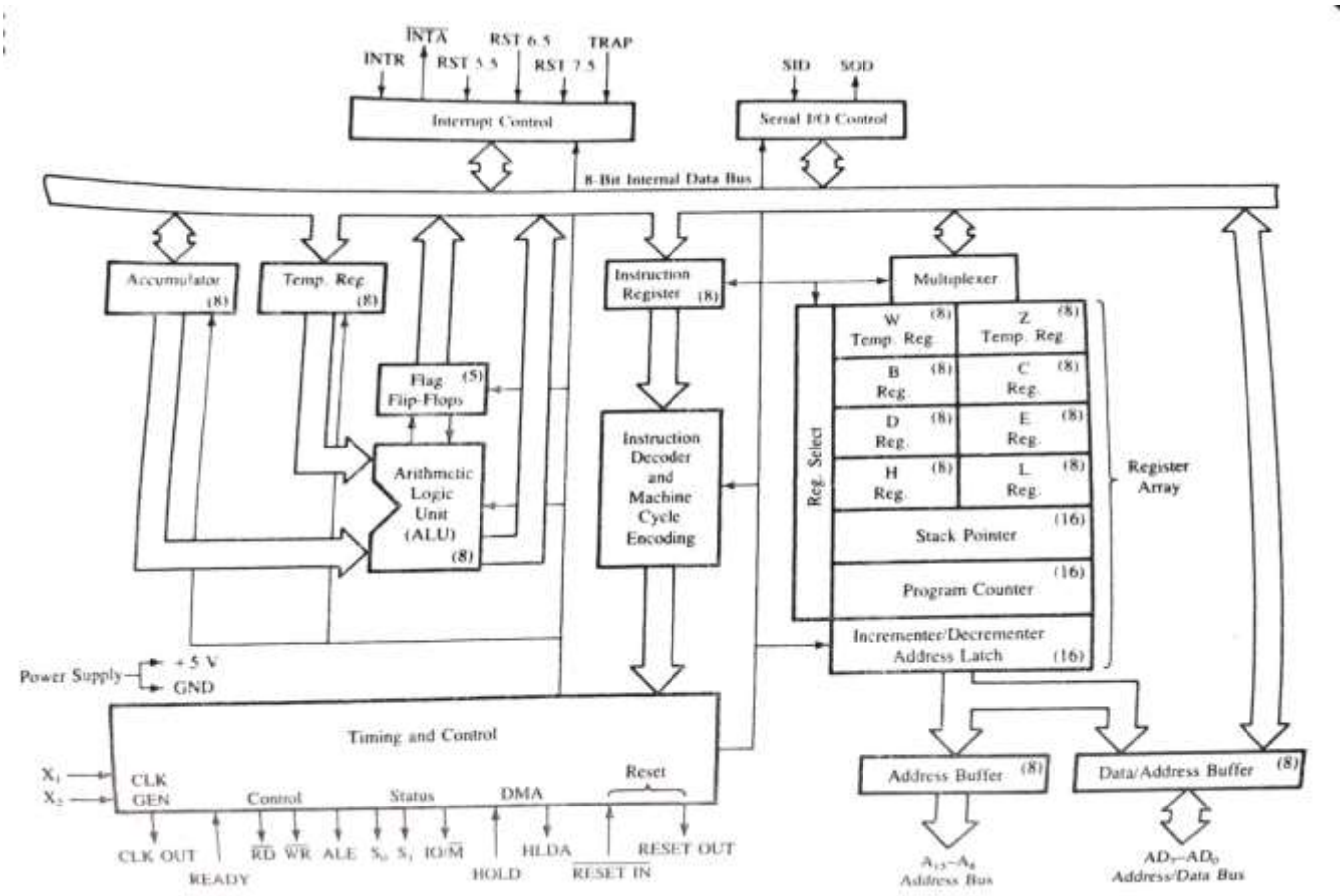


Figure. 7 The 8085A Microprocessor: Functional Block Diagram

THE ALU

The arithmetic/logic unit performs the computing functions; it includes the accumulator. the temporary register, the arithmetic and logic circuits, and five flags. The temporary register is used to hold data during an arithmetic/logic operation. The result is stored in the accumulator, and the flags flip-flops) are set or reset according to the result of the operation. The flags are affected by the arithmetic and logic operations in the ALU. In most of these operations, the result is stored in the accumulator. Therefore, the flags generally reflect data conditions in the accumulator-with some exceptions. The descriptions and conditions of the flags are as follows:

S-Sign flag: After the execution of an arithmetic or logic operation, if bit D7, of the result (usually in the accumulator) is 1. the Sign flag is set. This flag is used with signed numbers. In a given byte, if D7 is 1. the number will be viewed as a negative number; if it is 0, the number will be considered positive. In arithmetic operations with signed numbers, bit D7, is reserved for indicating the sign, and the remaining seven bits are used to represent the magnitude of a number. However, this flag is irrelevant for the operations of unsigned numbers. Therefore, for unsigned numbers, even if bit D7, of a result is I and the flag is set, it does not mean the result is negative.

Z-Zero flag: The Zero flag is set if the ALU operation results in 0. and the flag is reset if the result is not 0. This flag is modified by the results in the accumulator as well as in the other registers.

AC-Auxiliary Carry flag: In an arithmetic operation, when a carry is generated by digit D3, and passed on to digit D4., the AC flag is set. The flag is used only internally for BCD (binary-coded decimal)

operations and is not available for the programmer to change the sequence of a program with a jump instruction.

P-Parity flag: After an arithmetic or logical operation, if the result has an even number of Is, the flag is set. If it has an odd number of Is, the flag is reset. (For example. the data byte 0000 0011 has even parity even if the magnitude of the number is odd.)

CY-Carry flag: If an arithmetic operation results in a carry, the Carry flag is set: otherwise it is reset. The Carry flag also serves as a borrow flag for subtraction.

The bit positions reserved for these flags in the flag register are as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z		AC		P		CY

Among the five flags, the AC flag is used internally for BCD arithmetic: the instruction set does not include any conditional jump instructions based on the AC flag. Of the remaining four flags, the Z and CY flags are those most commonly used.

TIMING AND CONTROL UNIT

This unit synchronizes all the microprocessor operations with the clock and generates the control signals necessary for communication between the microprocessor and peripherals. The control signals are similar to a sync pulse in an oscilloscope. The RD' and WR' signals are sync pulses indicating the availability of data on the data bus.

INSTRUCTION REGISTER AND DECODER

The instruction register and the decoder are part of the ALU. When an instruction is fetched from memory, it is loaded in the instruction register. The decoder decodes the instruction and establishes the sequence of events to follow. The instruction register is not programmable and cannot be accessed through any instruction.

REGISTER ARRAY

The programmable registers are used to store the data temporarily. Two additional registers, called temporary registers W and Z, are included in the register array. These registers are used to hold 8-bit data during the execution of some instructions. However, because they are used internally, they are not available to the programmer.

Decoding and Executing an Instruction

Decoding and executing an instruction after it has been fetched can be illustrated with the example
Example: Assume that the accumulator contains data byte 82H, and the instruction MOV C, A (4FH) is List the steps in decoding and executing the instructio. To decode and execute the instruction, the following steps are performed.

The microprocessor.

1. Places the contents of the data bus (4FH) in the instruction register (Figure 8) and decodes the instruction.
2. Transfers the contents of the accumulator (82H) to the temporary register in the ALU,
3. Transfers the contents of the temporary register to register C.

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

4.The 8085 MPU

- transfers data from memory locations to the microprocessor by using the control signal Memory Read (MEMR-active low). This is also called reading from memory. The term data refers to any byte that is placed on the data bus; the byte can be an instruction code, data, or an address.
- transfers data from the microprocessor to memory by using the control signal Memory Write (MEMW-active low). This is also called writing into memory.
- accepts data from input devices by using the control signal I/O Read (IOR-active low). This is also known as reading from an input port.
- sends data to output devices by using the control signal 1/O Write (10W-active low). This is also known as writing to an output port.

5. To execute an instruction, the MPU

- places the memory address of the instruction on the address bus.
- indicates the operation status on the status lines.
- sends the MEMR control signal to enable the memory, fetches the instruction byte. and places it in the instruction decoder.
- Executes the instruction.

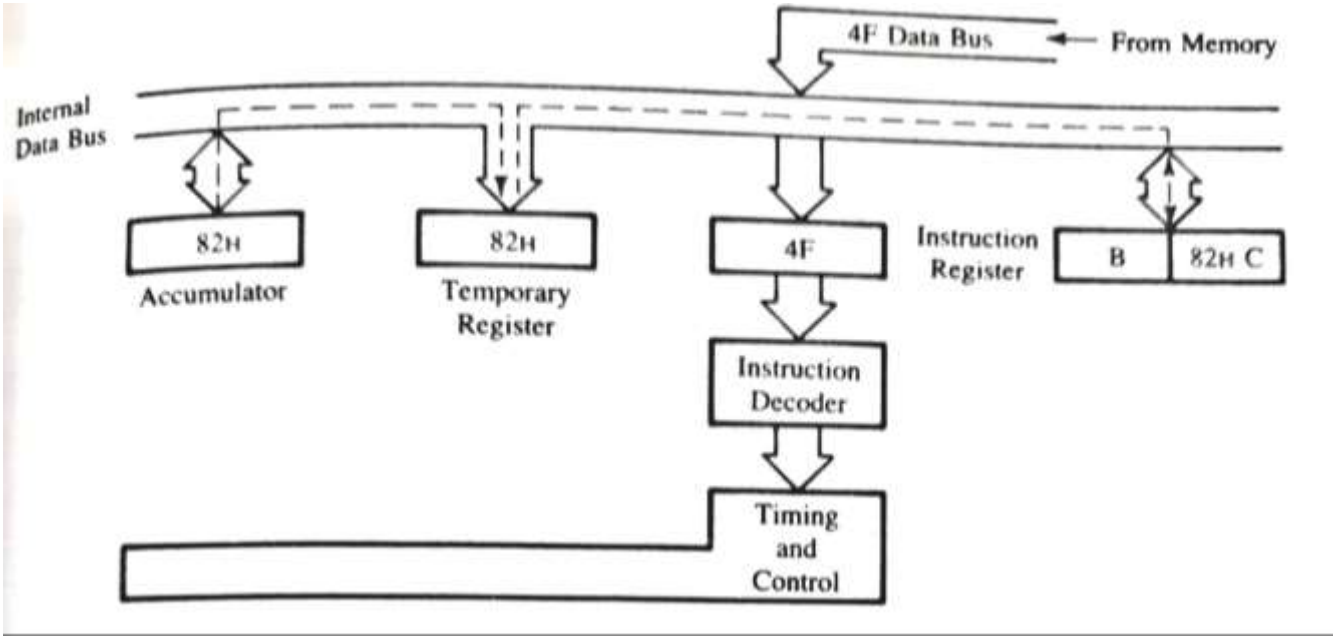


FIGURE .8 Instruction Decoding and Execution

DATA FORMATS:

The operand is an another name for data. It may appear in different forms

- Addresses
- Numbers/Logical data and
- Characters

Addresses: The address is a 16-bit unsigned integer, number used to refer a memory location.

Numbers/Data: The 8085 supports following numeric data types.

Siddartha Institute of Science and Technology::Puttur
(Autonomous)
(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

- Signed Integer: A signed integer number is either a positive number or a negative number. In 8085, 8- bits are assigned for signed integer, in which most significant bit is used for sign and remaining seven bits are used for Sign bit 0 indicates positive number whereas sign bit 1 indicates negative number.
- Unsigned Integer: The 8085 microprocessor supports 8-bit unsigned integer.
- BCD: The term BCD number stands for binary coded decimal number. It uses ten digits from 0 through 9. The 8-bit register of 8085 can store two digit BCD

Characters: The 8085 uses ASCII code to represent characters. It is a 7-bit alphanumeric code that represents decimal numbers, English alphabets, and other special characters.

DATA STORAGE:

For storing data and Instructions Data storage known as memory are used with microprocessor. Memory is a storage of binary bits. Memory chips used in most systems are 8-bit registers stacked one above the other. It includes four registers and each register can store 8 bits. This chip can be refered to as a 4-byte or 32 bit (4X8) bits memory chip.It has two address lines A0 and A1 to identify four registers, 8 data lines to store 8 bits and three timing control signals: Read (RD'), Write (WR') and Chip select (CS'): all control signals are designed to be active low. The processor can select this chip and identify its register and store (write) or access (read) 8 bit at a time.The memory addresses assigned to these registers are determined by the interfacing logic used in the system.

OVERVIEW OF THE 8085 INSTRUCTION SET

The 8085 microprocessor instruction set has 74 operation codes that result in 246 in- structions. The set includes all the 8080A instructions plus two additional instructions (SIM and RIM, related to serial I/O). It is an overwhelming experience for a beginner to study these instructions. You are strongly advised not to attempt to read all these instructions at one time. However, you should be able to grasp an overview of the set by exam ining the frequently used instructions listed below The following notations are used in the description of the instructions

R = 8085 8-bit register (A. B. C. D. E. H. L)
M = Memory register (location)
Rs = Register source
Rd = Register destination (A, B, C. D. E, H., L.)
Rp = Register pair (BC. DE, HL, SP)
() = Contents of

1. Data Transfer (Copy) Instructions: These instructions perform the following six operations.

(i) Load an 8-bit number in a register
(ii) Copy from register to register
(iii) Copy between I/O and accumulator
(iv) Load 16-bit number in a register pair
(v) Copy between register and memory
(vi) Copy between registers and stack memory

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

Mnemonics	Examples	Operration
1.1 MVI R. 8-bit	MVI B, 4FH	Load an 8-bit data (byte) in a register
1.2 MOV Rd, Rs**	MOV B. A MOV C. B	Copy data from source register Rs into destination register Rd
1.3 LXI Rp. 16-bit	LXI B, 2050H	Load 16-bit number in a register pair
1.4 OUT 8-bit (port address)	OUT 01H	Send(write) data byte from the accumulator to an \ output device
1.5 IN 8-bit (port address)	IN 07H	Accept (read) data byte from an input device and place it in the accumulator
1.6 LDA 16-bit	LDA 2050H	Copy the data byte into A from the memory specified by 16-bit address
1.7 STA 16-bit	STA 2070H	Copy the data byte from A into the memory specified by 16-bit address
1.8 LDAX Rp	LDAX B	Copy the data byte into A from the memory specified by register pair
1.9 STAX Rp	STAX D	Copy the data byte from A into the memory specified by register pair
1.10 MOV R, M	MOV B,M	Copy the data byte into register from the memory address specified in HL register pair
1.11 MOV M,R	MOV M,C	Copy the data byte from register into the memory address specified in HL register pair

2. Arithmetic Instruction: The frequently used arithmetic operations are:

1. Add 2) Subtract 3) Increment 4) Decrement

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

	Mnemonics	Examples	Operation
2.1	ADD R	ADD B	Add the contents of a register to the register to the contents of A
2.2	ADI 8-bit	ADI 37H	Add 8-bit data to the contents of A
2.3	ADD M	ADD M	Add the contents of memory to A; the address of memory is in HL register
2.4	SUB R	SUB C	Subtract the contents of a register from the contents of A
2.5	SUI 8-bit	SUI 7FH	Subtract 8-bit data from the contents of A
2.6	SUB M	SUB M	Subtract the contents of memory from A; the address of memory is in HL register
2.7	INR R	INR D	Increment the contents of a register
2.8	INR M	INR M	Increment the contents of memory, the address of which is in HL
2.9	DCR R	DCR E	Decrement the contents of a register
2.10	DCR M	DCR M	Decrement the contents of a memory, the address of which is in HL
2.11	INX Rp	INX H	Increment the contents of a register pair
2.12	DCX Rp	DCX B	Decrement the contents of a register pair

3. Logic and Bit Manipulation Instructions. These instructions include the followin operations:

- AND
- OR
- X-OR (Exclusive OR)
- Compare
- Rotate Bit

	Mnemonics	Examples	Operation
3.1	ANA R	ANA B	Logically AND the contents of a register with the contents of A

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

3.2	ANI 8-bit	ANI 2FH	Logically AND 8-bit data with the contents of A
3.3	ANA M	ANA M	Logically AND the contents of memory with the contents of A; the address of memory is in HL register
3.4	ORA R	ORA E	Logically OR the contents of a register with the contents of A
3.5	ORI 8-bit	ORI 3FH	Logically OR 8-bit data with the contents of A
3.6	ORA M	ORA M	Logically OR the contents of memory with the contents of A; the address of memory is in HL register
3.7	XRA R	XRA B	Exclusive-OR the contents of a register with the contents of A
3.8	XRI 8-bit	XRI 6AH	Exclusive-OR 8-bit data with the contents of A
3.9	XRA M	XRA M	Exclusive-OR the contents of memory with the contents of A; the address of memory is in HL register
3.10	CMP R	CMP B	Compare the contents of register with the contents of A for less than, equal to, or greater than
3.11	CPI 8-bit	CPI 4FH	Compare 8-bit data with the contents of A for less than, equal to, or greater than

4. Branch Instructions. The following instructions change the program sequence.

4.1	JMP 16-bit address	JMP 2050H	Change the program sequence to the specified 16-bit address
4.2	JZ 16-bit address	JZ 2080H	Change the program sequence to the specified 16-bit address if the Zero flag is set
4.3	JNZ 16-bit address	JNZ 2070H	Change the program sequence to the specified 16-bit address if the Zero flag is reset
4.4	JC 16-bit address	JC 2025H	Change the program sequence to the specified 16-bit address if the Carry flag is set
4.5	JNC 16-bit address	JNC 2030H	Change the program sequence to the specified 16-bit address if the Carry flag is reset
4.6	CALL 16-bit address	CALL 2075H	Change the program sequence to the location of a subroutine
4.7	RET	RET	Return to the calling program after completing the subroutine sequence

Siddartha Institute of Science and Technology::Puttur

(Autonomous)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
(NAAC Accredited Institution with 'A' Grade)
SIDDHARTH NAGAR, NARAYANAVANAM ROAD, PUTTUR – 517 583
CHITTOOR DIST., A.P., INDIA

5. Machine Control Instructions. These instructions affect the operation of the processor.

5.1	HLT	HLT	Stop processing and wait
5.2	NOP	NOP	Do not perform any operation

This set of instructions is a representative sample; it does not include various instructions related to 16-bit data operations, additional jump instructions, and conditional Call and Return instructions.

UNIT -III

THE 8051 ARCHITECTURE

THE 8051 ARCHITECTURE 11
Introduction 11
8051 Microcontroller Hardware 11
The 8051 Oscillator and Clock 16
Program Counter and Data Pointer 17
A and B CPU Registers 17
Flags and the Program Status Word (PSW) 18
Internal Memory 19
Internal RAM 19
The Stack and the Stack Pointer 19
Special Function Registers 21
Internal ROM 22
Input/Output Pins, Ports, and Circuits 22
Port 0 23
Port 1 25
Port 2 25
Port 3 25
External Memory 26
Connecting External Memory 26
Counter and Timers 28
Timer Counter Interrupts 29
Timing 30
Timer Modes of Operation 30
Timer Mode 0 30
Timer Mode 1 30
Timer Mode 2 31
Timer Mode 3 32
Counting 32
Serial Data Input/Output 32
Serial Data Interrupts 32
Data Transmission 34
Data Reception 34
Serial Data Transmission Modes 34
Serial Data Mode 0—Shift Register Mode 34
Serial Data Mode 1—Standard UART 35
Mode 1 Baud Rates 36
Serial Data Mode 3 37
Interrupts 37
Timer Flag Interrupt 39
Serial Port Interrupt 39
External Interrupts 39
Reset 40
Interrupt Control 40
Interrupt Enable/Disable 40
Interrupt Priority 41
Interrupt Destinations 41
Software Generated Interrupts 41
Summary 41
Questions 42

8051 MICROCONTROLLER HARDWEAR

The first task faced when learning to use a new computer is to become familiar with the capability of the machine. The features of the computer are best learned by studying the internal hardware design, also called the architecture of the device, to determine the type, number, and size of the registers and other circuitry.

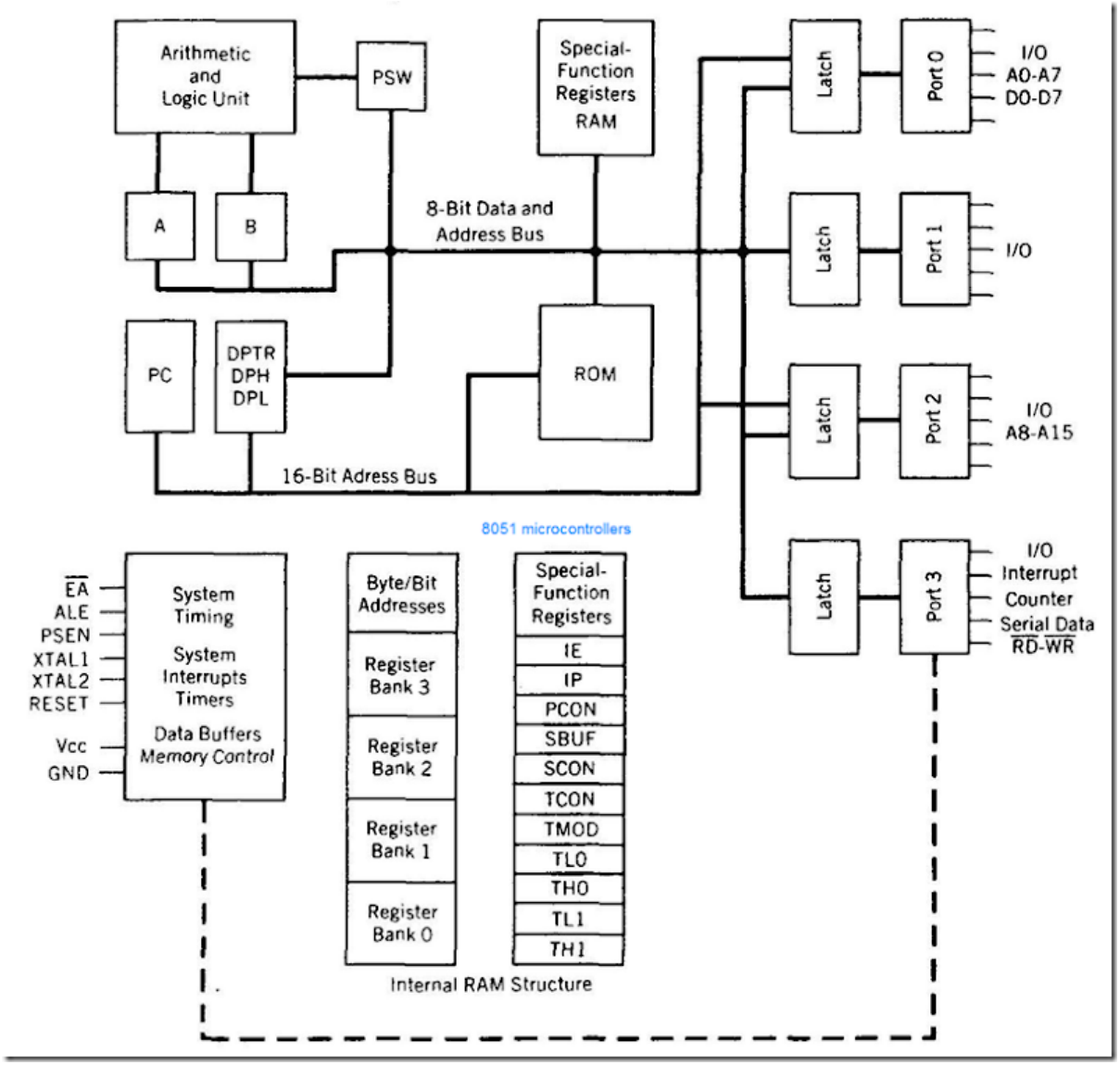
The hardware is manipulated by an accompanying set of program instructions, or software, which is usually studied next. Once familiar with the hardware and software, the system designer can then apply the microcontroller to the problems at hand.

A natural question during this process is "What do I do with all this stuff?" Similar to attempting to write a poem in a foreign language before you have a vocabulary and rules of grammar, writing meaningful programs is not possible until you have become acquainted with both the hardware and the software of a computer.

This chapter provides a broad overview of the architecture of the 8051 . In subsequent chapters, we will cover in greater detail the interaction between the hardware and the software.

8051 Microcontroller Hardware

The 8051 microcontroller actually includes a whole family of microcontrollers that have numbers ranging from 8031 to 8751 and are available in N-Channel Metal Oxide Silicon (NMOS) and Complementary Metal Oxide Silicon (CMOS) construction in a variety of package types. An enhanced version of the 8051. the 8052, also exists with its own family of variations and even includes one member that can be programmed in BASIC. An inspection of Appendix E shows that there are dozens of other variations on the “core” 8051 architecture .This galaxy of parts, the result of desires by the manufacturers to leave no market niche unfilled. would require many topics to cover.



In this topic , we will study a "generic" 8051. housed in a 40-pin DIP. and direct the investigation of a particular type to the data books. The block diagram of the 8051 in Figure l a shows all of the features unique to microcontrollers:

Internal ROM and RAM

I/O ports with programmable pins Timers and counters

Serial data communication

The figure also shows the usual CPU components: program counter, ALU, working registers, and clock circuits.

The 8051 architecture consists of these specific features:

Eight-bit CPU with registers A (the accumulator) and B

Sixteen-bit program counter (PC) and data pointer (DPTR)

Eight-bit program status word (PSW)

Eight-bit stack pointer (SP)

Internal ROM or EPROM (8751) of 0 (8031) to 4K (8051)

Internal RAM of 128 bytes:

Four register banks, each containing eight registers

Sixteen bytes, which may be addressed at the bit level

Eighty bytes of general-purpose data memory

Thirty-two input/output pins arranged as four 8-bit ports: PO-P3

Two 16-bit timer/counters: TO and TI

Full duplex serial data receiver/transmitter: SBUF

Control registers: TCON, TMOD, SCON, PCON, IP, and IE

Two external and three internal interrupt sources

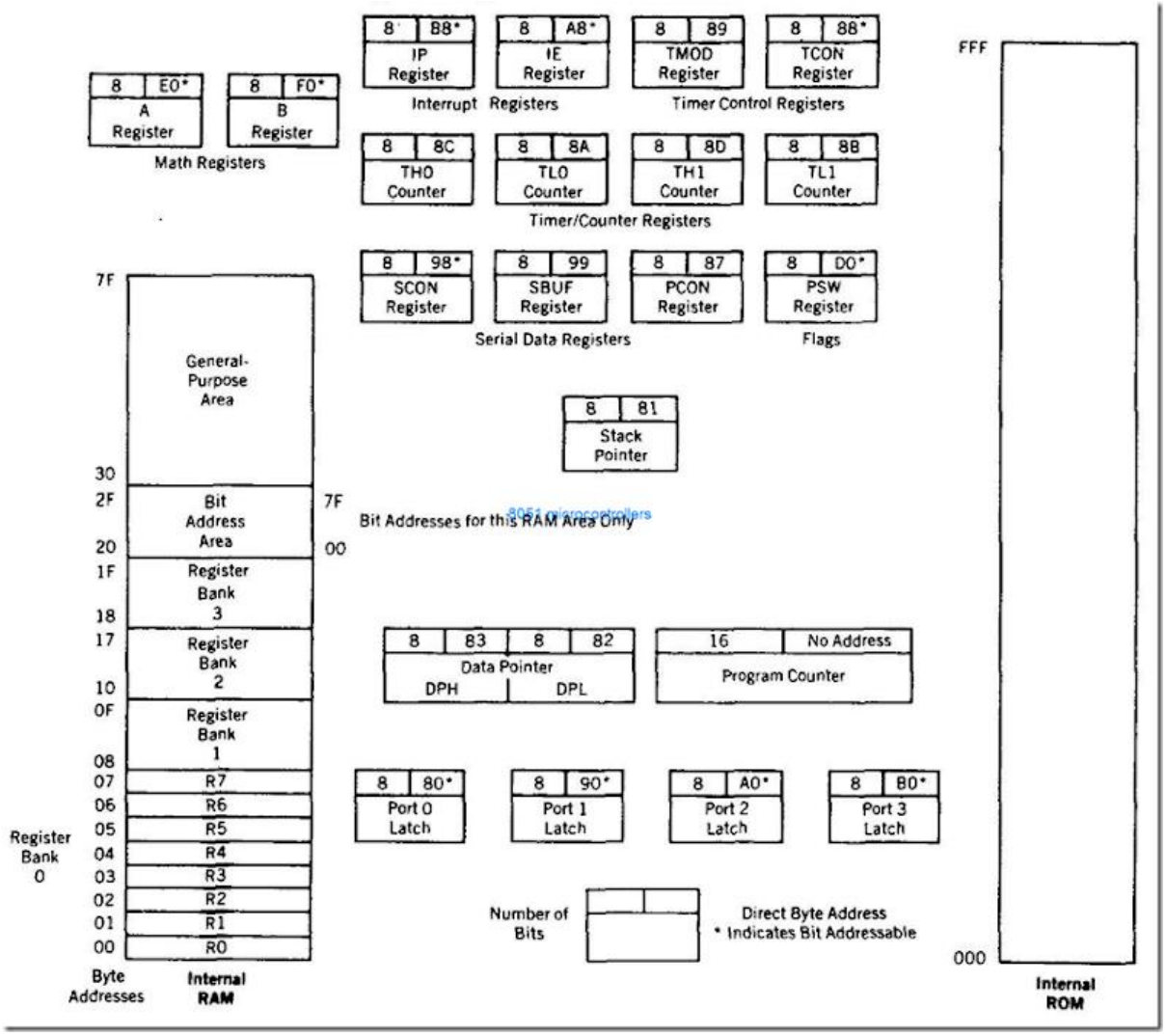
Oscillator and clock circuits . Knowledge of the details of circuit operation that cannot be affected by any instruction or external data. while intellectually stimulating. tends to confuse the student new to the 8051. For this reason. this text will concentrate on the essential features of the R051: the more advanced student may wish to refer to manufacturers' data books for additional information.

The programming model of the 8051 in Figure 1 b shows the 8051 as a collection of 8- and 16-bit registers and 8-bit memory locations. These registers and memory locations can be made to operate using the software instructions that are incorporated as part of the design. The program instructions have to do with the control of the registers and digital data paths that are physically contained inside the 8051, as well as memory locations that are physically located outside the 8051 .

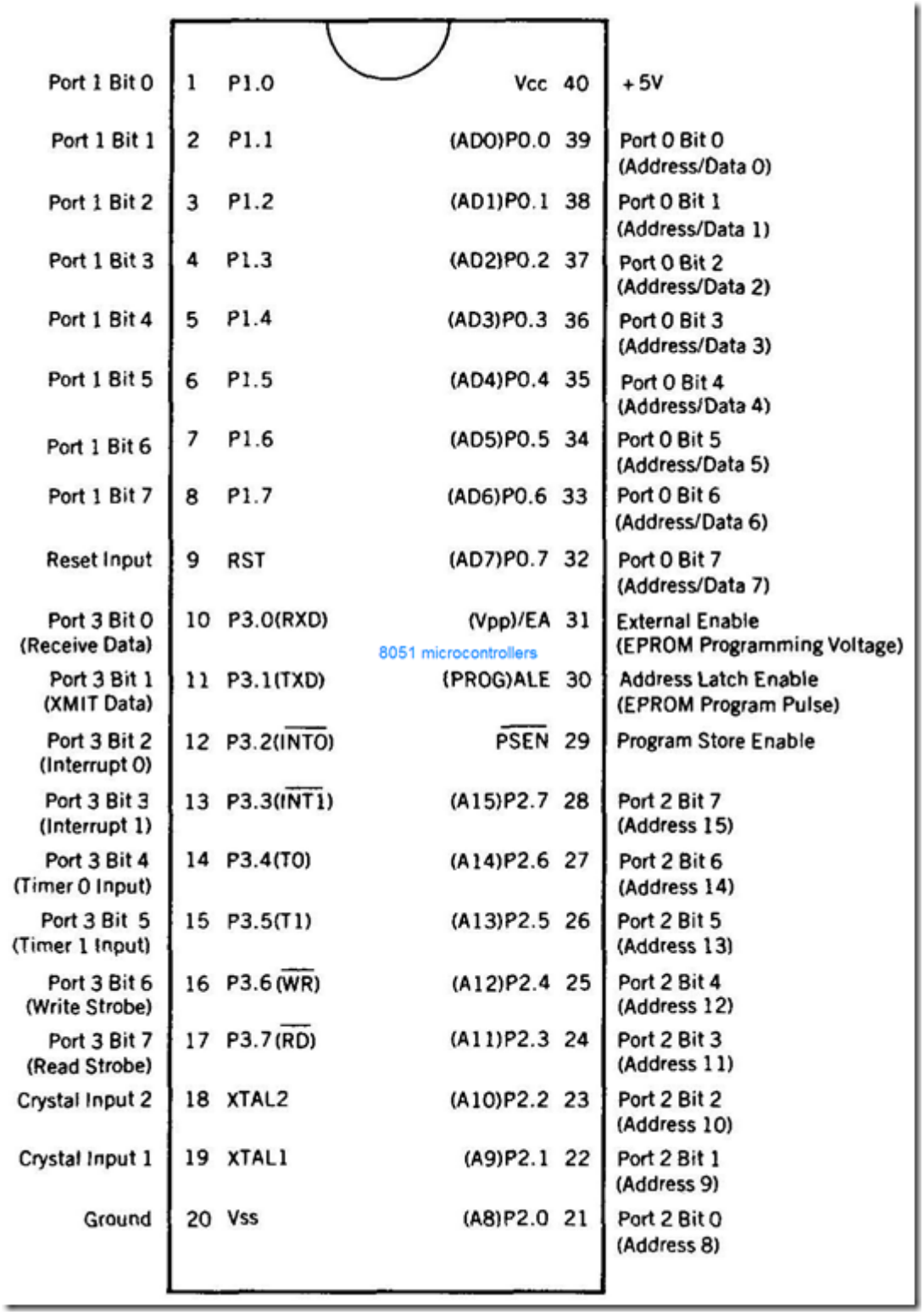
The model is complicated by the number of special-purpose registers that must be present to make a microcomputer a microcontroller. A cursory inspection of the model is recommended for the first-time viewer; return to the model as needed while progressing through the 'remainder of the text.

Most of the registers have a specific function; those that do occupy an individual block with a symbolic name, such as A or TH0 or PC. Others, which are generally indistinguishable from each other, are grouped in a larger block, such as internal ROM or RAM memory.

Each register, with the exception of the program counter, has an internal l-byte address assigned to it. Some registers (marked with an asterisk * in Figure 1b) are both byte and bit addressable. That is, the entire byte of data at such register addresses may be read or altered, or individual bits may be read or altered. Software instructions are generally able to specify a register by its address, its symbolic name, or both.



A pinout of the 8051 packaged in a 40-pin DIP is shown in Figure 2 with the full and abbreviated names of the signals for each pin. It is important to note that many of the pins are used for more than one function (the alternate functions are shown in parentheses in Figure 2). Not all of the possible 8051 features may be used at the same time.

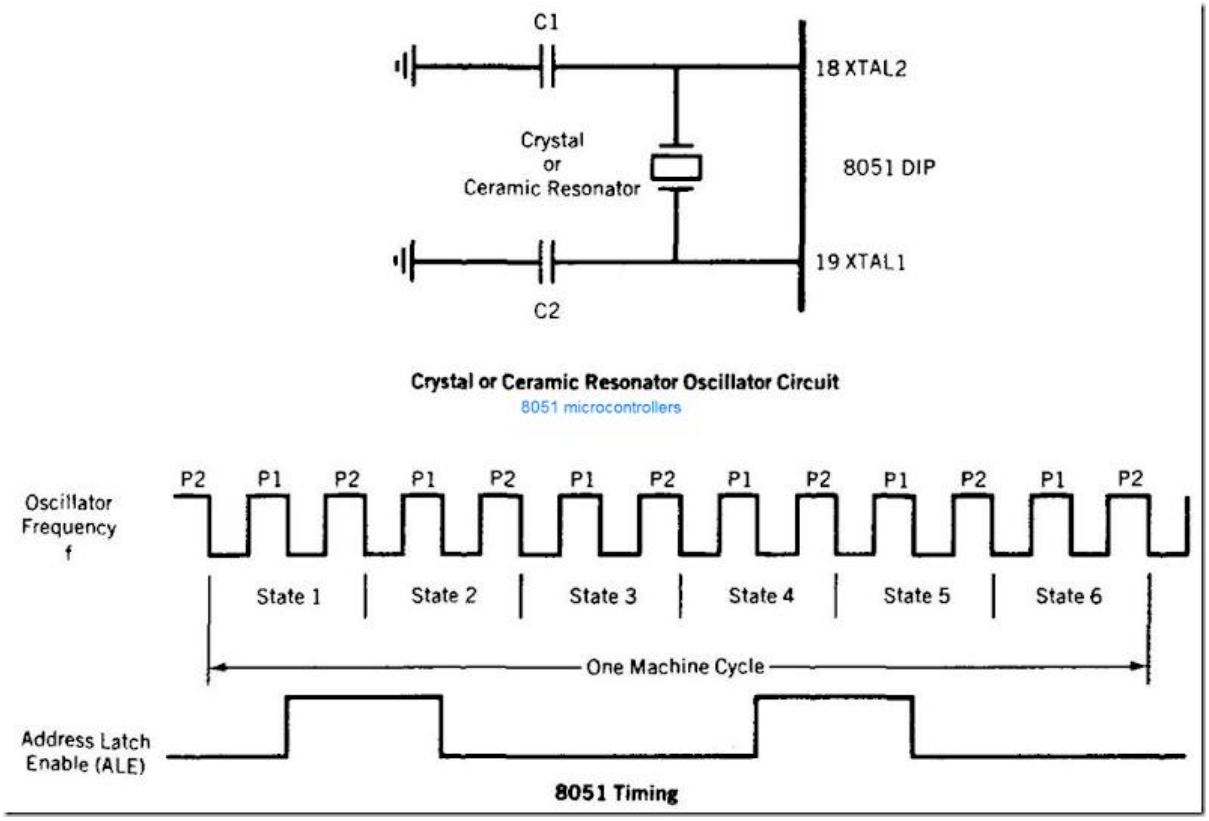


Programming instructions or physical pin connections determine the use of any multifunction pins. For example, port 3 bit 0 (abbreviated P3.0) may be used as a general-purpose I/O pin, or as an input (RXD) to SBUF, the serial data receiver register. The system designer decides which of these two functions is to be used and designs the hardware and software affecting that pin accordingly.

The 8051 Oscillator and Clock

The heart of the 8051 is the circuitry that generates the clock pulses by which all internal operations are synchronized. Pins XTALI and XTAL2 are provided for connecting a resonant network to form an oscillator. Typically, a quartz crystal and capacitors are employed, as shown

in Figure 3. The crystal frequency is the basic internal clock frequency of the microcontroller. The manufacturers make available 8051 designs that can run at specified maximum and minimum frequencies, typically I megahertz to 16 megahertz. Minimum frequencies imply that some internal memories are dynamic and must always operate above a minimum frequency, or data will be lost.



Serial data Communication needs often dictate the frequency of the oscillator due to the requirement that internal counters must divide the basic clock rate to yield standard communication bit per second (baud) rates. If the basic clock frequency is not divisible without a remainder. then the resulting communication frequency is not standard.

Ceramic resonators may be used as a low-cost alternative to crystal resonators. However, decreases in frequency stability and accuracy make the ceramic resonator a poor choice if high-speed serial data communication with other systems, or critical timing, is to be done.

The oscillator formed by the crystal, capacitors, and an on-chip inverter generates a pulse train at the frequency of the crystal, as shown in Figure 3.

The clock frequency, *f*. establishes the smallest interval of time within the microcontroller, called the pulse, P, time. The smallest interval of time to accomplish any simple instruction, or part of a complex instruction, however, is the machine cycle. The machine cycle is itself made up of six states. A state is the basic time interval for discrete operations of the microcontroller such as fetching an opcode byte, decoding an opcode, executing an opcode, or writing a data byte. Two oscillator pulses define each state.

Program instructions may require one, two, or four machine cycles to be executed. depending on the type of instruction. Instructions are fetched and executed by the microcontroller automatically, beginning with the instruction located at ROM memory address 0000h at the time the microcontroller is first reset.

To calculate the time any particular instruction will take to be executed, find the number of cycles, C. from the list in Appendix A. The time to execute that instruction is then found by multiplying C by 12 and dividing the product by the crystal frequency:

For example, if the crystal frequency is 16 megahertz. then the time to execute an ADD A, RI one-cycle instruction is .75 microseconds. A 12 megahertz crystal yields the convenient time of one microsecond per cycle. An 11.0592 megahertz crystal, while seemingly an odd value, yields a cycle frequency of 921.6 kilohertz, which can be divided evenly by the standard communication baud rates of 19200, 9600, 4800, 2400, 1200, and 300 hertz.

PROGRAM COUNTER AND DATA POINTER

The 8051 contains two 16-bit registers: the program counter (PC) and the data pointer (DPTR). Each is used to hold the address of a byte in memory.

Program instruction bytes are fetched from locations in memory that are addressed by the PC. Program ROM may be on the chip at addresses 0000h to OFFFh, external to the chip for addresses that exceed 0FFFh, or totally external for all addresses from 0000h to FFFFh. The PC is automatically incremented after every instruction byte is fetched and may also be altered by certain instructions. The PC is the only register that does not have an internal address.

The DPTR register is made up of two 8-bit registers. named DPH and DPL, that are used to furnish memory addresses for internal and external code access and external data access. The DPTR is under the control of program instructions and can be specified by its 16-bit name, DPTR, or by each individual byte name, DPH and DPL. DPTR does not have a single internal address; DPH and DPL are each assigned an address.

A and B CPU Registers

The 8051 contains 34 general-purpose, or working. registers. Two of these, registers A and B, comprise the mathematical core of the 8051 central processing unit (CPU). The other 32 are arranged as part of internal RAM in four banks, B0-B3. of eight registers and comprise the mathematical core .

The A (accumulator) register is the most versatile of the two CPU registers and is used for many operations. including addition, subtraction, integer multiplication and division, and Boolean bit manipulations. The A register is also used for all data transfers between the 8051 and any external memory. The B register is used with the A register for multiplication and division operations and has no other function other than as a location where data may be stored.

Flags and the Program Status Word (PSW)

Flags are 1-bit registers provided to store the results of certain program instructions. Other instructions can test the condition of the flags and make decisions based upon the flag states. In order that the flags may be conveniently addressed, they are grouped inside the program status word (PSW) and the power control (PCON) registers.

The 8051 has four math flags that respond automatically to the outcomes of math operations and three general-purpose user flags that can be set to I or cleared to 0 by the programmer as desired. The math flags include carry (C), auxiliary carry (AC), overflow (OY), and parity (P). User flags are named FO, GFO, and GFI; they are general-purpose flags that may be used by

the programmer to record some event in the program. Note that all of the flags can be set and cleared by the programmer at will. The math Hags, however, are also affected by math operations .

The program status word is shown in Figure 4. The PSW contains the math flags, user program Hag FO, and the register select bits that identify which of the four general-purpose register banks is currently in use by the program. The remaining two user flags, GFO and GFI, are stored in PCON, which is shown in Figure 13.

7	6	5	4	3	2	1	0
CY	AC	F0	RS1	RS0	OV	—	P

THE PROGRAM STATUS WORD (PSW) SPECIAL FUNCTION REGISTER

Bit	Symbol	Function
7	CY	Carry flag; used in arithmetic, JUMP, ROTATE, and BOOLEAN instructions
6	AC	Auxilliary carry flag; used for BCD arithmetic
5	F0	User flag 0
4	RS1	Register bank select bit 1
3	RS0	Register bank select bit 0

RS1	RS0	
0	0	Select register bank 0
0	1	Select register bank 1
1	0	Select register bank 2
1	1	Select register bank 3

2	OV	Overflow flag; used in arithmetic instructions
1	—	Reserved for future use
0	P	Parity flag; shows parity of register A: 1 = Odd Parity

Bit addressable as PSW.0 to PSW.7

Detailed descriptions of the math Hag operations will be discussed in topics that cover the opcodes that affect the flags. The user flags can be set or cleared using data move instructions will be covered .

INTERNAL MEMORY

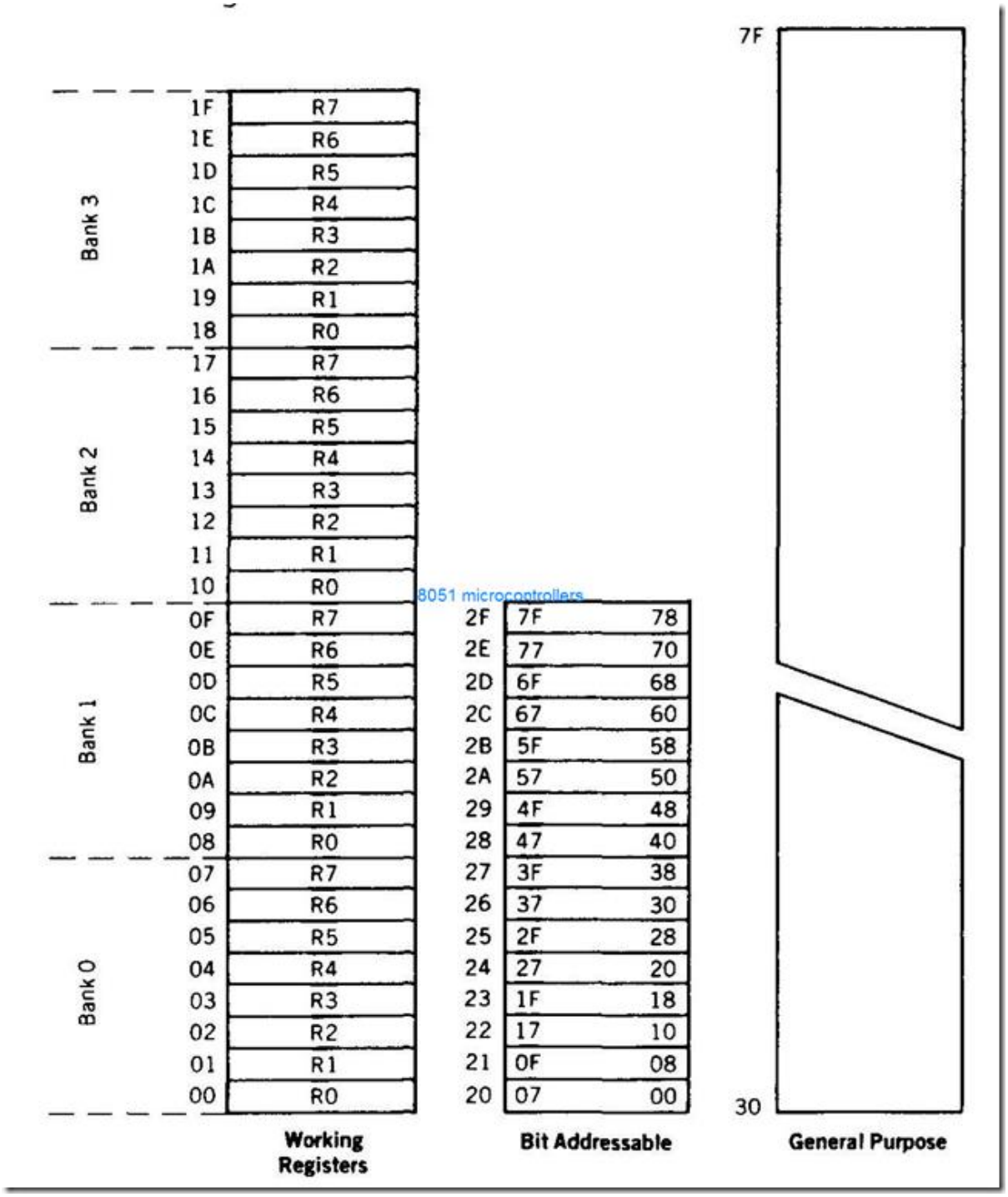
A functioning computer must have memory for program code bytes, commonly in ROM, and RAM memory for variable data that can be altered as the program runs. The 8051 has internal RAM and ROM memory for these functions. Additional memory can be added externally using suitable circuits.

Unlike microcontrollers with Von Neumann architectures, which can use a single memory address for either program code or data, but not for both, the 8051 has a Harvard architecture, which uses the same address, in different memories, for code and data. Internal circuitry accesses the correct memory based upon the nature of the operation in progress.

INTERNAL RAM

The 128-byte internal RAM, which is shown generally in Figure 1 and in detail in Figure 5, is organized into three distinct areas:

1. Thirty-two bytes from address 00h to 1 Fh that make up 32 working registers organized as four banks of eight registers each. The four register banks are numbered 0 to 3 and are made up of eight registers named R0 to R 7. Each register can be addressed by name (when its bank is selected) or by its RAM address. Thus R0 of bank 3 is R0 (if bank 3 is currently selected) or address 18h (whether bank 3 is selected or not). Bits RS0 and RS 1 in the PSW determine which bank of registers is currently in use at any time when the program is running. Register banks not selected can be used as general-purpose RAM. Bank 0 is selected upon reset.



2. A bit-addressable area of 16 bytes occupies RAM byte addresses 20h to 2Fh, forming a total of 128 addressable bits. An addressable bit may be specified by its bit address of 00h to 7Fh, or 8 bits may form any byte address from 20h to 2Fh. Thus. for example, bit address 4Fh is also bit 7 of byte address 29h. Addressable bits are useful when the program need only remember a binary event (switch on, light off, etc.). Internal RAM is in short supply as it is, so why use a byte when a bit will do?

3. A general-purpose RAM area above the bit area, from 30h to 7Fh, addressable as bytes.

THE STACK AND THE STACK POINTER

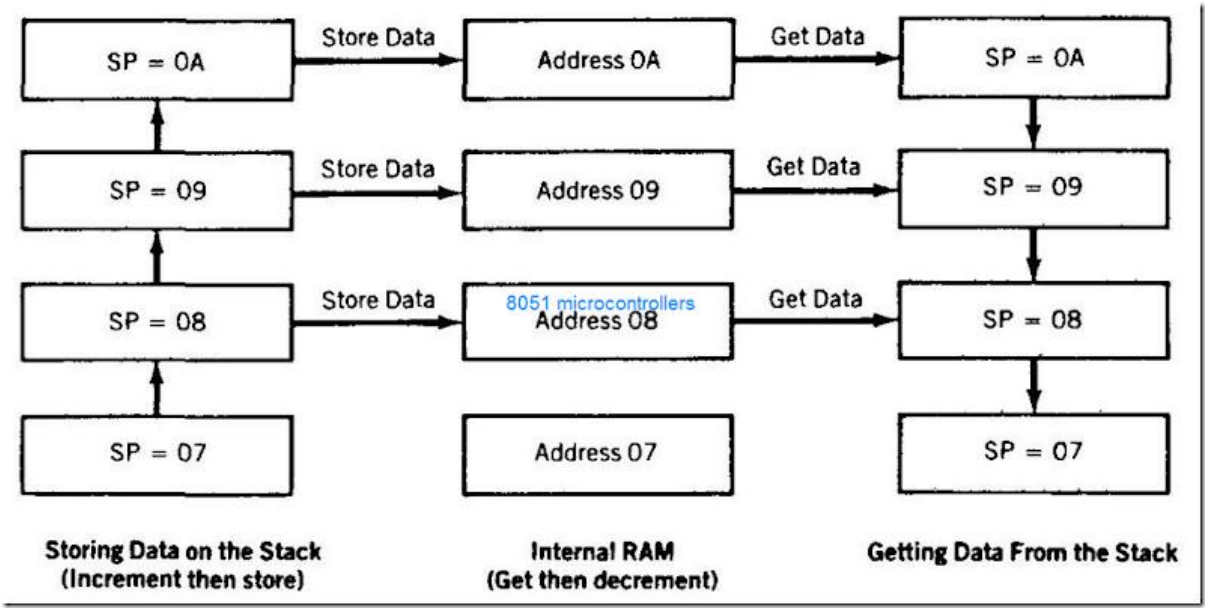
The stack refers to an area of internal RAM that is used in conjunction with certain opcodes to store and retrieve data quickly. The 8-bit stack pointer (SP) register is used by the 8051 to hold an internal RAM address that is called the "top of the stack." The address held in the SP register is the location in internal RAM where the last byte of data was stored by a stack operation.

When data is to be placed on the stack, the SP increments before storing data on the stack so that the stack grows up as data is stored. As data is retrieved from the stack, the byte is read from the stack, and then the SP decrements to point to the next available byte of stored data.

Operation of the stack and the SP is shown in Figure 6. The SP is set to 07h when the 8051 is reset and can be changed to any internal RAM address by the programmer.

The stack is limited in height to the size of the internal RAM. The stack has the potential (if the programmer is not careful to limit its growth) to overwrite valuable data in the register banks, bit-addressable RAM, and scratch-pad RAM areas. The programmer is responsible for making sure the stack does not grow beyond pre-defined bounds!

The stack is normally placed high in internal RAM, by an appropriate choice of the number placed in the SP register, to avoid conflict with the register, bit, and scratch-pad internal RAM areas.



SPECIAL FUNCTION REGISTERS

The 8051 operations that do not use the internal 128-byte RAM addresses from 00h to 7Fh are done by a group of specific internal registers. each called a special-function register (SFR). which may be addressed much like internal RAM. using addresses from 80h to FFh .

Some SFRs (marked with an asterisk * in Figure 1b) are also bit addressable. as is the case for the bit area of RAM. This feature allows the programmer to change only what needs to be altered. leaving the remaining bits in that SFR unchanged.

Not all of the addresses from 80h to FFh are used for SFRs. and attempting to use an address that is not defined. or "empty." results in unpredictable results. In Figure 2. I b, the SFR addresses are shown in the upper right corner of each block. The SFR names and equivalent internal RAM addresses are given in the following list :

NAME	FUNCTION	INTERNAL RAM ADDRESS (HEX)
A	Accumulator	OEO
B	Arithmetic	0F0
DPH	Addressing external memory	83
DPl	Addressing external memory	82
IE	Interrupt enable control	0A8
IP	Interrupt priority	0B8
PO	Input/output port latch	80
Pl	Input/output port latch	90
P2	Input/output port latch	A0
P3	Input/output port latch	0B0
PC ON	Power control	87
PSW	Program status word	0D0
SCON	Serial port control	98
SBUF	Serial port data buffer	99
SP	Stack pointer	81
TMOD	Timer / counter mode control	89
TCON	Timer / counter control	88
TLO	Timer 0 low byte	8A
THO	Timer 0 low byte	8C
TL1	Timer 0 low byte	8B
TH1	Timer 1 high byte	8D

Note that the PC is not part of the SFR and has no internal RAM address . see also Appendix F

SFRs are named in certain opcodes by their functional names, such as A or TH0, and are referenced by other opcodes by their addresses, such as 0E0h or 8Ch. Note that any address used in the program must start with a number; thus address E0h for the A SFR begins with 0. Failure to use this number convention will result in an assembler error when the program is assembled.

Internal ROM

The 8051 is organized so that data memory and program code memory can be in two entirely different physical memory entities. Each has the same address ranges.

The structure of the internal RAM has been discussed previously. A corresponding block of internal program code, contained in an internal ROM, occupies code address space 0000h to 0FFFh. The PC is ordinarily used to address program code bytes from addresses 0000h to FFFFh. Program addresses higher than 0FFFh, which exceed the internal ROM capacity, will cause the 8051 to automatically fetch code bytes from external program memory. Code bytes can also be fetched exclusively from an external memory, addresses 0000h to FFFFh, by connecting the external access pin (EA pin 31 on the DIP) to ground. The PC does not care where the code is; the circuit designer decides whether the code is found totally in internal ROM, totally in external ROM, or in a combination of internal and external ROM.

INPUT/OUTPUT PINS, PORT AND CIRCUITS

I/O ports of 8051:

There are four ports P0, P1, P2 and P3 each use 8 pins, making them 8-bit ports. All the ports upon RESET are configured as output, ready to be used as output ports. To use any of these ports as an input port, it must be programmed. Each port of 8051 has bidirectional capability. Port 0 is called 'true bidirectional port' as it floats (tristated) when configured as input. Port-1, 2, 3 are called 'quasi bidirectional port'.

Configuration :

• The below figures represent one bit (one line) of Port 0, 1,2 & 3.
• Every port line has a one-bit latch, in the form of a D Flip Flop
• It is used to hold the value on the port, when used as an output port.
• The latch will capture a "0" or a "1" from the internal bus, on getting "write to latch" signal.
• The port also has buffers that allow us to either read from the latch or read from the port line.

Input operation:

• When a "1" is written to the latch, the port becomes an input port, by turning the FETs off.

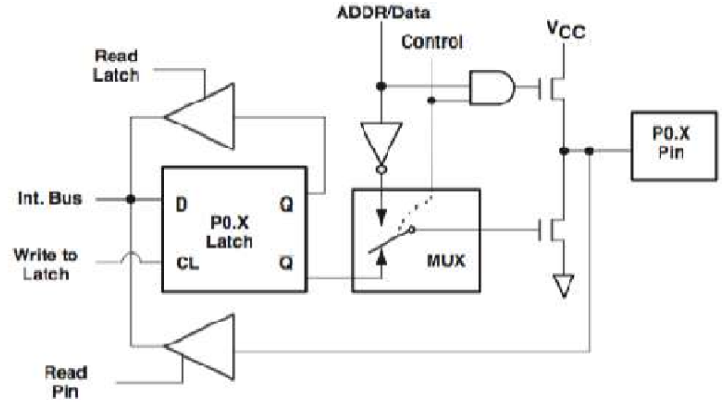
Output operation:

• To send a "0": Write a "0" on the latch. This turns ON the FET and the port pin gets grounded, so the port pin contains logic "0".
• To send a "1": Write a "1" on the latch. This turns OFF the FET and the port line "floats", containing neither logic 0 nor 1 (just like input).
• By connecting VCC using an external pull up resister, we can output a "1" on the port line.

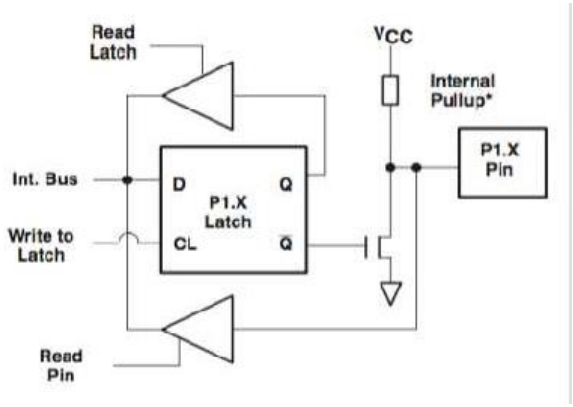
Alternate function:

• Port 0 can also be used as the multiplexed bus AD7 – AD0, and can carry address or data.
• Port 1 does not have alternate functions.
• Port 2 can be used as the higher order address bus (A15 – A8).
• The "control" signal shown in the Port 0, Port 2 diagrams directs the address line to the "gate" of the FET.
• Port 3 can be used as (P3.0-RXD, P3.1-TXD, P3.2-INT0, P3.3-INT1,P3.4- T0, P3.5 – T1, P3.6 – WR, P3.7 – RD). The "Alternate Output function" signal directs the alternate function to the "gate" of the FET.

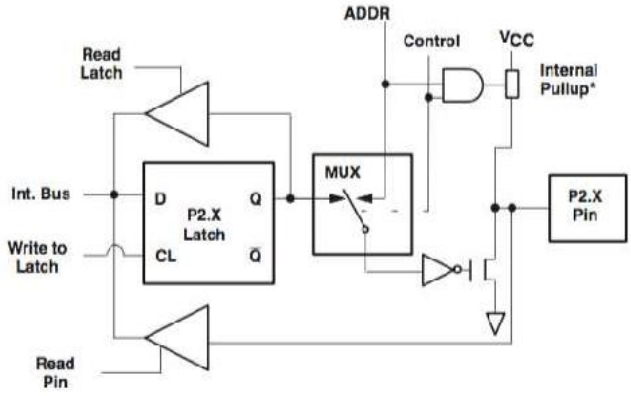
PORT 0 PIN CONGIGURATION



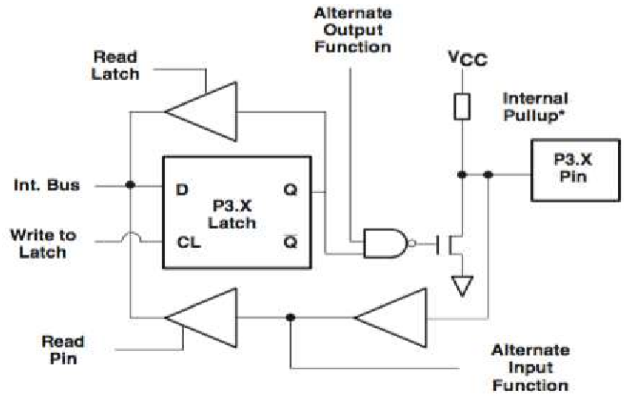
PORT 1 PIN CONGIGURATION



PORT 2 PIN CONGIGURATION



PORT 3 PIN CONGIGURATION



Port-0

Port 0 pins may serve as inputs, outputs, or, when used together, as a bidirectional Low order address and data bus for external memory. For example, when a pin is to be used as an input, a 1 must be written to the corresponding port 0 latch by the program, thus turning both of the output transistors off, which in turn causes the pin to "float" in a high impedance state, and the pin is essentially connected to the input buffer

When used as an output, the pin latches that are programmed to a 0 will turn on the lower FET, grounding the pin. All latches that are programmed to a 1 still float; thus, external pull up resistors will be needed to supply a logic high when using port 0 as an output.

When port 0 is used as an address bus to external memory, internal control Signal switch the address lines to the gates of the Field Effect Transistors (FETs). A logic 1 on an address bit will turn the upper

FET on and the lower FET off to provide a logic high at the pin. When the address bit is a zero, the lower FET is on and the upper FET off to provide a logic low at the pin. After the address has been formed and latched into external circuits by the Address Latch Enable (ALE) pulse, the bus is turned around to become a data bus. Port 0 now reads data from the external memory and must be configured as an input, so a logic 1 is automatically written by internal control logic to all port 0 latches

PORT-1

Port 1 pins have no dual functions. Therefore, the output latch is connected directly to the gate of the lower FET, which has an FET circuit labeled "Internal FET Pull up" as an active pull up load. Used as an input, a 1 is written to the latch, turning the lower FET off; the pin and the input to the pin buffer are pulled high by the FET load. An external circuit can overcome the high impedance pull up and drive the pin low to input a 0 or leave the input high for a 1. If used as an output, the latches containing a I can drive the input of an external circuit high through the pull up. If a 0 is written to the latch, the lower FET is on, the pull up is off, and the pin can drive the input of the external circuit low. To aid in speeding up switching times when the pin is used as an output, the internal FET pull up has another FET in parallel with it. The second FET is turned on for two oscillator time periods during a low-to-high transition on the pin, as shown in Figure 2.7. This arrangement provides a low impedance path to the positive voltage supply to help reduce rise times in charging any parasitic capacitances in the external circuitry

PORT-2

Port 2 may be used as an input/output port similar in operation to port 1. The alternate use of port 2 is to supply a high-order address byte in conjunction with the port 0 low-order byte to address external memory.

Port 2 pins are momentarily changed by the address control signals when supplying the high byte of a 16-bit address. Port 2 latches remain stable when external memory is addressed, as they do not have to be turned around (set to 1) for data input as is the case for port 0

PORT-3

Port 3 is an input/output port similar to port I. The input and output functions can be programmed under the control of the P3 latches or under the control of various

PIN	ALTERNATE USE	SFR
P3.0–RXD	Serial data input	SBUF
P3.1–TXD	Serial data output	SBUF
P3.2–$\overline{\text{INT0}}$	External interrupt 0	TCON.1
P3.3–$\overline{\text{INT1}}$	External interrupt 1	TCON.3
P3.4–T0	External timer 0 input	TMOD
P3.5–T1	External timer 1 input	TMOD
P3.6–$\overline{\text{WR}}$	External memory write pulse	—
P3.7–$\overline{\text{RD}}$	External memory read pulse	—

EXTERNAL MEMORY

The system designer is not limited by the amount of internal RAM and ROM available on chip. Two separate external memory spaces are made available by the 16-bit PC and DPTR and by different control pins for enabling external ROM and RAM chips. Internal control circuitry accesses the correct physical memory, depending upon the machine cycle state and the op code being executed.

There are several reasons for adding external memory, particularly program memory, when applying the 8051 in a system. When the project is in the prototype stage, the expense—in time and money—of having a masked internal ROM made for each program "try" is prohibitive.

To alleviate this problem, the manufacturers make available an EPROM version, the 8751, which has 4K of on-chip EPROM that may be programmed and erased as needed as the program is developed. The resulting circuit board layout will be identical to one that uses a factory-programmed 8051. The only drawbacks to the 8751 are the specialized EPROM programmers that must be used to program the non-standard 40-pin part, and the limit of "only" 4096 bytes of program code. The 8751 solution works well if the program will fit into 4K bytes.

Unfortunately, many times, particularly if the program is written in a high-level language, the program size exceeds 4K bytes, and an external program memory is needed. Again, the manufacturers provide a version for the job, the ROM less 8031. The EA pin is grounded when using the 8031, and all program code is contained in an external EPROM that may be as large as 64K bytes and that can be programmed using standard EPROM programmers.

External RAM, which is accessed by the DPTR, may also be needed when 128 bytes of internal data storage is not sufficient. External RAM, up to 64K bytes, may also be added to any chip in the 8051 family

CONNECTING EXTERNAL MEMORY

shows the connections between an 8031 and an external memory configuration consisting of I6K bytes of EPROM and 8K bytes of static RAM. The 8051 accesses external RAM whenever certain program instructions are executed. External ROM is accessed whenever the EA (external access) pin is connected to ground or when the PC contains an address higher than the last address in the internal 4K bytes ROM (OFFFh). 8051 designs can thus use internal and external ROM automatically; the 8031, having no internal ROM, must have EA grounded.

shows the timing associated with an external memory access cycle. During any memory access cycle, port 0 is time multiplexed. That is, it first provides the lower byte of the 16-bit memory address, then acts as a bidirectional data bus to write or read a byte of memory data. Port 2 provides the high byte of the memory address during the entire memory read/write cycle. The lower address byte from port 0 must be latched into an external register to Savethe byte. Address byte save is accomplished by the ALE clock pulse that provides the correct timing for the '373 type data latch. The port 0 pins then become free to serve as a data bus.

If the memory access is for a byte of program code in the ROM, the PSEN (program store enable) pin will go low to enable the ROM to place a byte of program code on the data bus. If the access is for a RAM byte, the WR (write) or RD (read) pins will go low, enabling data to flow between the RAM and the data bus.

The ROM may be expanded to 64K by using a 27512 type EPROM and connecting the remaining port 2 upper address lines AI4-A15 to the chip. At this time the largest static RAMs available are 32K in size; RAM can be expanded to 64K by using two 32K RAMs that are connected through address

A14 of port 2. The first 32K RAM (OOOOh-7FFFh) can then be enabled when AI5 of port 2 is low, and the second 32K RAM (SOOOh-FFFFh) when A15 is high, by using an inverter

Note that the WR and RD signals are alternate uses for port 3 pins 16 and 17. Also, port 0 is used for the lower address byte and data; port 2 is used for upper address bits. The use of external memory consumes many of the port pins, leaving only port 1 and parts of port 3 for general I/O.

FIGURE External Memory Connections

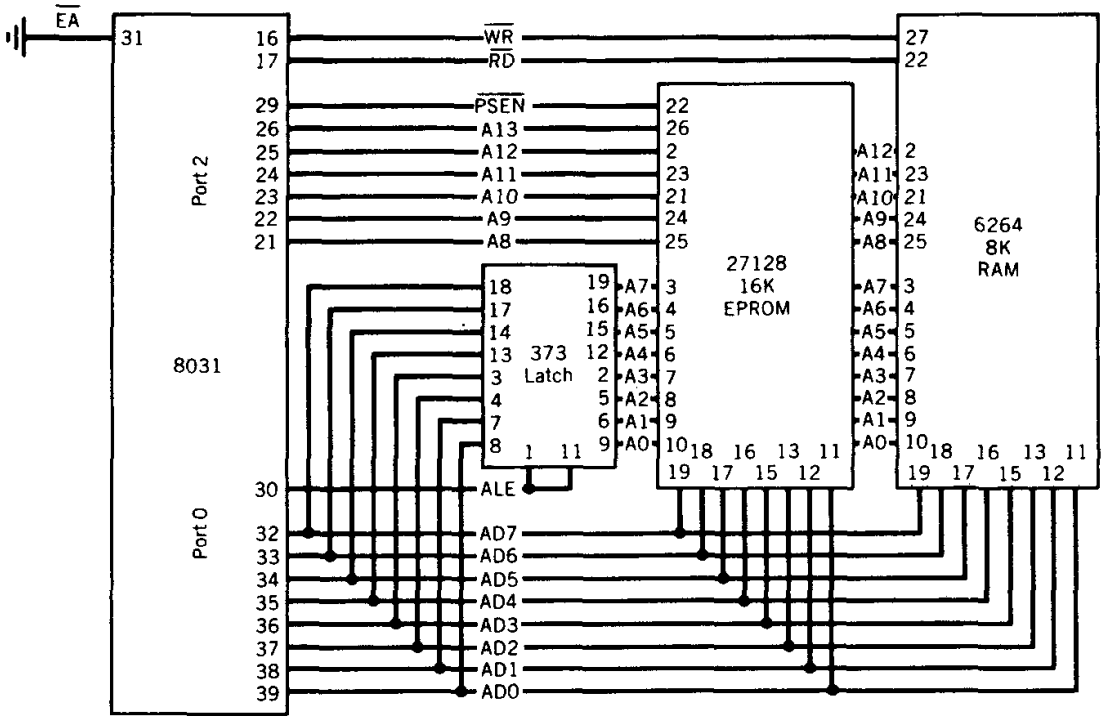
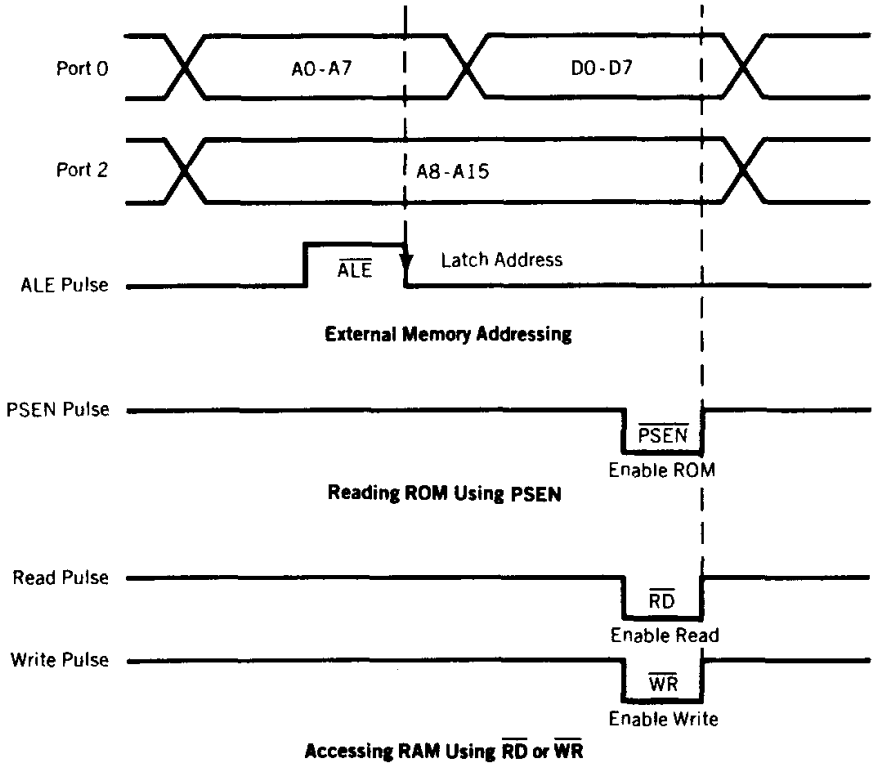
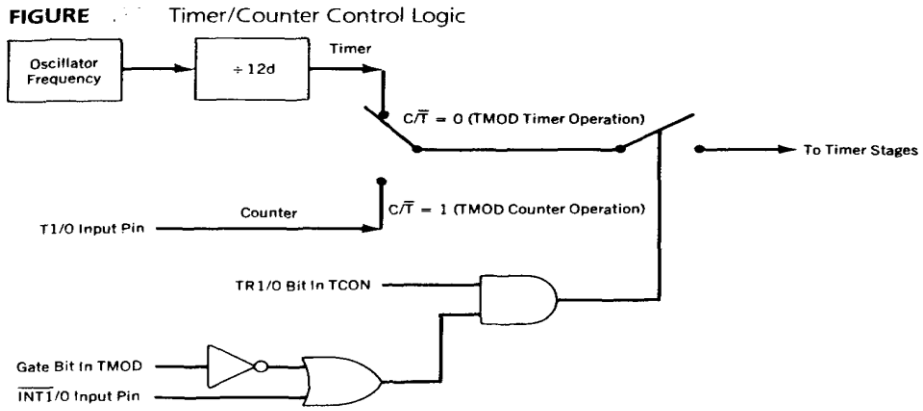


FIGURE External Memory Timing



COUNTER AND TIMERS

it is a device that counts down from a specified time interval and used to generate a time delay, for example, an hourglass is a timer. a counter is a device that stores (and sometimes displays) the number of times a particular event or process occurred, with respect to a clock signal.



TIMER/COUNTER LOGIC

TIMER COUNTER INTERRUPTS

It is a device that counts down from a specified time interval and used to generate a time delay, for example, an hourglass is a timer. A counter is a device that stores (and sometimes displays) the number of times a particular event or process occurred, with respect to a clock signal. For example, an interrupt occurs when a down counting timer reaches 0 and reloads the modulus in the main counter. The timer sends a hardware signal to an "interrupt controller" which suspends execution of the main program and makes the processor jump to a software function called an "interrupt service routine" or ISR

TIMER MODES OF OPERATION

Timers of 8051:

• 8051 has 2, 16-bit Up Counters T1 and T0.
• If the counter counts internal clock pulses it is known as timer.
• If it counts external clock pulses it is known as counter.
• Each counter is divided into 2, 8-bit registers TH1 - TL1 and TH0 - TL0.
• The timer action is controlled mainly by the TCON and the TMOD registers.

TCON - Timer Control (SFR) [Bit-Addressable As TCON.7 to TCON.0]

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1 and TF0: (Timer Overflow Flag)

• Set (1) when Timer 1 or Timer 0 overflows respectively i.e. its bits roll over from all 1's to all 0's.
• Cleared (0) when the processor executes ISR (address 001BH for Timer 1 and 000BH for Timer 0).

TR1 and TR0: (Timer Run Control Bit)

• Set (1) - Starts counting on Timer 1 or Timer 0 respectively.
• Cleared (0) - Halts Timer 1 or Timer 0 respectively.

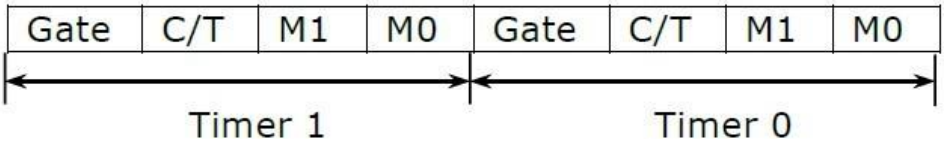
IE1 and IE0: (External Interrupt Edge Flag)

- Set (1) when external interrupt signal received at **INT1** or **INT0** respectively.
- Cleared (0) when ISR executed (address 0013H for Timer 1 and 0003H for Timer 0).

IT1 and IT0: (External Interrupt Type Control Bit)

- Set (1) - Interrupt at **INT1** or **INT0** must be -ve edge triggered.
- Cleared (0) - Interrupt at **INT1** or **INT0** must be low-level triggered.

TMOD - Timer Mode Control (SFR) [NOT Bit-Addressable]



C/T**: (Counter/Timer)**

- Set (1) - Acts as Counter (Counts external frequency on T1 and T0 pin inputs).
- Cleared (0) - Acts as Timer (Counts internal clock frequency, fosc/12).

Gate**: (Gate Enable Control bit)**

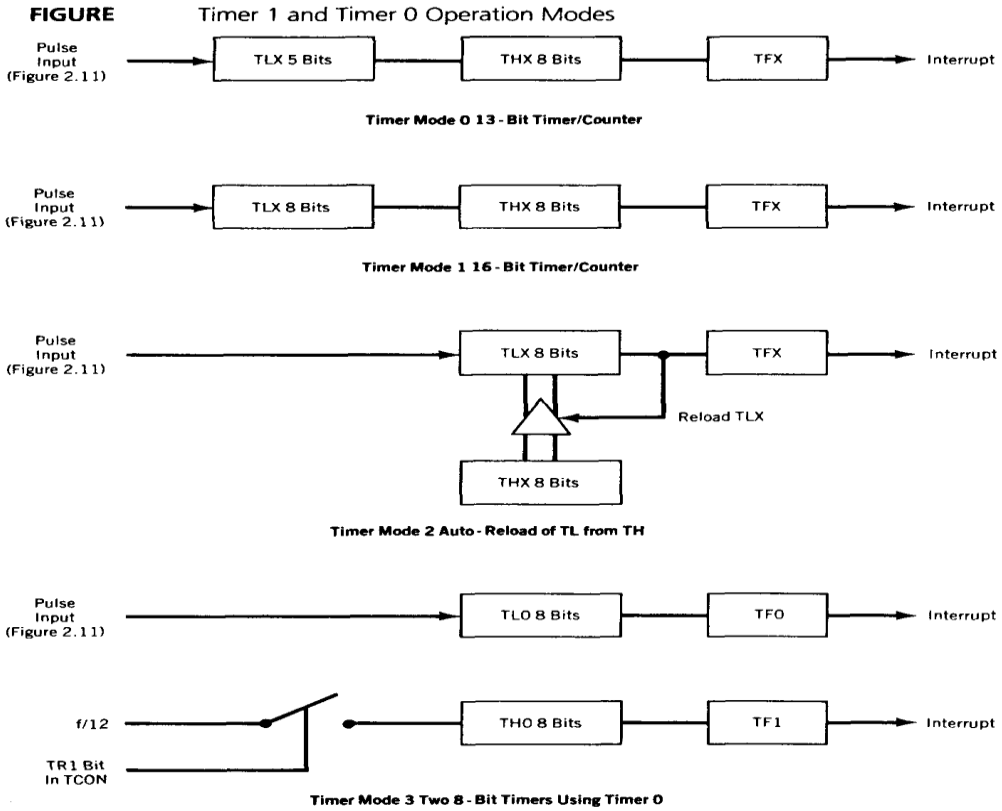
- Set (1) - Timer controlled by hardware i.e. INTX signal.
- Cleared (0) – Counting independent of INTX signal.

M1, M0: (Mode Selection bits)

Used to select the operational modes

M1	M0	Timer Mode
0	0	Mode 0
0	1	Mode 1
1	0	Mode 2
1	1	Mode 3

FIGURE Timer 1 and Timer 0 Operation Modes



TIMER 1 AND TIMER 0 OPERATION MODES

SERIAL DATA INPUT /OUTPUT

Computers must be able to communicate with other computers in modern multiprocessor distributed systems. One cost-effective way to communicate is to send and receive data bits serially. The 8051 has a serial data communication circuit that uses register SBUF to hold data. Register SCON controls data communication, register PCON controls data rates, and pins RXD (P3.0) and TXD (P3.1) connect to the serial data network.

SBUf is physically two registers. One is write only and is used to hold data to be transmitted out of the 8051 via TXD. The other is read only and holds received data from external sources via RXD. Both mutually exclusive registers use address 99h.

There are four programmable modes for serial data communication that are chosen by setting the SMX bits in SCON. Baud rates are determined by the mode chosen. Figure 13 shows the bit assignments for SCON and PCON.

Serial Data Interrupts

Serial data communication is a relatively slow process, occupying many milliseconds per data byte to accomplish. In order not to tie up valuable processor time, serial data flags are

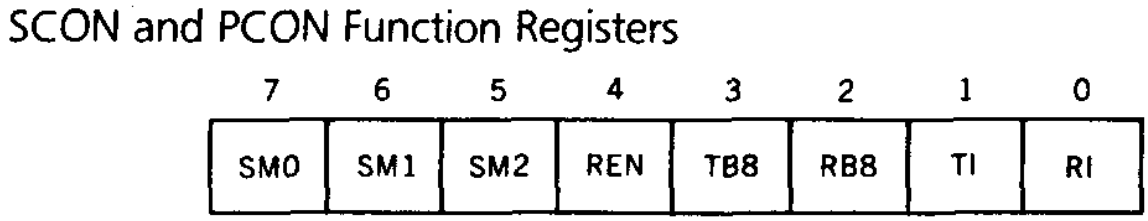


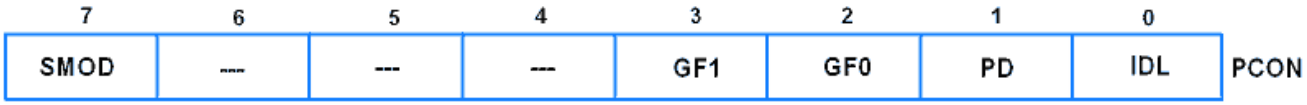
FIGURE SCON FUNCTION REGISTERS

THE SERIAL PORT CONTROL (SCON) SPECIAL FUNCTION REGISTER

<table>
<tr><th>Bit</th><th>symbol</th><th>Function</th></tr>
<tr><td>7</td><td>SM0</td><td>Serial port mode bit 0. Set/cleared by program to select mode.</td></tr>
<tr><td>6</td><td>SM1</td><td>Serial port mode bit 1. Set/cleared by program to select mode.
<table>
<tr><th>SM0</th><th>SM1</th><th>Mode</th><th>Description</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Shift register; baud = f/12</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>8-bit UART; baud = variable</td></tr>
<tr><td>1</td><td>0</td><td>2</td><td>9-bit UART; baud = 1/32 or f/64</td></tr>
<tr><td>1</td><td>1</td><td>3</td><td>9-bit UART; baud = variable</td></tr>
</table></td></tr>
<tr><td>5</td><td>SM2</td><td>Multiprocessor communications bit. Set/cleared by program to enable multiprocessor communications in modes 2 and 3. When set to 1 an interrupt is generated if bit 9 of the received data is a 1; no interrupt is generated if bit 9 is a 0. If set to 1 for mode 1, no interrupt will be generated unless a valid stop bit is received. Clear to 0 if mode 0 is in use.</td></tr>
<tr><td>4</td><td>REN</td><td>Receive enable bit. Set to 1 to enable reception; cleared to 0 to disable reception.</td></tr>
<tr><td>3</td><td>TB8</td><td>Transmitted bit B. Set/cleared by program in modes 2 and 3.</td></tr>
</table>

2	RB8	Received bit B. Bit B of received data in modes 2 and 3; stop bit in mode 1. Not used in mode 0.
1	T1	Transmit interrupt flag. Set to one at the end of bit 7 time in mode 0, and at the beginning of the stop bit for other modes. Must be cleared by the program.
0	R1	Receive interrupt flag. Set to one at the end of bit 7 time in mode 0, and halfway through the stop bit for other modes. Must be cleared by the program.

Bit addressable as SCON.O to SCON.7



THE POWER MODE CONTROL (PCON) SPECIAL FUNCTION REGISTER

Bit	symbol	Function
7	SMOD	Serial baud rate modify bit. Set to 1 by program to double baud rate using timer 1 for modes 1, 2, and 3. Cleared to 0 by program to use timer 1 baud rate.
6-4	-	Not implemented.
3	GF1	General purpose user flag bit 1. Set/cleared by program.
2	GF0	General purpose user flag bit 0 . Set/cleared by program.
1	PD	Power down bit. Set to 1 by program to enter power down configuration for CHMOS processors.
0	IDL	Idle mode bit. Set to 1 by program to enter idle mode configuration for CHMOS processors. PCON is not bit addressable.

included in SCON to aid in efficient data transmission and reception. Notice that data transmission is under the complete control of the program, but reception of data is unpredictable and at random times that are beyond the control of the program.

The serial data flags in SCON. T1 and R1, are set whenever a data byte is transmitted (T1) or received (R1). These flags are ORed together to produce an interrupt to the program. The program must read these flags to determine which caused the interrupt and then clear the flag. This is unlike the timer flags that are cleared automatically; it is the responsibility of the programmer to write routines that handle the serial data flags.

DATA TRANSMISSION

Transmission of serial data bits begins anytime data is written to SBUF. T1 is set to a 1 when the data has been transmitted and signifies that SBUF is empty (for transmission purposes) and that another data byte can be sent. If the program fails to wait for the T1 flag and overwrites SBUF while a previous data byte is in the process of being transmitted, the results will be unpredictable (a polite term for "garbage out").

DATA RECEPTION

Reception of serial data will begin if the receive enable bit (REN) in SCON is set to 1 for all modes. In addition, for mode 0 only, R1 must be cleared to 0 also. Receiver interrupt flag R1

is set after data has been received in all modes. Setting REN is the only direct program control that limits the reception of unexpected data; the requirement that R1 also be 0 for mode 0 prevents the reception of new data until the program has dealt with the old data and reset R1.

Reception can begin in modes 1, 2, and 3 if R1 is set when the serial stream of bits begins. R1 must have been reset by the program before the last bit is received or the incoming data will be lost. Incoming data is not transferred to SBUF until the last data bit has been received so that the previous transmission can be read from SBUF while new data is being received.

SERIAL DATA TRANSMISSION MODES

The 8051 designers have included four modes of serial data transmission that enable data communication to be done in a variety of ways and a multitude of baud rates. Modes are selected by the programmer by setting the mode bits SM0 and SM1 in SCON. Baud rates are fixed for mode 0 and variable, using timer 1 and the serial baud rate modify bit (SMOD) in PCON, for modes 1. 2. and 3.

SERIAL DATA MODE 0-SHIFT REGISTER MODE

Setting bits SM0 and SM 1 in SCON to 00b configures SBUF to receive or transmit eight data bits using pin RXD for both functions. Pin TXD is connected to the internal shift frequency pulse source to supply shift pulses to external circuits. The shift frequency, or baud rate, is fixed at 1/12 of the oscillator frequency, the same rate used by the timers when in the timer configuration. The TXD shift clock is a square wave that is low for machine cycle states S3-S4-S5 and high for S6-S1-S2. Figure 14 shows the timing for mode 0 shift register data transmission.

When transmitting, data is shifted out of RXD; the data changes on the falling edge of S6P2, or one clock pulse after the rising edge of the output TXD shift clock. The system designer must design the external circuitry that receives this transmitted data to receive the data reliably based on this timing.

FIGURE Shift Register Mode 0 Timing

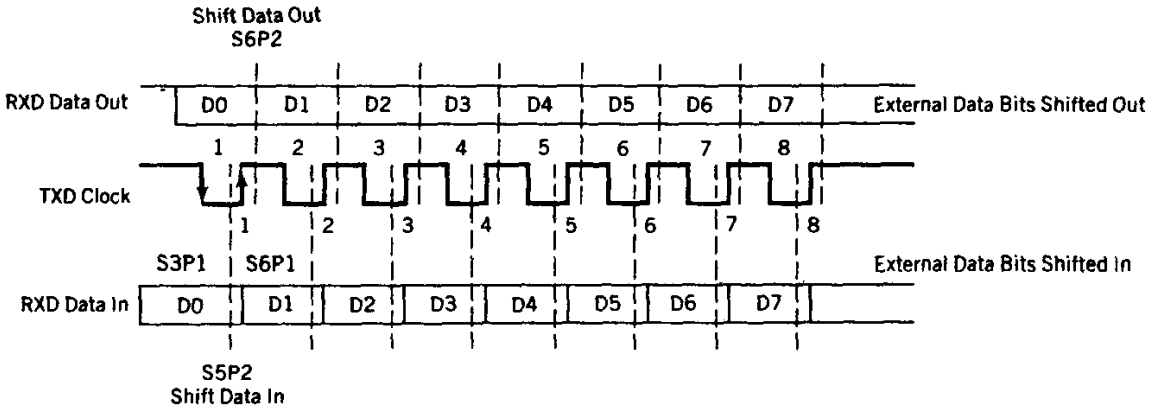


FIGURE Shift Register Mode 0 Timing

Received data comes in on pin RXD and should be synchronized with the shift clock produced at TXD. Data is sampled on the Jailing edge of S5P2 and shifted in to SBUF on the rising edge of the shift clock.

Mode 0 is intended not for data communication between computers, but as a high-speed serial data-collection method using discrete logic to achieve high data rates. The baud rate used in mode 0 will be much higher than standard for any reasonable oscillator frequency; for a 6 megahertz crystal, the shift rate will be 500 kilohertz.

Serial Data Mode 1-Standard UART

When SM0 and SM1 are set to 01b, SBUF becomes a 10-bit full-duplex receiver/ transmitter that may receive and transmit data at the same time. Pin RXD receives all data, and pin TXD transmits all data. Figure 15 shows the format of a data word.

Transmitted data is sent as a start bit, eight data bits (Least Significant Bit, LSB, first), and a stop bit. Interrupt flag T1 is set once all ten bits have been sent. Each bit interval is the inverse of the baud rate frequency, and each bit is maintained high or low over that interval.

Received data is obtained in the same order; reception is triggered by the falling edge of the start bit and continues if the stop bit is true (0 level) halfway through the start bit interval. This is an anti-noise measure; if the reception circuit is triggered by noise on the transmission line, the check for a low after half a bit interval should limit false data reception.

Data bits are shifted into the receiver at the programmed baud rate, and the data word will be loaded to SBUF if the following conditions are true: R1 must be 0, and mode bit SM2 is 0 or the stop bit is 1 (the normal state of stop bits). R1 set to 0 implies that the program has read the previous data byte and is ready to receive the next; a normal stop bit will then complete the transfer of data to SBUF regardless of the state of SM2. SM2 set to 0 enables the reception of a byte with any stop bit state, a condition which is of limited use in this mode, but very useful in modes 2 and 3. SM2 set to 1 forces reception of only "good" stop bits, an anti-noise safe guard.

Of the original ten bits, the start bit is discarded, the eight data bits go to SBUF. and the stop bit is saved in bit RB8 of SCON. R1 is set to 1, indicating a new data byte has been received.

If R1 is found to be set at the end of the reception, indicating that the previously received data byte has not been read by the program. or if the other conditions listed are not true, the new data will not be loaded and will be lost.

FIGURE Standard UART Data Word

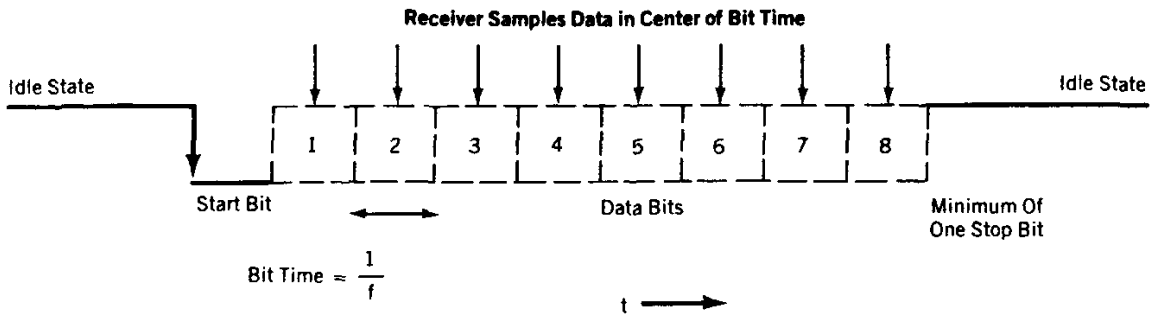


FIGURE Standard UART Data Word

MODE 1 BAUD RATES

Timer 1 is used to generate the baud rate for mode I by using the overflow flag of the timer to determine the baud frequency. Typically. timer I is used in timer mode 2 as an autoload R-hit timer that generates the baud frequency:

$$f_{baud} = \frac{2^{SMOD}}{32d} \times \frac{\text{oscillator frequency}}{12d \times [256d - (TH1)]}$$

SMOD is the control hit in PCON and can be 0 or 1, which raises the 2 in the equation to a value of 1 or 2.

If timer 1 is not run in timer mode 2, then the baud rate is

$$f_{baud} = \frac{2^{SMOD}}{32d} \times (\text{timer 1 overflow frequency})$$

and timer 1 can be run using the internal clock or as a counter that receives clock pulses from any external source via pin T1.

The oscillator frequency. is chosen to help generate both standard and nonstandard baud rates. If standard baud rates are desired, then an 11.0592 megahertz crystal could be selected. To get a standard rate of 9600 hertz then, the setting of TH 1 may be found as

follows:

$$TH1 = 256d - \left(\frac{2^0}{32d} \times \frac{11.0592 \times 10^6}{12 \times 9600d}\right) = 253.0000d = 0FDh$$

if SMOD is cleared to 0 .

Serial Data Mode 2-Multiprocessor Mode

Mode 2 is similar to mode 1 except 11 bits are transmitted: a start bit, nine data bits. and a stop bit, as shown in Figure 16. The ninth data bit is gotten from bit TB8 in SCON during transmit and stored in bit RB8 of SCON when data is received. Both the start and stop bits are discarded.

The baud rate is programmed as follows:

$$f_{baud2} = \frac{2^{SMOD}}{64d} \times \text{oscillator frequency}$$

FIGURE : Multiprocessor Data Word

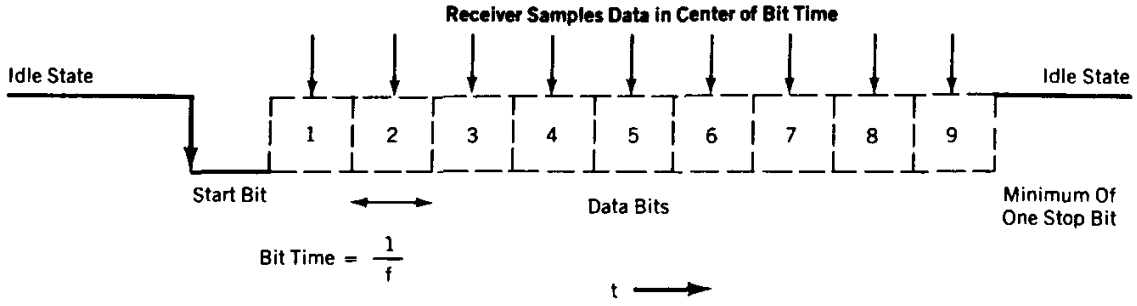


figure Multiprocessor Data Word

Here, as in the case for mode 0, the baud rate is much higher than standard communication rates. This high data rate is needed in many multi-processor applications. Data can be collected quickly from an extensive network of communicating microcontrollers if high baud rates are employed.

The conditions for setting RI for mode 2 are similar to mode 1: RI must be 0 before the last bit is received, and SM2 must be 0 or the ninth data bit must be a 1. Setting RI based upon the state of SM2 in the receiving 8051 and the state of bit 9 in the transmitted message makes multiprocessing possible by enabling some receivers to be interrupted by certain messages, while other receivers ignore those messages. Only those 8051's that have SM2 set to 0 will be interrupted by received data which has the ninth data bit set to 0; those with SM2 set to 1 will not be interrupted by messages with data bit 9 at 0. All receivers will be interrupted by data words that have the ninth data bit set to 1; the state of SM2 will not block reception of such messages.

This scheme allows the transmitting computer to "talk" to selected receiving computers without interrupting other receiving computers. Receiving computers can be commanded by the "talker" to "listen" or "deafen" by transmitting coded byte(s) with the ninth data bit set to 1 . The I in data bit 9 interrupts all receivers. instructing those that are programmed to respond to the coded byte(s) to program the state of SM2 in their respective SCON registers. Selected listeners then respond to the bit 9 set to 0 messages. while all other receivers ignore these messages. The talker can change the mix of listeners by transmitting bit 9 set to 1 messages that instruct new listeners to set SM2 to 0, while others are instructed to set SM2 to 1 .

Serial Data Mode 3

Mode 3 is identical to mode 2 except that the baud rate is determined exactly as in mode 1 . using Timer I to generate communication frequencies

INTERRUPTS

Interrupts are the events that temporarily suspend the main program, pass the control to other functions or sources and execute their task. It then passes the control to the main program where it had left off.
As code size increases and your application handles multiple modules, sequential coding would be too long and too complex. The interrupt mechanism helps to embed your software with hardware in a much simpler and efficient manner. In this topic, we will discuss the interrupts in 8051 using AT89S52 microcontroller.

When an interrupt is received, the controller stops after executing the current instruction. It transfers the content of the program counter into the stack.

TIMER FLAG INTERRUPTS

Each Timer is associated with a Timer interrupt. When a timer has finished counting, the Timer interrupt will notify the microcontroller by setting the required flag bit

SERIAL PORT INTERRUPT

This interrupt is used for serial communication. When enabled, it notifies the controller when a byte has been received or transmitted according to how the interrupt is configured.

EXTERNAL INTERRUPTS

8051 based AT89S52 microcontroller has two active-low external interrupts, INT0 and INT1.
See more information here: External Interrupt in 8051 microcontroller

IE register (Interrupt Enable Register)

Value after reset	0	X	0	0	0	0	0	0
IE	EA	-	ET2	ES	ET1	EX1	ET0	EX0
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

EA bit enables or disables all interrupt sources (globally):

- **0** – disables all interrupts (even enabled).
- **1** – enables specific interrupts.

EX0 bit enables or disables External interrupt 0:

- **0** – External interrupt 0 disabled.
- **1** – External interrupt 0 enabled.

ET0 bit enables or disables Timer T0 interrupt:

- **0** – Timer T0 interrupt disabled.
- **1** – Timer T0 interrupt enabled.

EX1 bit enables or disables External interrupt 1:

- **0** – External interrupt 1 disabled.

- **1** – External interrupt 1 enabled.

ET1 bit enables or disables Timer 1 overflow interrupt:

- **0** – Timer 1 overflow interrupt disabled.
- **1** – Timer 1 overflow interrupt enabled.

ES bit enables or disables serial port interrupt :

- **0** – serial port interrupt disabled.
- **1** – serial port interrupt enabled.

ET2 bit enables or disables Timer 2 overflow interrupt :

- **0** – Timer 2 overflow interrupt disabled.
- **1** – Timer 2 overflow interrupt enabled.

IP Register (Interrupt Priority Register)

Value After reset	0	0	0	0	0	0	0	0
IP	-	-	PT2	PS	PT1	PX1	PT0	PX0
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

- **PT2:** It defines the Timer 2 interrupt priority level (8052 only).
- **PS:** It defines the serial port interrupt priority level.
- **PT1:** It defines the Timer 1 interrupt priority level.
- **PX1:** It defines the external interrupt priority level.
- **PT0:** It defines the Timer 0 interrupt priority level.
- **PX0:** It defines the external interrupt 0 priority level.

Interrupt Subroutine (ISR)

Once all the configurations are done, the next step is to write the functions to execute when an interrupt occurs. And this is done by writing ISR functions. This function gets called automatically when an interrupt occurs.

While writing the ISR, The function definition must have the keyword ‘Interrupt’ followed by the interrupt number. Interrupt number is unique for each interrupt signal and a subroutine for a particular interrupt is identified by this number.

Interrupt number	Interrupt signal	Bit
0	External 0	EX0
1	Timer 0	IT0
2	External 1	EX1
3	Timer 1	IT1
4	Serial	ES
5	Timer 2	ET2

RESET

A reset can be considered to be the ultimate interrupts because the program may not block the action of the voltage on the RST pin, this type of interrupt is often called non maskable ,because no combination of bit in any register can stop or mask the reset action .unlike other interrupts the PC is not stored for later program resumption **.**

REGISTER	VALUE(HEX)
PC	0000
DPTR	0000
A	00
B	00
SP	07
PSW	00
P0—3	FF
IP	XXX00000b
IE	0XX00000b
TCON	00
TMOD	00
TH0	00
TL0	00
TH1	00
TL1	00
SCON	00
SBUF	XX
PCON	0XXXXXXXb

INTERRUPT CONTROL

The program must be able at critical times to inhibit the action of some or all of the interrupts so that crucial operational can be finished. the IE register hold s the programmable bit that can enable or disable all the interrupts as a group or if the group is enabled each individual interrupt source can be enabled or disabled

INTERRUPT ENABLE/DISABLE

Bits in the IE register are set to 1 if the corresponding interrupt source is to be enabled and set to 0 to disable the interrupt source. bit EA is a master or global bit that can enable or disable all of the interrupts

INTERRUPT PRIORITY

Register IP bits determine if any interrupt is to have a hight or low priority, bits set to 1 give the accompanying interrupt a high priority a 0 assigns a low priority. 0 assigns low priority.

IE0

TF0

1E1

TF1

SERIAL=RI OR TI

INTERRUPT DESTINATIONS

each interrupt source causes the program to do a hardware call to one of the dedicated addresses in program memory it is the responsibility of the program to place a routine at the address that will service the interrupt.

INTERRUPT	ADDRESS(HEX)
IE0	0003
TF0	000B
IE1	0013
TF1	001B
SERIAL	0023

UNIT -IV

PROGRAMMING THE 8051

ADDRESSING MODES -133
MOVING DATA 139-144
LOGICAL OPERATION 151-159
ARITHMETIC OPERATION 169-182
JUMP AND CALL INSTRUCTIONS 189- 202

ADDRESSING MODES:

An 8085 microprocessor uses five addressing modes: Immediate addressing mode, Register addressing mode, Register indirect addressing mode, Direct addressing mode, and Implicit addressing mode.

Immediate addressing mode

In this addressing mode, the data on which the operation is to be performed is mentioned in the instruction. Instead of specifying an address, the instruction specifies an operand along with the operation that is to be performed. The instruction is 2 bytes when there is 8-bit data and 4 bytes when there is 16-bit data. Examples of instructions that use immediate addressing mode are:

- **MVI B 05 –** This instruction transfers the data, that is 05, to register B.
- **JMP address –** This instruction moves the execution of the program to the specified address.
- **LXI H 3000 –** This instruction transfers the data, that is, 3000, to the H-L pair register.

Register addressing mode

In this addressing mode, the instruction mentions a register which stores some data. The operation specified by the instruction is performed in the register specified by it. The instruction can also specify two registers. The size of the instructions in register addressing mode is 1 byte. The instruction's opcode includes both the register and the operation to be performed. Examples of instructions that use register addressing mode are:

- **ADD B –** This instruction adds the data within register B with the data in the accumulator and stores the result in the accumulator.
- **INR B –** This instruction increases the data stored in register B by 1.
- **MOVE B, D –** This instruction moves the content of register D to register B.
- **PCHL –** This instruction transfers the contents stored in the HL pair to the program counter.

Direct addressing mode

In this addressing mode, the instruction specifies some address which stores the data. This memory location is mentioned in the instruction as an operand. The size of an instruction in the direct addressing mode is 3 bytes. However, input/output instructions in direct addressing mode are 2 bytes. Examples of instructions that use direct addressing mode are:

- **IN 35 –** This instruction reads the data from a port. The address of the port is 35.
- **LHLD address –** This instruction loads the data stored in the memory location specified by 'address' into the HL pair.
- **LDA 2050 –** This instruction stores the contents of the memory location 2050H into the accumulator.

- **STA 2050 –** This instruction stores the contents of the accumulator in the memory location 2050H.

Register Indirect addressing mode

In this addressing mode, the data on which the operation is to be performed is stored in some memory location. This memory location is specified in a register pair. The instruction then specifies this register pair. Examples of instructions that use register indirect addressing mode are:

- **LDAX B –** This instruction loads the contents stored in the memory location specified by the register pair BC into the accumulator.
- **LXIH 9500 –** This instruction stores the address '9500' into the HL pair.
- **MOV A, M –** This instruction moves the contents stored in the address specified by the HL pair into the accumulator.

Implied/Implicit addressing mode

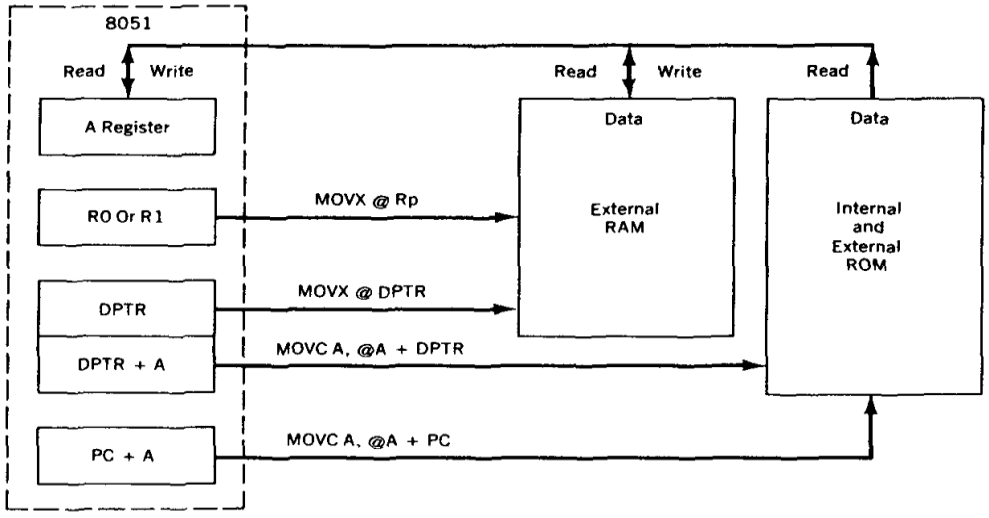
In this addressing mode, the operand is described implicitly in the definition of the instruction. The operand is specified with the opcode of the instruction. The size of an implied addressing mode instruction is 1 byte. The instructions generally operate on data stored in the accumulator. Examples of instructions that use implied addressing mode are as follows:

- **RRC –** This instruction rotates the contents of the accumulator to the right by one bit.
- **RLC –** This instruction rotates the contents of the accumulator to the left by one bit.
- **CMA –** This instruction complements the content stored in the accumulator. The result is then stored in the accumulator itself.

EXTERNAL DATA MOVES

It is possible to expand RAM and Rom memory space by adding external memory chips to the 8051 microcontroller the external memory can be as large as 64k for each of the RAM &ROM memory areas. opcodes that access this external memory always use indirect addressing to specify the external memory

FIGURE External Addressing using MOVX and MOVC



External addressing using mov x and mov c

Code memory read-only data moves

Data moves between RAM locations and 8051 registers are made by using MOV and MOVX opcodes. The data is usually of a temporary or "scratch pad" nature and disappears when the system is powered down.

There are times when access to a preprogrammed mass of data is needed, such as when using tables of predefined bytes. This data must be permanent to be of repeated use and is stored in the program ROM using assembler directives that store programmed data anywhere in ROM that the programmer wishes.

Access to this data is made possible by using indirect addressing and the A register in conjunction with either the PC or the DPTR, as shown in Figure 3.2. In both cases, the number in register A is added to the pointing register to form the address in ROM where the desired data is to be found. The data is then fetched from the ROM address so formed and placed in the A register. The original data in A is lost, and the addressed data takes its place.

As shown in the following table, the letter C is added to the MOV mnemonic to highlight the use of the opcodes for moving data from the source address in the Code ROM to the A register in the 8051:

Mnemonic	Operation
MOVC A,@A+DPTR	Copy the code byte, found at the ROM address formed by adding A and the DPTR, to A
MOVC A,@A+PC	Copy the code byte, found at the ROM address formed by adding A and the PC, to A

Note that the DPTR and the PC are not changed; the A register contains the ROM byte found at the address formed.

The following table shows examples of code ROM moves using register and indirect addressing modes:

Mnemonic	Operation
MOV DPTR,#1234h	Copy the immediate number 1234h to the DPTR
MOV A,#56h	Copy the immediate number 56h to A
MOVC A,@A+DPTR	Copy the contents of address 128Ah to A
MOVC A,@A+PC	Copies the contents of address 4059h to A if the PC contained 4000h and A contained 58h when the opcode is executed.

PUSH AND POP OPCODE

The PUSH and POP opcodes specify the direct address of the data. The data moves between an area of internal RAM, known as the stack, and the specified direct address. The stack pointer special-function register (SP) contains the address in RAM where data from the source address will be PUSHED, or where data to be POPED to the destination address is found. The SP register actually is used in the indirect addressing mode but is not named in the mnemonic. It is implied that the SP holds the indirect address whenever PUSHING or POPING. Figure 3.3 shows the operation of the stack pointer as data is PUSHED or POPED to the stack area in internal RAM.

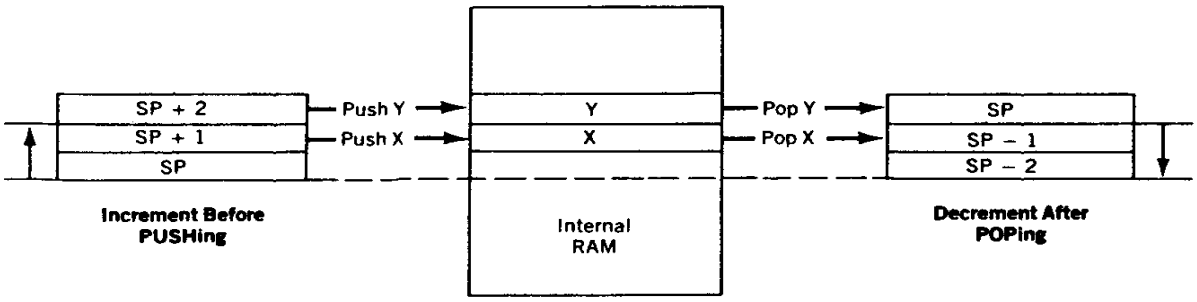
A PUSH opcode copies data from the source address to the stack. SP is incremented by one before the data is copied to the internal RAM location contained in SP so that the data is stored from low addresses to high addresses in the internal RAM. The stack grows up in memory as it is PUSHED. Excessive PUSHING can make the stack exceed 7Fb (the top of internal RAM). after which point data is lost.

A POP opcode copies data from the stack to the destination address. SP is decremented by one after data is copied from the stack RAM address to the direct destination to ensure that data placed on the stack is retrieved in the same order as it was stored.

The PUSH and POP opcodes behave as explained in the following table:

Mnemonic	Operation
PUSH add	Increment SP; copy the data in add to the internal RAM address contained in SP
POP add	Copy the data from the internal RAM address contained in SP to add; decrement the sp

FIGURE PUSH and POP the Stack



The SP register is set to 07h when the 8051 is reset, which is the same direct address in internal RAM as register R7 in bank 0. The first PUSH opcode would write data to R0 of bank 1. The SP should be initialized by the programmer to point to an internal RAM address above the highest address likely to be used by the program.

The following table shows examples of PUSH and POP opcodes:

Mnemonic	Operation
MOV 81h,#30h	Copy the immediate data 30h to the SP
MOV R0, #0ACh	Copy the immediate data ACh to R0
PUSH 00h	SP = 31h; address 31h contains the number ACh
PUSH 00h	SP = 32h; address 32h contains the number ACh
POP 0lh	SP = 31 h; register R 1 now contains the number Ach
POP 80h	SP = 30h; port 0 latch now contains the number ACh

MOV, PUSH, and POP opcodes all involve copying the data found in the source address to the destination address; the original data in the source is not changed. Exchange instructions actually move data in two directions: from source to destination and from destination to source. All addressing modes except immediate may be used in the XCH (exchange) opcodes:

Mnemonic	Operation
XCH A,Rr	Exchange data bytes between register Rr and A
XCH A,add	Exchange data bytes between add and A
XCH A,@Rp	Exchange data bytes between A and address in Rp
XCHD A,@Rp	Exchange lower nibble between A and address in Rp

Exchanges between A and any port location copy the data on the port pins to A, while the data in A is copied to the port latch. Register A is used for so many instructions that the XCH opcode provides a very convenient way to "save" the contents of A without the necessity of using a PUSH opcode and then a POP opcode.

The following table shows examples of data moves using exchange opcodes:

Mnemonic	Operation
XCH A,R7	Exchange bytes between register A and register R7
XCH A,0F0h	Exchange bytes between register A and register B
XCH A,@R1	Exchange bytes between register A and address in R 1
XCHD A,@R1	Exchange lower nibble in A and the address in R 1

LOGICAL OPERATION

The two data levels, byte or bit, at which the Boolean instructions operate are shown in the following table:

BOOLEAN OPERATOR	8051 MNEMONIC
AND	ANL (AND logical)
OR	ORL (OR logical)
XOR	XRL (exclusive OR logical)
NOT	CPL (complement)

There are also rotate opcodes that operate only on a byte, or a byte and the carry flag, to permit limited 8- and 9-bit shift-register operations. The following table shows the rotate opcodes:

Mnemonic	Operation
RL	Rotate a byte to the left; the Most Significant Bit (MSB) becomes the Least Significant Bit (LSB)
RLC	Rotate a byte and the carry bit left; the carry becomes the LSB, the MSB becomes the carry
RR	Rotate a byte to the right; the LSB becomes the MSB
RRC	Rotate a byte and the carry to the right; the LSB becomes the carry, and the carry the MSB
SWAP	Exchange the low and high nibbles in a byte

Byte-Level Logical Operations

The byte-level logical operations use all four addressing modes for the source of a data byte. The A register or a direct address in internal RAM is the destination of the logical operation result.

Keep in mind that all such operations are done using each individual bit of the destination and source bytes. These operations, called byte-level Boolean operations because tlfe entire byte is affected, are listed in the following table:

Mnemonic	Operation
ANL A,#n	AND each bit of A with the same bit of immediate number n; put the results in A
ANL. A.add	AND each bit of A with the same bit of the direct RAM address; put the results in A
ANLA,Rr	AND each bit of A with the same bit of register Rr; put the results in A AND each bit of A with the same bit of the contents of the RAM address contained in Rp; put the results in A
ANL A,@Rp	AND each bit of A with the direct RAM address; put the results in the direct RAM address
ANL add.A	AND each bit of the RAM address with the same bit in the number n; put the result in the RAM address
ANL add,#n	OR each bit of A with the same bit of n; put the results in A
ORL A,#n	OR each bit of A with the same bit of the direct RAM address; put the results in A
ORL A,add	OR each bit of A with the same bit of register Rr; put the results in A OR each bit of A with the same bit of the contents of the RAM address contained in Rp; put the results in A
ORL A,Rr	OR each bit of A with the direct RAM address; put the results in the direct RAM address
ORL A,@Rp	OR each bit of the RAM address with the same bit in the number n; put the result in the RAM address

ORL add.A	XOR each bit of A with the same bit of n; put the results in A
ORL add,#n	XOR each bit of A with the same bit of the direct RAM address; put the results in A
XRL A,#n	XOR each bit of A with the same bit of register Rr; put the results in A XOR each bit of A with the same bit of the contents of the RAM address contained in Rp; put the results in A
XRL A,add	XOR each bit of A with the direct RAM address; put the results in the direct RAM address
XRL A,Rr	Operation
XRL A,@Rp	AND each bit of A with the same bit of immediate number n; put the results in A
XRL add.A	AND each bit of A with the same bit of the direct RAM address; put the results in A
XRL add,#n	XOR each bit of the RAM address with the same bit in the number n; put the result in the RAM address
CLRA	Clear each bit of the A register to zero
CPL A	Complement each bit of A; every I becomes a 0, and each 0 becomes a 1

Note that no flags are affected unless the direct RAM address is the PSW.

Many of these byte-level operations use a direct address, which can include the port SFR addresses, as a destination. The normal source of data from a port is the port pins; the normal destination for port data is the port latch. When the destination of a logical operation is the direct address of a port, the latch register, not the pins, is used both as the source for the original data and then the destination for the altered byte of data. Any port operation that must first read the source data, logically operate on it, and then write it back to the source (now the destination) must use the latch. Logical operations that use the port as a source, but not as a destination, use the pins of the port as the source of the data.

For example, the port 0 latch contains FFh, but the pins are all driving transistor bases and are close to ground level. The logical operation

ANL PO,#0 Fh

which.is designed to turn the upper nibble transistors off, reads FFh from the latch, ANDs it with 0Fh to produce 0Fh as a result, and then writes it back to the latch to turn these transistors off. Reading the pins produces the result 00h, turning all transistors off, in error. But, the operation

ANL A ,P0

produces A = 00h by using the port 0 pin data, which is 00h.

The following table shows byte-level logical operation examples:

Mnemonic	**Operation**
MOV A,#0FFh	A= FF h
MOV R0,#77h	R0= 77h
ANL A,R0	A= 77h
MOV 15h,A	15h = 77h
CPL A	A= 88h
ORL 15h,#88h	15h = FFh
XRL A,15h	A= 77h
XRL A,R0	A= 00h
ANL A,15h	A= 00h
ORL A,R0	A= 77h
CLR A	A= 00h

XRL 15h,A	15h = FFh
XRL A.R0	A= 77h

Note that instructions that can use the SFR port latches as destinations are ANL, ORL, and XRL.

Bit-Level Logical Operations

Certain internal RAM and SFRs can be addressed by their byte addresses or by the address of each bit within a byte. Bit addressing is very convenient when you wish to alter a single bit of a byte, in a control register for instance, without having to wonder what you need to do to avoid altering some other crucial bit of the same byte. The assembler can also equate bit addresses to labels that make the program more readable. For example, bit 4 of TCON can become TR0, a label for the timer 0 run bit.

The ability to operate on individual bits creates the need for an area of RAM that contains data addresses that hold a single bit. Internal RAM byte addresses 20h to 2Fh serve this need and are both byte and bit addressable. The bit addresses are numbered from 00h to 7Fh to represent the 12Sd bit addresses (16d bytes x S bits) that exist from byte addresses 20h to 2Fh. Bit 0 of byte address 20h is bit address 00h, and bit 7 of byte address 2Fh is bit address 7Fh. You must know your bits from your bytes to take advantage of this RAM area.

Internal RAM Bit Addresses

The availability of individual bit addresses in internal RAM makes the use of the RAM very efficient when storing bit information. Whole bytes do not have to be used up to store one or two bits of data.

The correspondence between byte and bit addresses are shown in the following table:

BYTE ADDRESS (HEX)	BIT ADDRESSES (HEX)
20	00-07
21	08-0F
22	10-17
23	18-lF
24	20-27
25	28-2F
26	30-37
27	38-3F
28	40-47
29	48-4F
2A	50-57
28	58-5F
2C	60-67
20	68-6F
2E	70-77
2F	78-7F

Interpolation of this table shows, for example, the address of bit 3 of internal RAM byte address 2Ch is 63h, the bit address of bit 5 of RAM address 21 h is 00h, and bit address 47h is bit 7 of RAM byte address 28h.

SFR Bit Addresses

All SFRs may be addressed at the byte level by using the direct address assigned to it, but not all of the SFRs are addressable at the bit level. The SFRs that are also bit addressable form the

bit address by using the five most significant bits of the direct address for that SFR, together with the three least significant bits that identify the bit position from position 0 (LSB) to 7 (MSB).

The bit-addressable SFR and the corresponding bit addresses are as follows:

SFR	DIRECT ADDRESS (HEX)	BIT ADDRESSES (HEX)
A	OED	OEO-OE7
6	OFO	OFO-OF7
IE	OAS	OAS-OAF
IP	06S	06S-06F
PO	so	SO-S7
Pl	90	90-97
P2	OAO	OAO-OA7
P3	060	060-067
PSW	ODO	ODO-OD7
TCON	SS	SS-SF
SCON	9S	9S-9F

The patterns in this table show the direct addresses assigned to the SFR bytes all have bits 0-3 equal to zero so that the address of the byte is also the address of the LSB. For example, bit 0E3h is bit 3 of the A register. The carry flag, which is bit 7 of the PSW, is bit addressable as 0D7h. The assembler can also "understand" more descriptive mnemonics, such as P0.5 for bit 5 of port 0, which is more formally addressed as 85h.

Figure 4. I shows all the bit-addressable SFRs and the function of each addressable bit. (Refer to Chapter 2 for more detailed descriptions of the SFR bit functions.)

Bit-Level Boolean Operations

The bit-level Boolean logical opcodes operate on any addressable RAM or SFR bit. The carry flag (C) in the PSW special-function register is the destination for most of the opcodes because the flag can be tested and the program flow changed using instructions covered

The following table lists the Boolean bit-level operations.

Mnemonic	Operation
ANL C,b	AND C and the addressed bit; put the result in C
ANL C,/b	AND C and the complement of the addressed bit; put the result in C; the addressed bit is not altered
ORL C,b	OR C and the addressed bit; put the result in C
ORLC,/b	OR C and the complement of the addressed bit; put the result in C; the addressed bit is not altered
CPLC	Complement the C flag
CPL b	Complement the addressed bit
CLRC	Clear the C flag to zero
CLR b	Clear the addressed bit to zero
MOV C,b	Copy the addressed bit to the C flag
MOV b,C	Copy the C flag to the addressed bit
SETB C	Set the flag to one
SETB b	Set the addressed bit to one

Note that no flags, other than the C flag, are affected, unless the flag is an addressed bit.

As is the case for byte-logical operations when addressing ports as destinations, a port bit used as a destination for a logical operation is part of the SFR latch, not the pin. A port bit used as a source only is a pin, not the latch. The bit instructions that can use a SFR latch bit are: CLR, CPL, MOY, and SETB.

Bit-Addressable Control Registers

PROGRAM STATUS WORD (PSW) SPECIAL FUNCTION REGISTER. BIT ADDRESSES D0h to D7h.

7	6	5	4	3	2	1	0
CY	AC	F0	RSI	RS0	0V	Resarved	P

Bit	Function
7	Carry flag
6	Auxiliary carry flag
5	User flag 0
4	Register bank select bit 1
3	Register bank select bit 0
2	Overflow flag
1	Not used (reserved for future)
0	Parity flag

INTERRUPT ENABLE (IE) SPECIAL FUNCTION REGISTER. BIT ADDRESSES A8h TO AFh.

7	6	5	4	3	2	1	0
EA	Reserved	Reserved	ES	ET1	EX 1	ET0	EX0

Bit	Function
7	Disables all interrupts
6	Not used (reserved for future)
5	Not used (reserved for future)
4	Serial port interrupt enable
3	Timer 1 overflow interrupt enable
2	External interrupt 1 enable
1	Timer 0 interrupt enable
0	External interrupt 0 enable

EA disables all interrupts when cleared to 0; if EA = 1 then each individual interrupt will be enabled if 1, and disabled if 0.

INTERRUPT PRIORITY (IP) SPECIAL FUNCTION REGISTER. BIT ADDRESSES B8h to BFh.

7	6	5	4	3	2	1	0
*	*	Reserved	PS	PT1	PX1	PT0	PX0

Bit	Function
7	Not implemented

6	Not implemented
5	Not used (reserved for future)
4	Serial port interrupt priority
3	Timer 1 interrupt priority
2	External interrupt 1 priority
1	Timer 0 interrupt priority
0	External interrupt 0 priority

The priority bit may be set to 1 (highest) or 0 (lowest).

TIMER/COUNTER CONTROL (TCON) SPECIAL FUNCTION REGISTER. BIT ADDRESSES 88h to 8Fh.

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Bit	Function
7	Timer 1 overflow flag
6	Timer run control
5	Timer 0 overflow flag
4	Timer 0 run control
3	External interrupt 1 edge flag
2	External interrupt 1 mode control
1	External interrupt 0 edge flag
0	External interrupt 0 mode control

All flags can be set by the indicated hardware action; the flags are cleared when interrupt is serviced by the processor.

SERIAL PORT CONTROL (SCON) SPECIAL FUNCTION REGISTER. BIT ADDRESSES 98h to 9Fh

7	6	5	4	3	2	1	0
SM0	SM1	SM2	REN	TBB	RBB	TI	RI

.

Bit	Function
7	Serial port mode bit 0
6	Serial port mode bit 1
5	Multiprocessor communications enable
4	Receive enable
3	Transmitted bit in modes 2 and 3
2	Received bit in modes 2 and 3
1	Transmit interrupt flag
0	Receive interrupt flag

Bit-level logical operation examples are shown in the following table:

Mnemonic	Operation
SETB 00h	Bit 0 of RAM byte 20h = 1
MOV C,00h	C=1
MOV 7Fh,C	Bit 7 of RAM byte 2Fh = 1
ANL C,/00h	C = 0; bit 0 of RAM byte 20h =1

ORL C,00h	C=1
CPL 7Fh	Bit 7 of RAM byte 2Fh = 0
CLR C	C=0
ORL C,/7Fh	C=1 ;

Rotate and Swap Operations

The ability to rotate data is useful for inspecting bits of a byte without using individual bit opcodes. The A register can be rotated one bit position to the left or right with or without including the C flag in the rotation. If the C flag is not included, then the rotation involves the eight bits of the A register. If the C flag is included, then nine bits are involved in the rotation. Including the C flag enables the programmer to construct rotate operations involving any number of bytes.

The SWAP instruction can be thought of as a rotation of nibbles in the A register.
Figure diagrams the rotate and swap operations. which are given in the following table:

Mnemonic	Operation
RL A	Rotate the A register one bit position to the left; bit A0 to bit A I, A I to A2. A2 to A3. A3 to A4, A4 to A5, A5 to A6. A6 to A 7, and A 7 to A0
RLC A	Rotate the A register and the carry flag. as a ninth bit, one bit position to the left; bit A0 to bit Al, Al to A2, A2 to A3, A3 to A4, A4 to A5. A5 to A6, A6 to A 7, A 7 to the carry flag, and the carry flag to A0
RR A	Rotate the A register one bit position to the right; bit A0 to bit A 7, A6 to A5, A5 to A4, A4 to A3, A3 to A2. A2 to Al, and At to A0
RRC A	Rotate the A register and the carry flag, as a ninth bit, one bit position to the right; bit A0 to the carry flag. carry flag to A 7, A 7 to A6, A6 to A5, A5 to A4, A4 to A3, A3 to A2, A2 to Al, and Al to A0
SWAP A	Interchange the nibbles of register A; put the high nibble in the low nibble position and the low nibble in the high nibble position

Note that no flags. other than the carry flag in RRC and RLC, are affected. If the carry is used as part of a rotate instruction, the state of the carry flag should be known before the rotate is done.

FIGURE *Register A Rotate Operations*

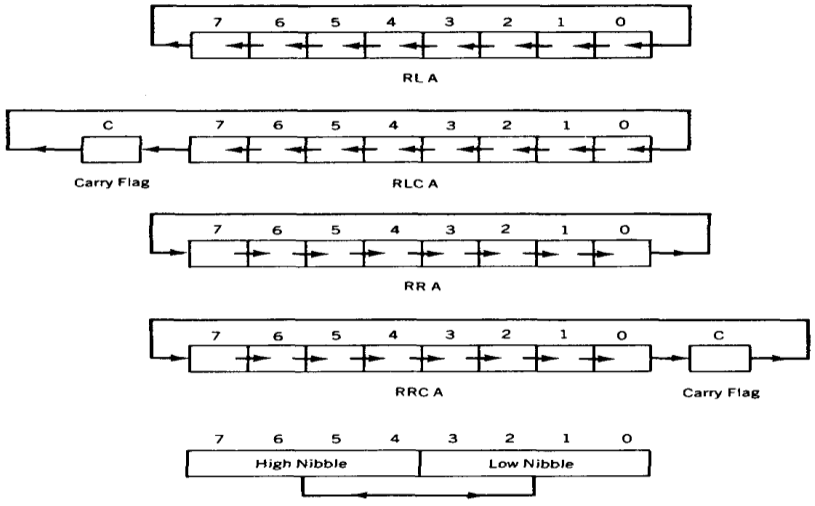


Figure Register a rotate operation

The following table shows examples of rotate and swap operations:

Mnemonic	Operation
OY A,#OA5h	A = I0l00I0lb = A5h
RR A	A = I I0I00I0b = D2h
RR A	A = 0l I0l00lb = 69h
RR A	A = I0I l0I00b = B4h
RR A	A = 0I0l I0I0b = 5Ah
SWAP A	A= I0I00I0lb = A5h
CLR C	C = 0; A = I0I00I0lb = A5h
RRC A	C = I; A= 0I0I00I0b = 52h
RRC A	C = 0; A = l0I0J00lb = A9h
RL A	A= 0I0I00llb = 53h
RL A	A = I0I00l I0b = A6h
SWAP A	C = 0; A = 0l 101010b = 6Ah
RLC A	C = 0; A = 1 I0I0l00b = D4h
RLC A	C = l; A= I0I0l000b = A8h
SWAP A	C = l; A= l000I0I0b = 8Ah

ARITHMETIC OPERATIONS

FLAGS

a key part of performing arithmetic operation is the ability to store certain results of those operation that affect the way in which the program operation example adding together two 1-byte number result in an 1-byte partial sum because the 8051 is an 8-bit machine.

INSTRUCTIONS AFFECTING FLAGS

The c, ac and over flags are arithmetical flags. they are set to 1 or clear to automatically depending on the outcomes of the instruction shown in table the instruction set includes all instruction that modify the flags and is not confined to arithmetic instruction

INSTRUCTION MNEMONIC	FLAGS AFFECTED		
ADD	C	AC	OV
ADDC	C	AC	OV
ANL C,direct	C		
CJNE	C		
CLR C	C = 0		
CPL C	C = $\overline{C}$		
DA A	C		
DIV	C = 0		OV
MOV C,direct	C		
MUL	C = 0		OV
ORL C,direct	C		
RLC	C		
RRC	C		
SETB C	C = 1		
SUBB	C	AC	OV

INCREMENTING AND DECREMENTING

The simplest arithmetic operations involve adding or subtracting a binary I and a number. These simple operations become very powerful when coupled with the ability to repeat the operation-that is, to "INCrement" or "OECrement" -until a desired result is reached.' Register,

Direct, and Indirect addresses may be INCremented or DECremented. No math flags (C, AC, OV) are affected.
The following table lists the increment and decrement mnemonics.

Mnemonic	Operation
INC A	Add a one to the A register
INC Rr	Add a one to register Rr
INC add	Add a one to the direct address
INC@Rp	Add a one to the contents of the address in Rp
INC DPTR	Add a one to the lti-bit DPTR
DEC A	Subtract a one from register A
DECRr	Subtract a one from register Rr
DEC add	Subtract a one from the contents of the direct address
DEC@Rp	Subtract a one from the contents of the address in register Rp

The following table shows examples of increment and decrement arithmetic operations:

Mnemonic	Operation
MOV A,#3Ah	A= 3Ah
DEC A	A= 39h
MOV R0,#15h	R0= 15h
MOV 15h,#12h	Internal RAM address 15h = 12h
INC@RO	Internal RAM address 15h = 13h
DEC I5h	Internal RAM address 15h = 12h
INC RO	R0= 16h
MOV 16h,A	Internal RAM address 16h = 39h
INC@RO	Internal RAM address 16h = 3Ah
MOV DPTR,#12	DPTR = 12FFh
FFh INC DPTR	DPTR = 1300h
DEC 83h	DPTR = 1200h (SFR 83h is the DPH byte)

Addition

All addition is done with the A register as the destination of the result. All addressing modes may be used for the source: an immediate number, a register, a direct address, and an indirect address. Some instructions include the carry flag as an additional source of a single bit that is included in the operation at the least significant bit position.
The following table lists the addition mnemonics.

Mnemonic	Operation
ADD A.#n	Add A and the immediate number n; put the sum in A
ADD A,Rr	Add A and register Rr; put the sum in A
ADD A,add	Add A and the address contents; put the sum in A
ADD A,@Rp	Add A and the contents of the address in Rp; put the sum in A

Note that the C flag is set to I if there is a carry out of bit position 7; it is cleared to 0 otherwise. The AC flag is set to 1 if there is a carry out of bit position 3; it is cleared otherwise. The OV flag is set to I if there is a carry out of bit position 7, but not bit position 6 or if there is a carry out of bit position 6 but not bit position 7, which may be expressed as the logical operation
OV = C7 XOR C6

Unsigned and Signed Addition

The programmer may decide that the numbers used in the program are to be unsigned numbers-that is, numbers that are 8-bit positive binary numbers ranging from 00h to FFh. Alternatively, the programmer may need to use both positive and negative signed numbers.

Signed numbers use bit 7 as a sign bit in the most significant byte (MSB) of the group of bytes chosen by the programmer to. represent the largest number to be needed by the program. Bits 0 to 6 of the MSB, and any other bytes, express the magnitude of the number. Signed numbers use a I in bit position 7 of the MSB as a negative sign and a 0 as a positive sign. Further, all negative numbers are not in true form, but are in 2's complement form. When doing signed arithmetic. the programmer must know how large the largest number is to be-that is, how many bytes are needed for each number.

In signed form, a single byte number may range in size from 10000000b, which is - I 28d to 01111111 b, which is +I 27d. The number 00000000b is 000d and has a positive sign, so there are 128d negative numbers and 128d positive numbers. The C and OV flags have been included in the 8051 to enable the programmer to use either numbering scheme.

Adding or subtracting unsigned numbers may generate a carry flag when the sum exceeds FFh or a borrow flag when the minuend is less than the subtrahend. The OV flag is not used for unsigned addition and subtraction. Adding or subtracting signed numbers can
lead to carries and borrows in a similar manner, and to overflow conditions due to the actions of the sign bits.

Unsigned Addition

Unsigned numbers make use of the carry flag to detect when the result of an ADD operation is a number larger than Ffb. If the carry is set to one after an ADD, then the carry can be added to a higher order byte so that the sum is not lost. For instance,
95d = 0101111 lb
189d = 10111101b
284d 1 000111 00b = 284d
The C flag is set to I to account for the carry out from the sum. The program could add the carry flag to another byte that forms the second byte of a larger number.

Signed Addition

Signed numbers may be added two ways: addition of like signed numbers and addition of unlike signed numbers. If unlike signed numbers are added, then it is not possible for the result to be larger than -128d or + 127d, and the sign of the result will always be correct. For example,
-001d 11111111b
+027d = 0001 1011b
+026d 0001 1010b = +026d
Here, there is a carry from bit 7 so the carry flag is I. There is also a carry from bit 6, and the OV flag is 0. For this condition, no action need be taken by the program to correct the sum.

If positive numbers are added, there is the possibility that the sum will exceed +I 27d, as demonstrated in the following example:

+ 100d = 01100100b
+050d = 001 10010b
+ 150d 10010110b = -106d
Ignoring the sign of the result, the magnitude is seen to be +22d which would be correct if we had some way of accounting for the + 128d, which, unfortunately, is larger than a single byte can hold. There is no carry from bit 7 and the carry flag is O; there is a carry from bit 6 so the OV flag is 1.
An example of adding two positive numbers that do not exceed the positive limit is:
+045d 00101101b
+075d = 01001011lb
+120d 01111000b = 120d
Note that there are no carries from bits 6 or 7 of the sum; the carry and OV flags are both 0.
The result of adding two negative numbers together for a sum that does not exceed the negative limit is shown in this example:
-030d = 11100010b
-050d = 11001110b
-080d 10110000b = -080d

Here, there is a carry from bit 7 and the carry flag is I; there is a carry from bit 6 and the OV flag is 0. These are the same flags as the case for adding unlike numbers; no corrections are needed for the sum.
When adding two negative numbers whose sum does exceed - l 28d, we have
-070d = 10111010b
-070d = 10111010b
-140d 01110100b = +116d
Or, the magnitude can be interpreted as - l 2d, which is the remainder after a carry out of - l 28d. In this example, there is a carry from bit position 7, and no carry from bit position 6, so the carry and the OV flags are set to I. The magnitude of the sum is correct; the sign bit must be changed to a I.
From these examples the programming actions needed for the C and OV flags are as follows:

FLAGS		**ACTION**
C	OV	
0	0	None
0	1	Complement the sign
1	0	None
1	1	Complement the sign

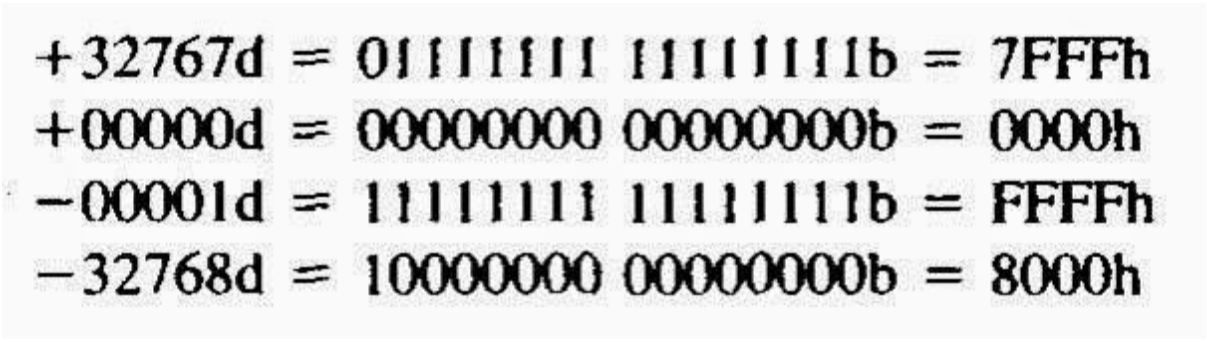
A general rule is that if the OV flag is set, then complement the sign. The OV flag also signals that the sum exceeds the largest positive or negative numbers thought to be needed in the program.

Multiple-Byte Signed Arithmetic

The nature of multiple-byte arithmetic for signed and unsigned numbers is distinctly different from single byte arithmetic. Using more than one byte in unsigned arithmetic means that carries or borrows are propagated from low-order to high-order bytes by the simple technique of

adding the carry to the next highest byte for addition and subtracting the borrow from the next highest byte for subtraction.

Signed numbers appear to behave like unsigned numbers until the last byte is reached.
Foi a signed number, the seventh bit of the highest byte is the sign; if the sign is negative, then the entire number is in 2's complement form.
For example, using a two-byte signed number, we have the following examples:



Note that the lowest byte of the numbers 00000 and - 32768d are exactly alike, as are the lowest bytes for+ 32767d and -00001d.
For multi-byte signed number arithmetic, then, the lower bytes are treated as unsigned numbers. All checks for overflow are done only for the highest order byte that contains the sign. An overflow at the highest order byte is not usually recoverable. The programmer has made a mistake and probably has made no provisions for a number larger than planned. Some error acknowledgment procedure, or user notification, should be included in the program if this type of mistake is a possibility.

The preceding examples show the need to add the carry Hag to higher order bytes in signed and unsigned addition operations. Opcodes that accomplish this task are similar to the ADD mnemonics: A C is appended to show that the carry bit is added to the sum in bit position 0.

The following table lists the add with carry mnemonics:

Mnemonic	Operation
ADDC A,#n	Add the contents of A, the immediate number n, and the C Hag; put the sum in A
ADDC A,add	Add the contents of A, the direct address contents, and the C Hag; put the sum in A
ADDC A,Rr A	Add the contents of A, register Rr, and the C Hag; put the sum in A
DDC A,@Rp	Add the contents of A, the contents of the indirect address in Rp, and the C flag; put the sum in A

Note that the C, AC, and OY flags behave exactly as they do for the ADD commands.
The following table shows examples of ADD and ADDC multiple-byte signed arithmetic operations:

Mnemonic	Operation
MOV A,#1Ch	A= 1Ch
MOV R5,#0Alh	R5 = A1h
ADD A,R5	A = BDh; C = 0, OV = 0

ADD A,R5	A = 5Eh; C = 1, OV = 1
ADDC A,#10h	A = 6Fh; C = 0, OV = 0
ADDC A,#IOh	A = 7Fh; C = 0, OV = 0

Subtraction

Subtraction can be done by taking the 2's complement of the number to be subtracted, the subtrahend, and adding it to another number, the minuend. The 8051, however, has commands to perform direct subtraction of two signed or unsigned numbers. Register A is the destination address for subtraction. All four addressing modes may be used for source addresses. The commands treat the carry flag as a borrow and always subtract the carry flag as part of the operation.

The following table lists the subtract mnemonics.

Mnemonic	Operation
SUBB A,#n	Subtract immediate number n and the C flag from A; put the result in A
SUBB A,add	Subtract the contents of add and the C flag from A; put the result in A
SUBB A,Rr	Subtract Rr and the C flag from A; put the result in A
SUBB A,@Rp	Subtract the contents of the address in Rp and the C flag from A; put the result in A

Note that the C flag is set if a borrow is needed into bit 7 and reset otherwise. The AC flag is set if a borrow is needed into bit 3 and reset otherwise. The OV flag is set if there is a borrow into bit 7 and not bit 6 or if there is a borrow into bit 6 and not bit 7. As in the case for addition, the OV Flag is the XOR of the borrows into bit positions 7 and 6.

Unsigned and Signed Subtraction

Again, depending on what is needed, the programmer may choose to use bytes as signed or unsigned numbers. The carry flag is now thought of as a borrow flag to account for situations when a larger number is subtracted from a smaller number. The OV flag indicates results that must be adjusted whenever two numbers of unlike signs are subtracted and the result exceeds the planned signed magnitudes.

Unsigned Subtraction

Because the C flag is always subtracted from A along with the source byte, it must be set to 0 if the programmer does not want the flag included in the subtraction. If a multi-byte subtraction is done, the C flag is cleared for the first byte and then included in subsequent higher byte operations.

The result will be in true form, with no borrow if the source number is smaller than A, or in 2's complement form, with a borrow if the source is larger than A. These are not signed numbers, as all eight bits are used for the magnitude. The range of numbers is from positive 255d (C = 0, A= FFh) to negative 255d (C = I, A= 0th).

The following example demonstrates subtraction of larger number from a smaller number:

015d = 0000111 lb

SUBB 100d = 01 01100100b

-085d 1 10101011b= 171d

The C flag is set to I, and the OV flag is set to 0. The 2's complement of the result is 085d.

The reverse of the example yields the following result:

100d = 01100100b

015d = 00001111b

085d 01010101b = 085d

The C flag is set to 0, and the OV flag is set to 0. The magnitude of the result is in true form.

Signed Subtraction

As is the case for addition, two combinations of unsigned numbers are possible when subtracting: subtracting numbers of like and unlike signs. When numbers of like sign are subtracted, it is impossible for the result to exceed the positive or negative magnitude limits of +I 27d or - I 28d, so the magnitude and sign of the result do not need to be adjusted, as shown in the following example:

+ 100d = 01100100b (Carry flag = 0 before SUBB)

SUBB +126d = 0111110b

-026d 11100110b = -026d

There is a borrow into bit positions 7 and 6; the carry flag is set to I, and the OV flag is cleared.

The following example demonstrates using two negative numbers:

-061d = 1100001 lb (Carry flag = 0 before SUBB)

SUBB -116d = 11000011b

+055d 00110111b = +55d

There are no borrows into bit positions 6 or 7, so the OY and carry flags are cleared to zero.

An overflow is possible when subtracting numbers of opposite sign because the situation becomes one of adding numbers of like signs, as can be demonstrated in the following example:

-099d = 10011101b (Carry flag = 0 before SUBB)

SUBB + 100d = 01 01100100b

-199d 00111001b = +057d

Here, there is a borrow into bit position 6 but not into bit position 7; the OV flag is set to I, and the carry flag is cleared to 0. Because the OV flag is set to I, the result must be adjusted. In this case, the magnitude can be interpreted as the 2's complement of 7 ld, the remainder after a carry out of l 28d from I 99d. The magnitude is correct, and the sign needs to be corrected to a 1 .

The following example shows a positive overflow:

+087d = 0101011 lb (Carry flag = 0 before SUBB)

SUBB -052d = 11001100b

+139d 1000101 lb = -I 17d

There is a borrow from bit position 7, and no borrow from bit position 6; the OV flag and the carry flag are both set to I. Again the answer must be adjusted because the OV flag is set to one. The magnitude can be interpreted as a +01 Id, the remainder from a carry out of I 28d. The sign must be changed to a binary 0 and the OV condition dealt with.

The general rule is that if the OV flag is set to I, then complement the sign bit. The OV flag also signals that the result is greater than - I 28d or +I 27d.

Again, it must be emphasized: When an overflow occurs in a program, an error has been made in the estimation of the largest number needed to successfully operate the program. Theoretically, the program could resize every number used, but this extreme procedure would tend to hinder the performance of the microcontroller.

Note that for all the examples in this section, it is assumed that the carry flag = 0 before the SUBB. The carry flag must be 0 before any SUBB operation that depends upon C = 0 is done.

The following table lists examples of SUBB multiple-byte signed arithmetic operations:

Mnemonic	Operation
MOV 0D0h,#00h	Carry flag= 0
MOV A,#3Ah	A= 3Ah
MOV 45h,#13h	Address 45h = 13h
SUBB A,45h	A = 27h; C = 0, OV = 0
SUBB A,45h	A = 14h; C = 0, OV = 0
SUBB A,#80h	A = 94h; C = 1 , OV = 1
SUBB A,#22h	A= 71h; C = 0, OV = 0
SUBB A,#0FFh	A = 72h; C = 1 , OV = 0

Multiplication and Division

The 8051 has the capability to perform 8-bit integer multiplication and division using the A and B registers. Register B is used solely for these operations and has no other use except as a location in the SFR space of RAM that could be used to hold data. The A register holds one byte of data before a multiply or divide operation, and one of the result bytes after a multiply or divide operation.

Multiplication and division treat the numbers in registers A and B as unsigned. The programmer must devise ways to handle signed numbers.

Multiplication

Multiplication operations use registers A and B as both source and destination addresses for the operation. The unsigned number in register A is multiplied by the unsigned number in register B, as indicated in the following table:

Mnemonic	Operation
MULAB	Multiply A by B: put the low-order byte of the product in A, put the high-order byte in B

The OV flag will be set if Ax B > FFh. Setting the OV flag does not mean that an error has occurred. Rather, it signals that the number is larger than eight bits, and the programmer needs to inspect register B for the high-order byte of the multiplication operation. The carry flag is always cleared to 0.

The largest possible product is FEO I h when both A and B contain FFh. Register A contains 0lh and register B contains FEh after multiplication of FFh by FFh. The OV flag is set to 1 to signal that register B contains the high-order byte of the product; the carry flag is 0.

The following table gives examples of MUL multiple-byte arithmetic operations:

Mnemonic	Operation
MOV A,#7Bh	A= 7Bh
MOV OFOh.#02h	B = 02h
MULAB	A = 00h and B = F6h: OV Flag = 0
MOV A,#0FEh	A = FEh
MULAB	A = 14h and B = F4h: OV Flag =1

Division

Division operations use registers A and B as both source and destination addresses for the operation. The unsigned number in register A is divided by the unsigned number in register B, as indicated in the following table:

Mnemonic	Operation
DIV AB	Divide A by B; put the integer part of quotient in register A and the integer part of the remainder in B

The OV flag is cleared to 0 unless B holds 00h before the DIV. Then the 0V flag is set to I to show division by 0. The contents of A and B, when division by 0 is attempted, are undefined. The carry flag is always reset.

Division always results in integer quotients and remainders, as shown in the following example:

$$\frac{A = 213d}{B = 017d} = \begin{array}{l}12 \text{ (quotient) and } 9 \text{ (remainder)} \\ 213\ [(12 \times 17) + 9]\end{array}$$

When done in hex:

$$\frac{A = 0D5h}{B = 011h} = C \text{ (quotient) and } 9 \text{ (remainder)}$$

The following table lists examples of DIV multiple-byte arithmetic operations:

Mnemonic	Operation
MOV A,#0FFh	A = FFh (255d)
MOV 0F0h,#2Ch	B = 2C (44d)
DIV AB	A = 05h and B = 23h [255d = (5 x 44) + 35]
DIV AB	A = 00h and B = 05h (05d = (0 x 35) + 5]
DIV AB	A = 00h and B = 00h [00d = (0 x 5) + 0J
DIV AB	A = ?? and B = ??; OV flag is set to one

Decimal Arithmetic

Most 8051 applications involve adding intelligence to machines where the hexadecimal numbering system works naturally. There are instances, however, when the application involves interacting with humans, who insist on using the decimal number system. In such cases, it may be more convenient for the programmer to use the decimal number system to represent all numbers in the program.

Four bits are required to represent the decimal numbers from 0 to 9 (0000 to 1001) and the numbers are often called Binary coded decimal (BCD) numbers. Two of these BCD numbers can then be packed into a single byte of data.

The 8051 does all arithmetic operations in pure binary. When BCD numbers are being used the result will often be a non-BCD number, as shown in the following example:

$$\begin{array}{rl} 49BCD = & 01001001b \\ +38BCD = & 00111000b \\ \hline 87BCD & 10000001b = 81BCD \end{array}$$

Note that to adjust the answer, an 06d needs to be added to the result.

The opcode that adjusts the result of BCD addition is the decimal adjust A for addition (DA A) command, as shown in the following table:

Mnemonic	Operation
DA A	Adjust the sum of two packed BCD numbers found in A register; leave the adjusted number in A.

The C flag is set to I if the adjusted number exceeds 99BCD and set to 0 otherwise. The DA A instruction makes use of the AC flag and the binary sums of the individual binary nibbles to adjust the answer to BCD. The AC flag has no other use to the programmer and no instructions-other than a MOV or a direct bit operation to the PSW-affect the AC flag.

It is important to remember that the DA A instruction assumes the added numbers were in BCD before the addition was done. Adding hexadecimal numbers and then using DA A will not convert the sum to BCD.

The DA A opcode only works when used with ADD or ADDC opcodes and does not give correct adjustments for SUBB, MUL or DIV operations. The programmer might best consider the ADD or ADDC and DA A as a single instruction and use the pair automatically when doing BCD addition in the 8051.

The following table gives examples of BCD multiple-byte arithmetic operations:

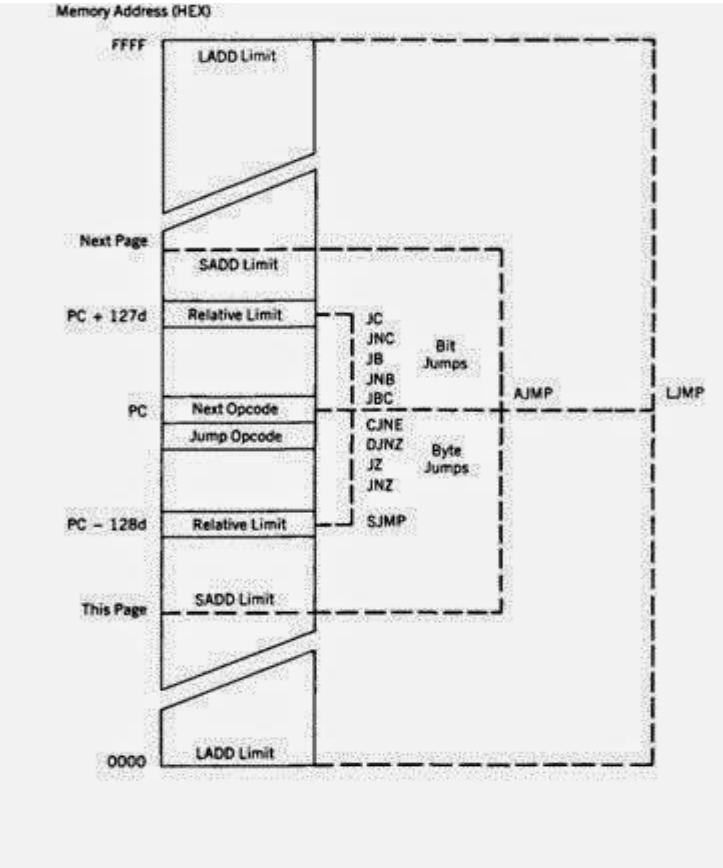
Mnemonic	Operation
MOV A,#42h	A= 42BCD
ADD A,#13h	A= 55h; C = 0
DA A	A= 55h; C = 0
ADD A,#17h	A= 6Ch; C = 0
DA A	A = 72BCD; C = 0
ADDC A,#34h	A= A6h; C = 0
DA A	A = 06BCD; C =1
ADDC A,#1 lh	A = l SBCD; C = 0
DA A	A = !&BCD; C = 0

The Jump and Call* Program *Range

A jump or call instruction can replace the contents of the program counter with a new program address number that causes program execution to begin at the code located at the new address. The difference, in bytes, of this new address from the address in the program where the jump or call is located is called the range of the jump or call. For example, if a jump instruction is located at program address 0100h, and the jump causes the program counter to become 0 I 20h, then the range of the jump is 20h bytes.

Jump or call instructions may have one of three ranges: a relative range of +I 27d, -128d bytes from the instruction following the jump or call instruction; an absolute range on the same 2K byte page as the instruction following the jump or call; or a long range of any address from 0000h to FFFFh, anywhere in program memory. Figure 6.1 shows the relative range of all the jump instructions.

FIGURE Jump Instruction Ranges



Relative Range

Jumps that replace the program counter contents with a new address that is greater than the address of the instruction following the jump by I 27d or less than the address of the instruction following the jump by I 28d are called relative jumps. They are so named because the address that is placed in the program counter is relative to the address where the jump occurs. If the absolute address of the jump instruction changes, then the jump address changes also hut remains the same distance away from the jump instruction. The address following the jump is used to calculate the relative jump because of the action of the PC. The PC is incremented to point to the next instruction before the current instruction is executed. Thus, the PC is set to the following address before the jump instruction is executed, or in the vernacular: "before the jump is taken."

Relative jumping has two advantages. First, only one byte of data need be specified, either in positive format for jumps ahead in the program or in 2's complement negative format for jumps behind. The jump address displacement byte can then be added to the PC to get the absolute address. Specifying only one byte saves program bytes and speeds up program execution. Second. the program that is written using relative jumps can be located anywhere in the program address space without re-assembling the code to generate absolute addresses.

The disadvantage of using relative addressing is the requirement that all addresses jumped be within a range of+ 127d, -128d bytes of the jump instruction. This range is not a serious problem. Most jumps form program loops over short code ranges that are within the relative address capability. Jumps are the only branch instructions that can use the relative range.

If jumps beyond the relative range are needed, then a relative jump can be done to another relative jump until the desired address is reached. This need is better handled, however. by the jumps that are covered in the next sections.

Short Absolute Range

Absolute range makes use of the concept of dividing memory into logical divisions called "pages." Program memory may be regarded as one continuous stretch of addresses from 0000h to FFFFh. Or, it may be divided into a series of pages of any convenient binary size. such as 256 bytes, 2K bytes, 4K bytes, and so on.

The 805 I program memory is arranged as 2K byte pages, giving a total of 32d (20h) pages. The hexadecimal address of each page is shown in the following table:

PAGE	ADDRESS(HEX)	PAGE	ADDRESS(HEX)	PAGE	ADDRESS(HEX)
00	0000-07FF	0B	5800-5FFF	16	B000-B7FF
01	0800-0FFF	oc	6000-67FF	17	8800-BFFF
02	1000-17FF	0D	6800-6FFF	18	C000-C7FF
03	1800-lFFF	0E	7000-77FF	19	C800-CFFF
04	2000-27FF	0F	7800-7FFF	lA	0000-07FF
05	2800-2FFF	10	8000-87FF	1B	0800-0FFF
06	3000-37FF	11	8800-8FFF	lC	E000-E7FF
07	3800-3FFF	12	9000--97FF	10	E800-EFFF
08	4000-47FF	13	9800-9FFF	lE	F0000-F7FF
09	4800-4FFF	14	A000-A7FF	1 F	F800-FFFF
0A	5000-57FF	15	A800-AFFF		

Inspection of the page numbers shows that the upper five bits of the program counter hold the page number, and the lower eleven bits hold the address within each page. An absolute address is formed by taking the page number of the instruction following the branch and attaching the absolute page range address of eleven bits to it to form the 16-bit address.

Branches on page

boundaries occur when the jump or call instruction finishes at X7FFh or XFFFh. The next instruction starts at X800h or X000h, which places the jump or call address on the same page as the next instruction after the jump or call. The page change presents no problem when branching ahead but could be troublesome if the branch is backwards in the program. The assembler should flag such problems as errors, so adjustments can be made by the programmer to use a different type of range.

Absolute range addressing has the same advantages as relative addressing; fewer bytes are needed and the code is relocatable as long as the relocated code begins at the start of a page. Absolute addressing has the advantage of allowing jumps or calls over longer programming distances than does relative addressing.

long Absolute Range

Addresses that can access the entire program space from 0000h to FFFFh use long range addressing. Long-range addresses require more bytes of code to specify and are relocatable only at the beginning of 64K byte pages. Since we are limited to a nominal ROM address range of 64K bytes, the program must be re-assembled every time a long-range address changes and these branches are not generally relocatable.

Long-range addressing has the advantage of using the entire program address space available to the 8051. It is most likely to be used in large programs.

Jumps

The ability of a program to respond quickly to changes in conditions depends largely upon the number and types of jump instructions available to the programmer. The 8051 has a rich set of

jumps that can operate at the bit and byte levels. These jump opcodes are one reason the 8051 is such a powerful microcontroller.

Jumps operate by testing for conditions that are specified in the jump mnemonic. If the condition is true, then the jump is taken-that is, the program counter is altered to the address that is part of the jump instruction. If the condition is false, then the instruction immediately following the jump instruction is executed because the program counter is not altered. Keep in mind that the condition of true does not mean a binary I and that false does not mean binary 0. The condition specified by the mnemonic is either true or false.

Bit Jumps

Bit jumps all operate according to the status of the carry flag in the PSW or the status of any bit-addressable location. All bit jumps are relative to the program counter.

Jump instructions that test for bit conditions are shown in the following table:

Mnemonic	Operation
JC radd	Jump relative if the carry flag is set to 1
JNC radd	Jump relative if the carry flag is reset to 0
JB b,radd	Jump relative if addressable bit is set to 1
JNB b,radd	Jump relative if addressable bit is reset to 0
JBC b.radd	Jump relative if addressable bit is set, and clear the addressable bit to 0

Note that no flags are affected unless the bit in JBC is a flag bit in the PSW. When the bit used in a JBC instruction is a port bit, the SFR latch for that port is read, tested, and altered.

The following program example makes use of bit jumps:

ADDRESS	MNEMONIC	COMMENT
LOOP:	MOV A,#l0h	; A = l0h
	MOV R0, A	;R0 = l0h
ADDA:	ADD A.R0	;add R0 to A
	JNC ADDA	;if the carry flag is 0. then no carry is
		; true; jump to address ADDA; jump until A
		;is F0h; the C flag is set to
		;l on the next ADD and no carry is
		; false: do the next instruction
	MOV A,#l0h	;A = l0h; do program again using JNB
ADDR:	ADD A.R0	;add R0 to A (R0 already equals !0h)
	JNB 0D7h.ADDR	;D7h is the bit address of the carry flag
	JBC 0D7h,L00P	;the carry bit is l; the jump to L00P
		;is taken. and the carry flag is cleared
		;to 0

Byte Jumps

Byte jumps-jump instructions that test bytes of data-behave as bit jumps. If the condition that is tested is true, the jump is taken; if the condition is false.the instruction after the jump is executed. All byte jumps are relative to the program counter.
The following table lists examples of byte jumps:

Mnemonic	Operation
CJNE A,add,radd	Compare the contents of the A register with the contents of the direct address; if they are not equal, then jump to the relative address; set the carry flag to I if A is less than the contents of the direct address; otherwise, set the carry flag to 0
CJNE A,#n,radd	Compare the contents of the A register with the immediate number n; if they are not equal, then jump to the relative address; set the carry flag to I if A is less than the number; otherwise, set the carry flag to 0
CJNE Rn,#n,radd	Compare the contents of register Rn with the immediate number n; if they are not equal, then jump to the relative address; set the carry flag to I if Rn is less than the number; otherwise, set the carry flag to 0
CJNE @Rp,#n,radd	Compare the contents of the address contained in register Rp to the number n; if they are not equal, then jump to the relative address; set the carry flag to I if the contents of the address in Rp are less than the number; otherwise, set the carry flag to 0
DJNZ Rn.radd	Decrement register Rn by I and jump to the relative address if the result is not zero; no flags are affected
DJNZ add.radd	Decrement the direct address by I and jump to the relative address if the result is not O; no flags are affected unless the direct address is the PSW
JZ radd	Jump to the relative address if A is 0; the flags and the A register are not changed
JNZ radd	Jump to the relative address if A is not O; the flags and the A register are not changed

Note that if the direct address used in a DJNZ is a port, the port SFR is decremented and tested for 0.

Unconditional Jumps

Unconditional jumps do not test any bit or byte to determine whether the jump should be taken. The jump is always taken. All jump ranges are found in this group of jumps. and these are the only jumps that can jump to any location in memory.
The following table shows examples of unconditional jumps:

Mnemonic	Operation
JMP@A+DPTR	Jump to the address formed by adding A to the DPTR; this is an unconditional jump and will always be done; the address can be anywhere in program memory; A, the DPTR, and the flags are unchanged

AJMP sadd	Jump to absolute short range address sadd; this is an unconditional jump and is always taken; no flags are affected
LJMP !add	Jump to absolute long range address ladd; this is an unconditional jump and is always taken; no flags are affected
SJMP radd	Jump to relative address radd; this is an unconditional jump and is always taken; no flags are affected
NOP	Do nothing and go to the next instruction; NOP (no operation) is used to waste time in a software timing loop; or to leave room in a program for later additions; no flags are affected

The following program example uses byte and unconditional jumps:

ADDRESS	MNEMONIC	COMMENT
	.0RG 0100h	;begin program at 0100h
BGN:	MOV A,#30h	;A = 30h
	MOV 50h,#00h	;RAM location 50h = 00h
AGN:	CJNE A,50h,AEQ	;compare A and the contents of 50h in RAM
	SJMP NXT	;SJMP will be executed if (50h) = 30h
AEQ:	DJNZ 50h.AGN	;count RAM location 50h down until (50h)
	NOP	;A; (50h) will reach 30h before 00h
NXT:	MOV R0,#0FFh	;R0 = FFh
DWN:	DJNZ R0.0WN	;count R0 to 00h; l00p here until done
	MOV A.R0	;A = R0 = 00h
	JNZ ABIG	;the jump will not be taken
	JZ AZR0	;the jump will be taken
HERE:	NOP	;this address will not be reached
	ORG l000h	;start this segment of program code at
		;l000h
AZRO:	MOV A,#08h	;A = 0Bh (code at 1000.lh)
	MOV DPTR, #l000h	;DPTR = l000h (code at 1002,3,4h)
	JMP @A+DPTR	;jump to location 1008h (code at 1005h)
	NOP	; (code at 1006h)
	NOP	; (code at 1007h)
ABIG:	AJMP AZRO	; (code at 1008h, all code on page 2)

Calls and Subroutines

The life of a microcontroller would be very tranquil if all programs could run with no thought as to what is going on in the real world outside. However, a microcontroller is specifically intended to interact with the real world and to react, very quickly, to events that require program attention to correct or control.

A program that does not have to deal unexpectedly with the world outside of the microcontroller could be written using jumps to alter program flow as external conditions require. This sort of program can determine external conditions by moving data from the port

pins to a location and jumping on the conditions of the port pin data. This technique is called "polling" and requires that the program does not have to respond to external conditions quickly. (Quickly means in microseconds; slowly means in milliseconds.)

Another method of changing program execution is using "interrupt" signals on certain external pins or internal registers to automatically cause a branch to a smaller program that deals with the specific situation. When the event that caused the interruption has been dealt with, the program resumes at the point in the program where the interruption took place. Interrupt action can also be generated using software instructions named calls.

Call instructions may be included explicitly in the program as mnemonics or implicitly included using hardware interrupts. In both cases, the call is used to execute a smaller, stand-alone program, which is termed a routine or, more often, a subroutine.

Subroutines

A subroutine is a program that may be used many times in the execution of a larger program. The subroutine could be written into the body of the main program everywhere it is needed, resulting in the fastest possible code execution. Using a subroutine in this manner has several serious drawbacks.

Common practice when writing a large program is to divide the total task among many programmers in order to speed completion. The entire program can be broken into smaller parts and each programmer given a part to write and debug. The main program
can then call each of the parts, or subroutines. that have been developed and tested by each individual of the team.

Even if the program is written by one individual, it is more efficient to write an oft-used routine once and then call it many times as needed. Also, when writing a program, the programmer does the main part first. Calls to subroutines, which will be written later, enable the larger task to be defined before the programmer becomes bogged down in the details of the application.

Finally, it is quite common to buy "libraries" of common subroutines that can be called by a main program. Again, buying libraries leads to faster program development.

Calls and the Stack

A call, whether hardware or software initiated, causes a jump to the address where the called subroutine is located. At the end of the subroutine the program resumes operation at the opcode address immediately following the call. As calls can be located anywhere in the program address space and used many times, there must be an automatic means of storing the address of the instruction following the call so that program execution can continue after the subroutine has executed.

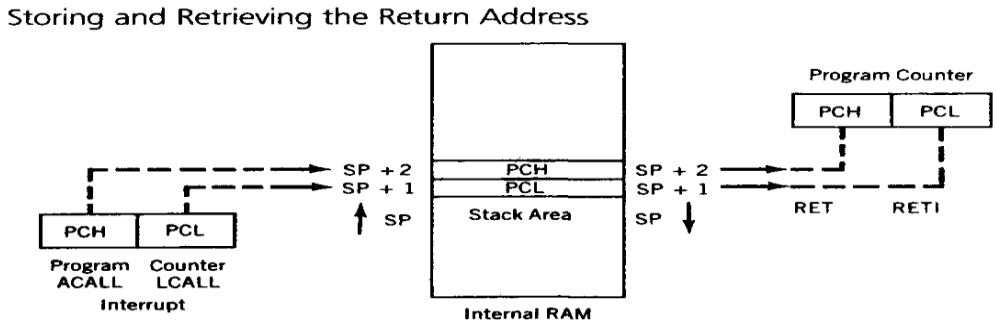
The stack area of internal RAM is used to automatically store the address, called the return address, of the instruction found immediately after the call. The stack pointer register holds the address of the last space used on the stack. It stores the return address above this space, adjusting itself upward as the return address is stored. The terms "stack" and "stack pointer" are often used interchangeably to designate the top of the stack area in RAM that is pointed to by the stack pointer.

Figure 6.2 diagrams the following sequence of events:

1. A call opcode occurs in the program software, or an interrupt is generated in the ardware circuitry.
2. The return address of the next instruction after the call instruction or interrupt is found in the program counter.
3. The return address bytes are pushed on the stack, low byte first.

4. The stack pointer is incremented for each push on the stack.
5. The subroutine address is placed in the program counter.
6. The subroutine is executed.
7. A RET (return) opcode is encountered at the end of the subroutine.

FIGURE Storing and Retrieving the Return Address



8. Two pop operations restore the return address to the PC from the stack area in internal RAM.
9. The stack pointer is decremented for each address byte pop.

All of these steps are automatically handled by the 8051 hardware. It is the responsibility of the programmer to ensure that the subroutine ends in a RET instruction and that the stack does not grow up into data areas that are used by the program.

Calls and Returns

Mnemonic	Operation
ACALL sadd	Call the subroutine located on the same page as the address of the opcode immediately following the ACALL instruction: push the address of the instruction immediately after the call on the stack
LCALL Ladd	Call the subroutine located anywhere in program memory space; push the address of the instruction immediately following the call on the stack
RET	Pop two bytes from the stack into the program counter

Note that no Hags are affected unless the stack pointer has been allowed to erroneously reach the address of the PSW special-function register.

Interrupts and Returns

As mentioned previously, an interrupt is a hardware-generated call. Just as a call opcode can be located within a program to automatically access a subroutine, certain pins on the 805 l can cause a call when external electrical signals on them go to a low state. Internal operations of the timers and the serial port can also cause an interrupt call to take place.

The subroutines called by an interrupt are located at fixed hardware addresses discussed in Chapter 2. The following table shows the interrupt subroutine addresses.

INTERRUPT	ADDRESS (HEX) CALLED
IE0	0003
TF0	0008
IE1	0013
TF1	0018
SERIAL	0023

When an interrupt call takes place, hardware interrupt disable Hip-Hops are set to prevent another interrupt of the same priority level from taking place until an interrupt return instruction has been executed in the interrupt subroutine. The action of the interrupt routine is shown in the table below.

Mnemonic	Operation
RETI	Pop two bytes from the stack into the program counter and reset the interrupt enable Hip-Hops

Note that the only difference between the RET and RETI instructions is the enabling of the interrupt logic when RETI is used. RET is used at the ends of subroutines called by an opcode. RETI is used by subroutines called by an interrupt.

The following program example use a call to a subroutine.

ADDRESS	MNEMONIC	COMMENT
MAIN:	MOV 8lh,#30h	:set the stack pointer to 30h in RAM
	LCALL SUB	: push address of N0P; PC = #SUB; SP
	NOP	:return from SUB to this opcode
	……	
	……	
SUB:	MOV A,#45h	;SUB loads A with 45h and returns
	RET	;pop return address to PC; SP = 30h

UNIT-V

APPLICATIONS

Keyboards

The predominant interface between humans and computers is the keyboard. These range in complexity from the "up-down" buttons used for elevators to the personal computer QWERTY layout, with the addition of function keys and numeric keypads. One of the first mass uses for the microcontroller was to interface between the keyboard and the main processor in personal computers. Industrial and commercial applications fall somewhere in between these extremes, using layouts that might feature from six to twenty keys.

The one constant in all keyboard applications is the need to accommodate the human user. Human beings can he irritable. They have litT1e tolerance for machine failure; watch what happens when the product isn't ejected from the vending machine. Sometimes they are bored, or even hostile, towards the machine. The hardware designer has to select keys that will survive in the intended environment. The programmer must write code that will anticipate and defeat inadvertent and also deliberate attempts by the human to confuse the program. It is very important to give instant feed hack to the user that the key hit has been acknowledged by the program. By the light a light, beep a beep, display the key hit, or whatever, the human user must know that the key has been recognized. Even feedback sometimes is not enough; note the behavior of people at an elevator. Even if the "up" light is lit when we arrive, we will push it again to let the machine know that 'Tm here too."

Human Factors

The keyboard application program must guard against the following possibilities:

More than one key pressed (simultaneously or released in any sequence)

Key pressed and held

Rapid key press and release

All of these situations can be addressed by hardware or software means; software, which is the most cost effective, is emphasized here.

Key Switch Factors

The universal key characteristic is the ability to bounce: The key contacts vibrate open and close for a number of milliseconds when the key is hit and often when it is released. These rapid pulses are not discernable to the human, hut they last a relative eternity in

the microsecond-dominated life of the microcontroller. Keys may be purchased that do not bounce, keys may be debounced with RS flip-flops, or debounced in software with time delays.

Keyboard Configurations

Keyboards are commercially produced in one of the three general hypothetical wiring configurations for a 16-key layout shown in Figure 8. I. The lead-per-key configuration is typically used when there are very few keys to be sensed. Since each key could tie up a port pin, it is suggested that the number be kept to 16 or fewer for this keyboard type. This configuration is the most cost effective for a small number of keys.

The X- Y matrix connections shown in Figure 8. I are very popular when the number of keys exceeds ten. The matrix is most efficient when arranged as a square so that N leads for X and N leads for Y can be used to sense as many as N2 keys. Matrices are the most cost effective for large numbers of keys.

Hypothetical Keyboard Wiring Configurations

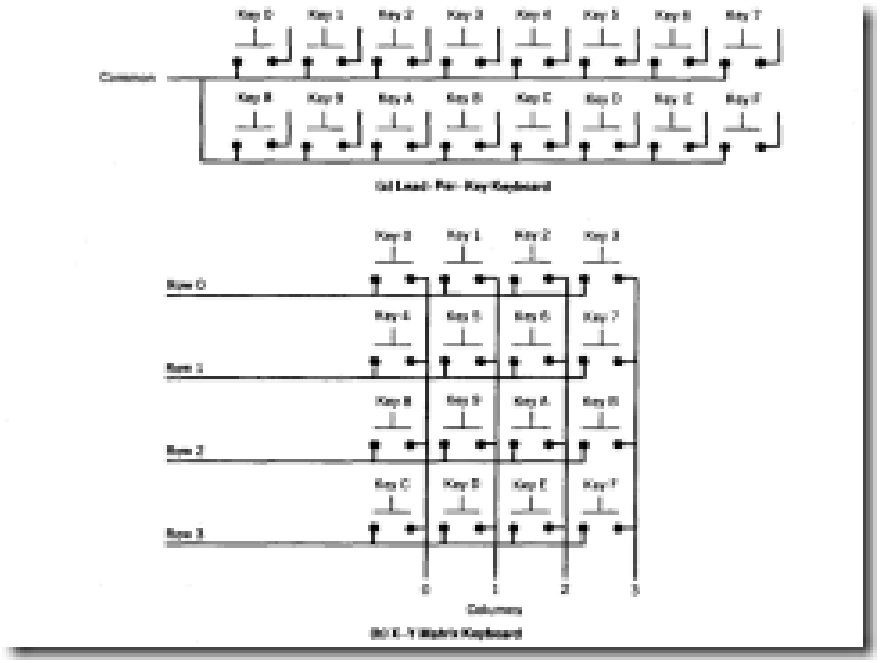
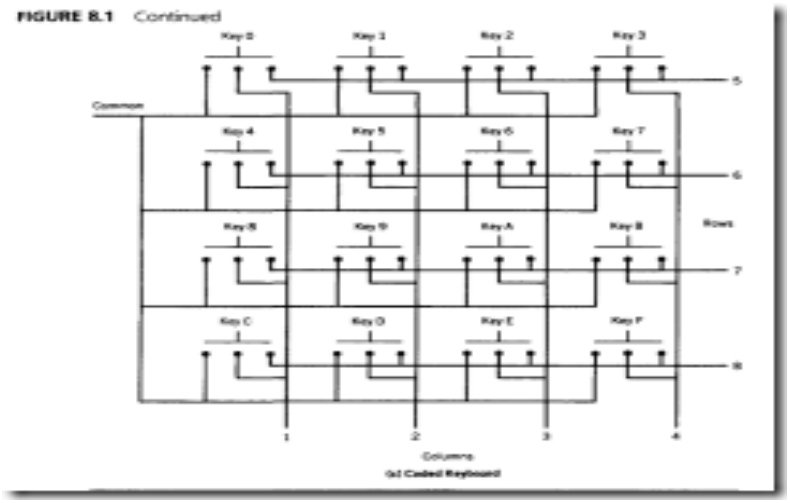


FIGURE 8.1 Continued



Coded keyboards were evolved originally for telephonic applications involving touchtone signaling. The coding permits multiple key presses to be easily detected. The quality and

durability of these keypads are excellent due to the high production volumes and intended use. They are generally limited to 16 keys or fewer, and tend to be the most expensive of all keyboard types.

Programs for Keyboards

Programs that deal with humans via keyboards approach the human and keyswitch factors identified in the following manner:

Bounce: A time delay that is known to exceed the manufacturer's specification is used to wait out the bounce period in both directions.

Multiple keys: Only patterns that are generated by a valid key pressed are accepted-all others are ignored-and the first valid pattern is accepted.

Key held: Valid key pattern accepted after valid de bounce delay; no additional keys accepted until all keys are seen to be up for a certain period of time.

Rapid key hit: The design is such that the keys are scanned at a rate faster than any human reaction time.

The last item brings up an important point: Should the keyboard be read as the program looks (software polled) or read only when a key has been hit (interrupt driven)?

In general, the smaller keyboards (lead-per-key and coded) can be handled either way. The common lead can be grounded and the key pattern read periodically. Or, the lows from each can be active-low ORed, as shown in Figure 8.2, and connected to one of the external $\overline{\text{INTX}}$ pins.

Matrix keyboards are scanned by bringing each X row low in sequence and detecting a Y column low to identify each key in the matrix. X- Y scanning can be done by using dedicated keyboard scanning circuitry or by using the microcontroller ports under program control. The scanning circuitry adds cost to the system. The programming approach takes processor time, and the possibility exists that response to the user may be sluggish if the program is busy elsewhere when a key is hit. Note how long your personal computer takes to respond to a break key when it is executing a print command, for instance. The choice between adding scanning hardware or program software is decided by how busy the processor is and the volume of entries by the user.

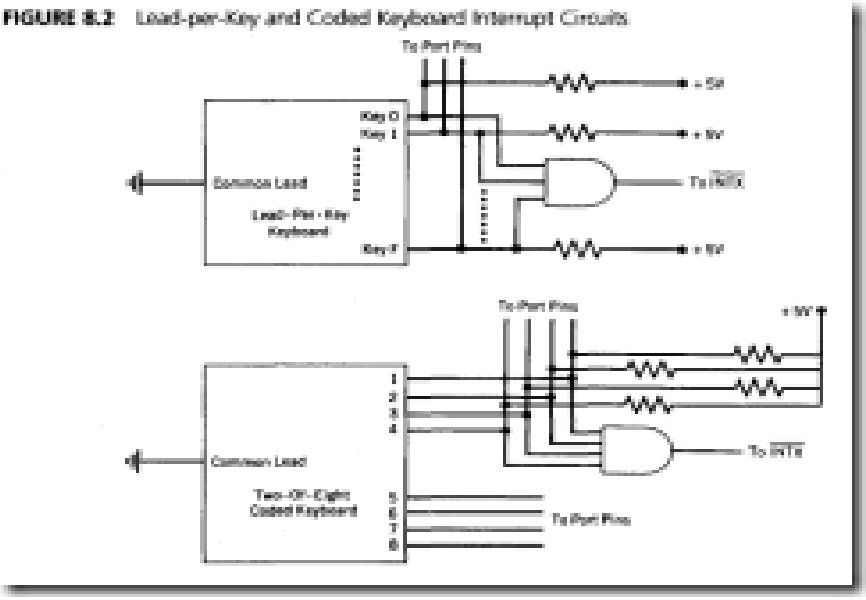


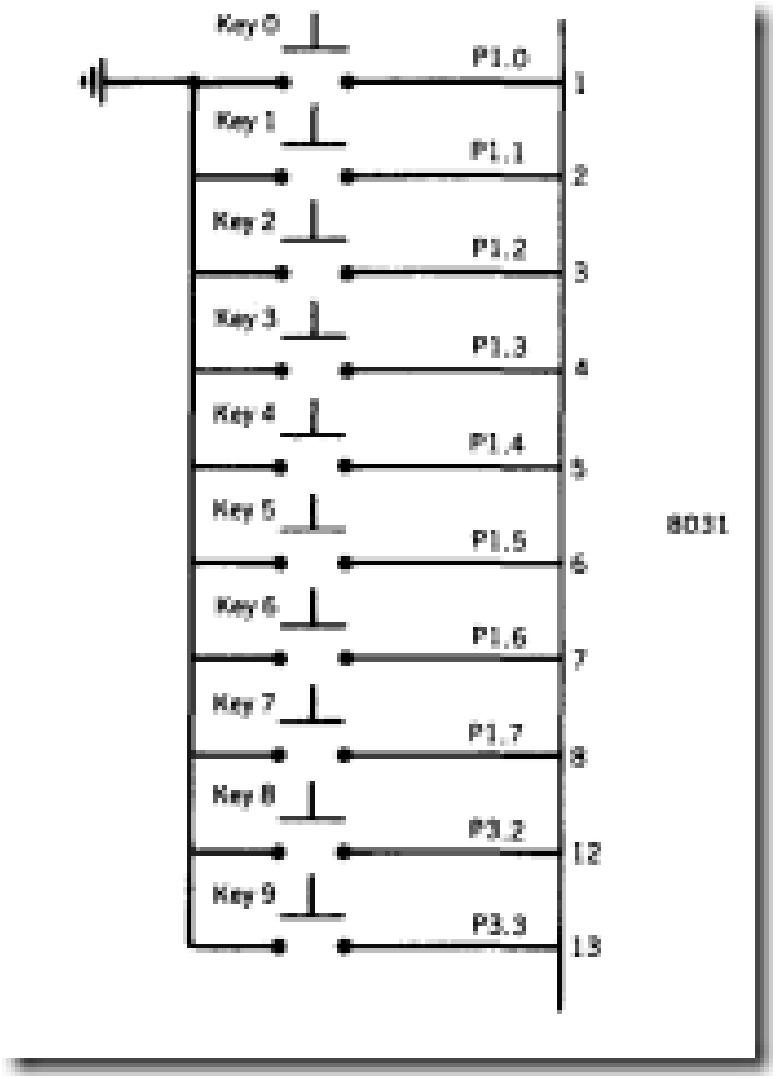
FIGURE 8.2 Load-per-Key and Coded Keyboard Interrupt Circuits

A Scanning Program for Small Keyboards

Assume that a lead-per-key keyboard is to be interfaced to the microcontroller. The keyboard has ten keys (0-9), and the de bounce time, when a key is pressed or released, is 20 milliseconds. The keyboard is used to select snacks from a vending machine, so the processor is only occupied when a selection is made. The program constanT1y scans the keyboard waiting for a key to be pressed before calling the vending machine actuator subroutine. The keys are connected to port I (0- 7) and ports 3.2 and 3.3 (8-9), as shown in Figure 8.3.

The 8031 works best when handling data in byte-sized packages. To save internal space, the ten-bit word representing the port pin configuration is converted to a single-byte number.

Because the processor has nothing to do until the key has been detected. the time delay "Softime" (see Chapter 7) is used to de bounce the keys.



Interrupt-Driven Programs for Small Keyboards

If the application is so time sensitive that the delays associated with debouncing and awaiting an "all-up" cannot be tolerated, then some form of interrupt must be used so that the main program can run unhindered.

A comproM ise may be made by polling the keyboard as the main program looks, but all time delays are done using timers so that the main program does not wait for a software delay. The "Get key" program can be modified to use a timer to generate the delays associated with the key down de bounce time and the "all-up" delay. The challenge associated with this approach is to have the program remember which delay is being timed out. Remembering which delay is in progress can be handled using a Hag bit, or one timer can be used to generate the key-down de bounce delay, and another timer to generate the key-up delay. The Hag approach is examined in the example given in this section.

The important feature of the program is that the main program will check a T1ag to see whether there is any keyboard activity. If the Hag is set, then the program finds the key stored in a RAM location and resets the Hag. The getting of the key is "transparent" to the main program; it is done in the interrupt program. The keyboard is still polled by the main program, but the interrupt program gets the key after that. The program named "Hard time" from Chapter 7 is used for the time delay. The keyboard user may notice some sluggishness in response if the main program takes so long to look that the keyboard initiation sequence is not done every quarter-second or so.

In key

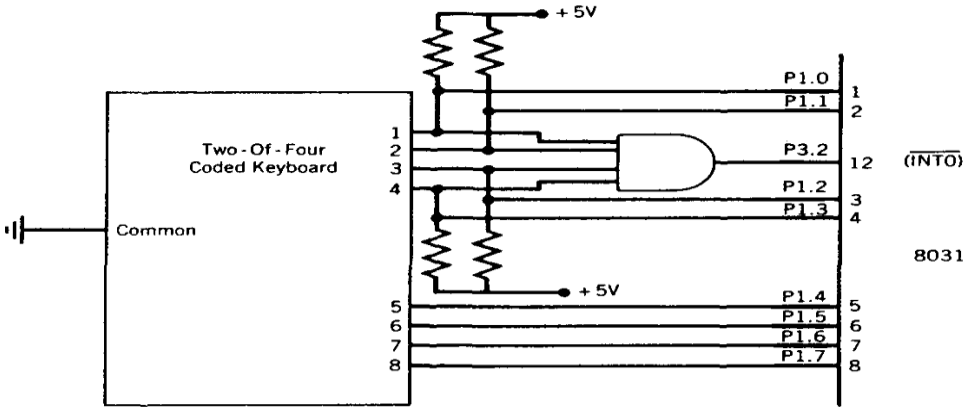
The program "lnkey" uses hardware timer T1 to generate all time delays. The keyboard sequence is initiated when a key is found to be down; otherwise, the program continues and checks for a key down in the next look. A key down initiates a de bounce time delay in timer T1 and sets a timer Hag to notify the interrupt program of the use of the timer. The interrupt program checks that a key is still down and is valid. Valid keys are stored, and a Hag is set that may be tested by the main program. The interrupt program then begins the key-up delay and sets the timer Hag to signify this condition. After each key-up delay, the interrupt program checks for all keys up. The time delay is reinitialized until all keys are up and the timer interrupts are halted.

Code key

The completely interrupt-driven small keyboard example given in this section requires no program action until a key has been pressed. Hardware must be added to attain a completely interrupt-driven event. The circuit of Figure 8.4 is used.

FIGURE 8.4 Keyboard Configuration Used for "Codekey" Program

Keyboard Configuration Used for "Codekey" Program



The keyboard is a two-of-eight type which codes the ten keys as follows:

KEY	CODE(HEX)
0	EE
1	ED
2	EB
3	E7
4	DE
5	DD
6	DB
7	D7
8	BE
9	BD

An inspection of the code reveals that each nibble has only one bit that is low for each key and that two of the eight bits are uniquely low for each key. If more than one key is pressed, then three or more bits go low, signaling an invalid condition. This popular scheme allows for up to 16 keys to be coded in this manner. Unlike the lead-per-key arrangement, only four of the lines must be active-low ORed to generate an interrupt.

The hardware serves to detect when any number of keys are hit by using an AND gate to detect when any nibble bit goes low. The high-to-low transition then serves to interrupt the microcontroller on port 3.2 ($\overline{INT0}$). The interrupt program reads the keys connected to port I and uses timer TO to generate the de bounce time and T1 for the keys-up delay. The total delay possible at 16 megahertz for the timers is 49.15 milliseconds, which covers the delay times used in the previous examples.

Program for a Large Matrix Keyboard

A 64-key keyboard, arranged as an 8-row by 8-column matrix will be interfaced to the 8051 rnicrocontroller, as shown in Figure 8.5. Port I will be used to bring each row low, one row at a time, using an 8-bit latch that is strobed by port 3.2. Pl will then read the 8-bit column pattern by enabling the tri-state buffer from port 3.3. A pressed key will have a unique row-column pattern of one row low, one column low. Multiple key presses are rejected by either an invalid pattern or a failure to match for three complete cycles. Each row is scanned at an interval of I millisecond, or an 8 millisecond cycle for the entire keyboard. A valid key must be seen to be

the same key for 3 cycles (24 milliseconds). There must then be three cycles with no key down before a new key will be accepted. The I millisecond delay between scans is generated by timer TO in an interrupt mode.

Big key

The "Big key" program scans an 8 x 8 keyboard matrix using TO to generate a periodic I ms delay in an interrupt mode. Each row is scanned via an external latch driven by port I and strobed by port 3.2. Columns are read via a tri-state buffer under control of port 3.3. Keys found to be valid are passed to the main program by setting the flag "new fig" and placing the key identifiers in locations "new row" and "new col." The main program resets "new flag" when the new key is fetched. R4 is used as a cycle counter for successful matches and up time cycles. R5 is used to hold the row scan pattern: only one bit low.

Displays

If keyboards are the predominant means of interface to human input, then visible displays are the universal means of human output. Displays may be grouped $\overline{INT0}$ three broad categories:

1. Single light(s)

2. Single character(s)

3. Intelligent alphanumeric

Single light displays include incandescent and, more likely, LED indicators that are treated as single binary points to be switched off or on by the program. Single character displays include numeric and alphanumeric arrays. These may be as simple as a sevensegment numeric display up to intelligent dot matrix displays that accept an 8-bit ASCII character and convert the ASCII code to the corresponding alphanumeric pattern. Intelligent alphanumeric displays are equipped with a built-in microcontroller that has been optimized for the application. Inexpensive displays are represented by multicharacter LCD windows, which are becoming increasingly popular in hand-held wands, factory fl00r terminals, and automotive dashboards. The high-cost end is represented by CRT ASCII terminals of the type commonly used to interface to a multi-user computer.

The individual light and intelligent single-character displays are easy to use. A port presents a bit or a character then strobes the device. The intelligent ASCII terminals are normally serial devices, which are the subject of Chapter 9.

The two examples in this section-seven-segment anp intelligent LCD displaysrequire programs of some length.

Seven-Segment Numeric Display

Seven-segment displays commonly contain LED segments arranged as an "8," with one common lead (anode or cathode) and seven individual leads for each segment. Figure 8.6 shows the pattern and an equivalent circuit representation of our example, a common cathode display. If more than one display is to be used, then they can be time multiplexed; the human eye can

not detect the blinking if each display is relit every JO milliseconds or so. The I 0 milliseconds is divided by the number of displays used to find the interval between updating each display.

The example examined here uses four seven-segment displays; the segment information is output on port I and the cathode selection is done on ports 3.2 to 3.5, as shown in

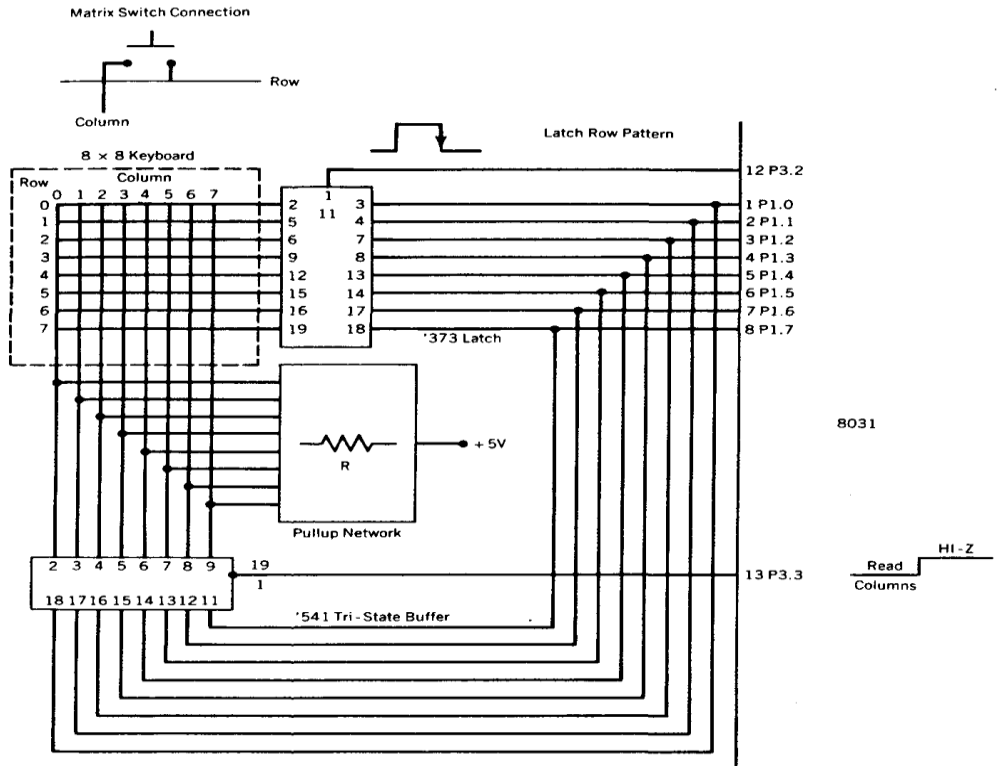


Figure A segment will he lit only if the segment line is brought high and the common cathode is brought low.

Transistors must be used to handle the currents required by the LEDs, typically io milliamperes for each segment and 70 milliamperes for each cathode. These are average current values; the peak currents will be four times as high for the 2.5 milliseconds each display is illuminated.

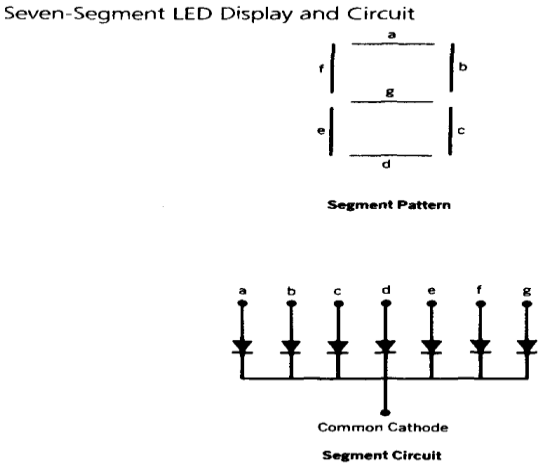
The program is interrupt driven by TO in a manner similar to that used in the program "Bigkey." The interrupt program goes to one of four two-byte character locations and finds the cathode segment pattern to be latched to port I and the anode pattern to be latched to port 3. The main program uses a lookup table to convert from a hex number to the segment pattern for that number. In this way, the interrupt program automatically displays whatever number the main program has placed in the character locations. The main program loads the character locations and is not concerned with how they are displayed.

Svnseg

The program "svnseg" displays characters found in locations "chi" to "ch4" on four common-cathode seven-segment displays. Port I holds the segment pattern from the low byte of chx; port 3 holds the cathode pattern from the high byte of chx. T0 generates a 2.5 ms delay interval between characters in an interrupt mode. The main program uses a lookup table to convert from

hex to a corresponding pattern. ro of bank one is dedicated as a pointer to the displayed character.

FIGURE Seven-Segment Display Circuit Used for "Svnseq" Program

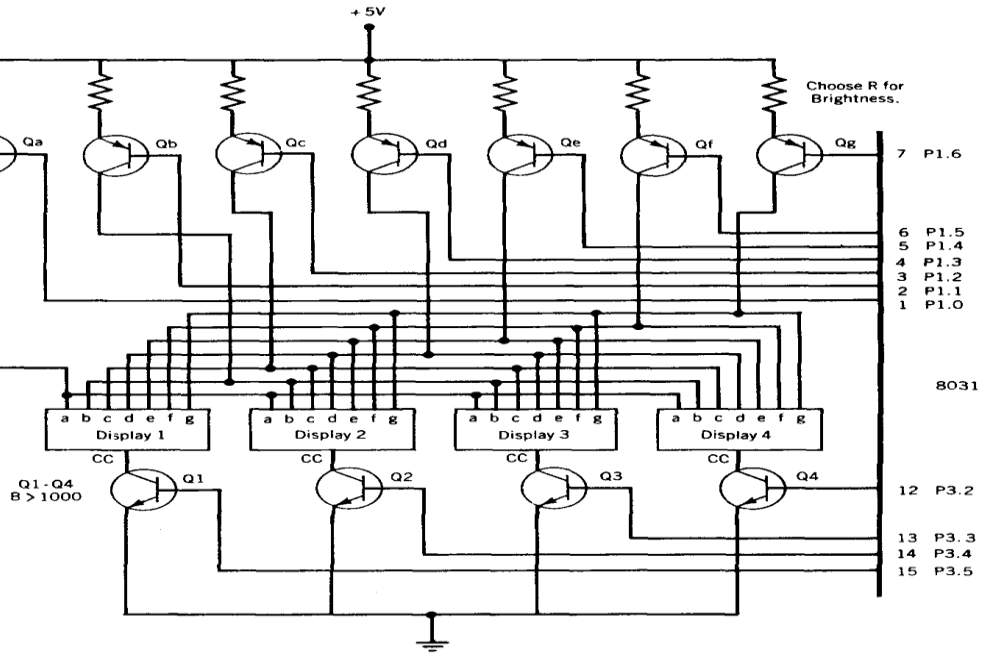


intelligent LCD Display

In this section, we examine an intelligent LCD display of two lines, 20 characters per line, that is interfaced to the 8051 . The protocol (handshaking) for the display is shown in Figure 8.8, and the interface to the 8051 in Figure 8.9.

The display contains two internal byte-wide registers, one for commands (RS = 0) and the second for characters to be displayed (RS = l). It also contains a user-programmed RAM area (the character RAM) that can be programmed to generate any desired character that can be formed using a dot matrix. To distinguish between these two data areas, the hex command byte 80 will be used to signify that the display RAM address 00h is chosen.

FIGURE Intelligent LCD Display



Port 1 is used to furnish the command or data byte, and ports 3.2 to 3.4 furnish regis ter select and read/write levels.

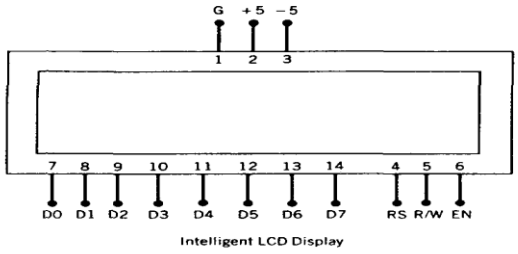
The display takes varying amounts of time to accomplish the functions listed in fissure 8.8. LCD bit 7 is monitored for a logic high (busy) to ensure the display is not""""" written. A sligh T1y more complicated LCD display (4 lines x 40 characters) is curn-nr}, being used in medical diagnostic systems to run a very similar program.

Lcdisp

The program "Icdisp" sends the message "hello" to an intelligent LCD display shown rn Figure 8.8. Port 1 supplies the data byte. Port 3.2 selects the command (0) or data t I 1 registers. Port 3.3 enables a read (0) or write (I) level, and port 3.4 generates an active. low-enable strobe.

FIGURE Intelligent LCD Circuit for "Lcdisp" Program

FIGURE Intelligent LCD Display



Intelligent LCD Display

BIT	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Function
	0	0	0	0	0	0	0	0	0	1	Clear LCD and memory, home cursor
	0	0	0	0	0	0	0	0	1	0	Clear and home cursor only
	0	0	0	0	0	0	0	1	I/O	S	Screen action as display character written S = 1/0: Shift screen/cursor I/O = 1/0: Cursor R/L, screen L/R
	0	0	0	0	0	0	1	D	C	B	D = 1/0: Screen on/off C = 1/0: Cursor on/off B = 1/0: Cursor Blink/Noblink
	0	0	0	0	0	1	S/C	R/L	0	0	S/C = 1/0: Screen/Cursor R/L = 1/0: Shift one space R/L
	0	0	0	0	1	DL	N	F	0	0	DL = 1/0: 8/4 Bits per character N = 1/0; 2/1 Rows of characters F = 1/0; 5X10/5X7 Dots/Character
	0	0	0	1	Character address						Write to character RAM Address after this
	0	0	1	Display data address							Write to display RAM Address after this
	0	1	BF	Current address							BF = 1/0: Busy/Notbusy
	1	0		Character byte							Write byte to last RAM chosen
	1	1		Character byte							Read byte from last RAM chosen

Pulse Measurement

Sensors used for industrial and commercial control applications frequenT1y produce pulses that contain information about the quantity sensed. Varying the sensor output frequency, using a constant duty cycle but variable frequency pulses to indicate changes in the measured variable, is most common. Varying the duration of the pulse width, resulting in constant frequency but variable duty cycle, is also used. In this section, we examine programs that deal with both techniques.

Measuring Frequency

Timers TO and T 1 can be used to measure external frequencies by configuring one timer as a counter and using the second timer to generate a timing interval over which the first can count. The frequency of the counted pulse train is.then

Unknown frequency = Counter/timer

For example, if the counter counts 200 pulses over an interval of .1 second generated by the timer. the frequency is

UF = 200/. 1 = 2000 Hz

Certain fundamental limitations govern the range of frequencies that can be measured. An input pulse must make a l-to-0 transition lasting two machine cycles. or f/24, to be counted. This restriction on pulse deviation yields a frequency of 667 kilohertz using our 16 megahertz crystal (assuming a square wave input).

The lowest frequency that can be counted is limited by the duration of the time interval generated, which can be exceedingly long using all the RAM to count timer rollovers (49.15 milliseconds x 2"32768). There is no practical limitation on the lowest frequency that can be counted.

Happily, most frequency variable sensors generate signals that fall inside of 0 to 667 kilohertz. Usually the signals have a range of 1,000 to io,000 hertz.

Our example will use a sensor that measures de voltage from 0 to 5 volts. At 0 V the sensor output is 1,000 hertz, and at full scale, or 5 volts, the sensor output is 6,000 hertz. The correspondence is I volt per 1,000 hertz, and we wish to be able to measure the voltage to the nearest .OJ V, or io hertz of resolution (assuming the sensor is this accurate). A timing interval of I second generates a frequency count accurate to the nearest I hertz, so an interval of . I s yields a count accurate to the nearest 1 0 hertz.

Another way to arrive at the desired timing interval, T, is to note that the desired accuracy is

$$\frac{.01\ \text{V}}{5\ \text{V}} = .002 = \frac{1}{512} = \frac{1}{2^9}$$

and that the range of the counter is from T x fmin to T x fmax, or a range of T x (fmax - fmin) from zero to full scale. The resolution of each counter bit is then

$$\text{LSB} = \frac{\text{T} \times (\text{fmax} - \text{fmin})}{2^n}$$

where n is the desired number of bits to be resolved. For our example, T = 512/5000 = . io24 seconds; . I second yields a sligh T1y better accuracy.

From earlier tries at generating decimal time delays in Chapter 7. it has been amply demonstrated that these cannot be done perfec T1y using a 16 megahertz crystal (. 75 micro-second count interval). We will be close enough to meet our requirements.

T1 is used in the auto-reload mode 2 to generate overflow interrupts every I 92 microseconds (256 x . 75 microseconds). These overflows are counted using R4 and R5 until .io0032 seconds have elapsed (52Jd overflows). For this example, TO is used as a counter to count the external

frequency that is fed to the port 3.4 (TO) pin during the T1 interval. Using the interval chosen, the range of counts in TO becomes

OV = io00 Hz x . io0032 s = io0d counts

5V = 6000 Hz x . 1 00032 s = 600d counts

.OJ V = ioHz x . io0032 s = 1 count

which meets the desired accuracy specification.

Freq

The program "freq" uses TO to count an external pulse train that is known to vary in frequency from io00 to 6000 hertz. T1 generates an exact time delay of 192 microseconds that is counted using registers R4 and R5 of bank I until T1 has overflowed 521d times, or a total delay of. io0032 seconds.

Pulse Width Measurement

Theoretically, if the input pulse is known to be a perfect square wave, the pulse frequency can be measured by finding the time the wave is high (Th). The frequency is then

$$UF = \frac{1}{Th \times 2}$$

If Th is 200 microseconds, for example, then UF is 2500 hertz. The accuracy of the measurement will fall as the input wave departs from a 50 percent duty cycle.

Timer X may be configured so that the internal clock is counted only when the corresponding $\overline{INT}X$ pin is high by setting the GATE X bit in TMOD. The accuracy of the measurement is within approximately one-half of the timer clock period, or .375 microsecond for a 16 megahertz crystal. This accuracy can only be attained if the measurement is started when the input wave is low and stopped when the input next goes low. Pulse widths greater than the capacity of the counter, which is 49.152 milliseconds for a 16 megahertz crystal, can be measured by counting the overflows of the timer flag and adding the final contents in the counter.

D/ A and A/D Conversions

Conversion between the analog and digital worlds requires the use of integrated circuits that have been designed to interface with computers. Highly intelligent converters are commercially available that all have the following essential characteristics:

Parallel data bus: tri-state, 8-bit

Control bus: enable (chip select), read/write, ready/busy

The choice the designer must make is whether to use the converter as a RAM memory location connected to the memory busses or as an I/O device connected to the ports. Once that choice is made, the set of instructions available to the programmer becomes limited. The memory location assignment is the most restrictive, having only MOVX available. The design could use the additional 32K RAM address space with the addition of circuitry for A 15. By enabling the RAM when A 15 is low, and the converter when A 15 is high, the designer could use the upper 32K RAM address space for the converter, as was done to expand port capacity by memory mapping in Chapter 7. All of the examples examined here are connected to the ports.

D/A Conversions

A generic R-2R type D/A converter, based on several commercial models, is connected to ports I and 3 as shown in Figure 8. I 0. Port I furnishes the digital byte to be converted to an analog voltage; port 3 controls the Conversion process. The converter has these features:

Vout = -Yref X (byte in/io0H), Yref = ± io V

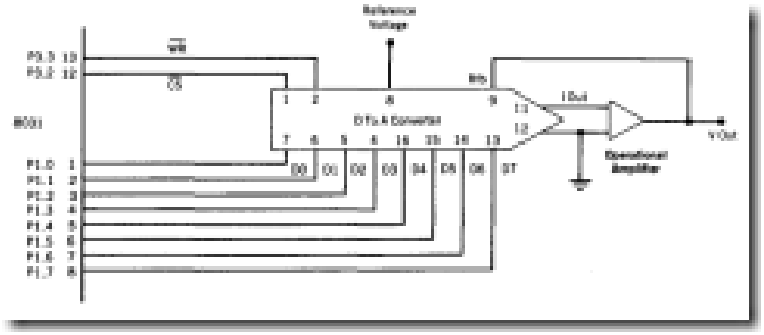
Conversion time: 5 µs

Control sequence: $\overline{CS}$ then $\overline{WR}$

For this example, a io00 hertz sine wave that will be generated can have a programmable frequency. Vref is chosen to be -io volts, and the wave will swing from +9.96 volts to O volt around a midpoint of 4.48 volts. The program uses a lookup table to generate the amplitude of each point of the sine wave; the time interval at which the converter is fed bytes from the table determines the wave frequency.

The Conversion time limits the highest frequency that can be generated using S sample point. In this example, the shortest period that can be used is

FIGURE 8.io DIA Converter Circuit for "Davcon" Program



The design tension is high frequency versus high resolution. For a io00 hertz wave, S could be 200d samples. In reality, we cannot use this many samples; the program cannot fetch the data, latch it to port I, and strobe port 3.3 in 5 microseconds. An inspection of the program will show that the time needed for a single wave point is 6 microseconds, and setting up for the next wave takes another 2.25 microseconds. S becomes 166d samples using the 6 microseconds interval. and the addition of 2.25 microseconds at the end of every wave yields a true frequency of io01. 75 hertz.

A/D Conversion

The easiest A/D converters to use are the "flash" types, which make Conversions based upon an array of internal comparators. The Conversion is very fast, typically in less than 1 microsecond. Thus, the converter can be told to start, and the digital equivalent of the input analog value will be read one or two instructions later. Modern successive approximation register (SAR) converters do not lag far behind, however, with Conversion times in the 2-4 microsecond range for eight bits.

At this writing, flash converters are more expensive (by a factor of two) than the traditional SAR types, but this cost differential should disappear within four years. Typical features of an eight-bit flash converter are

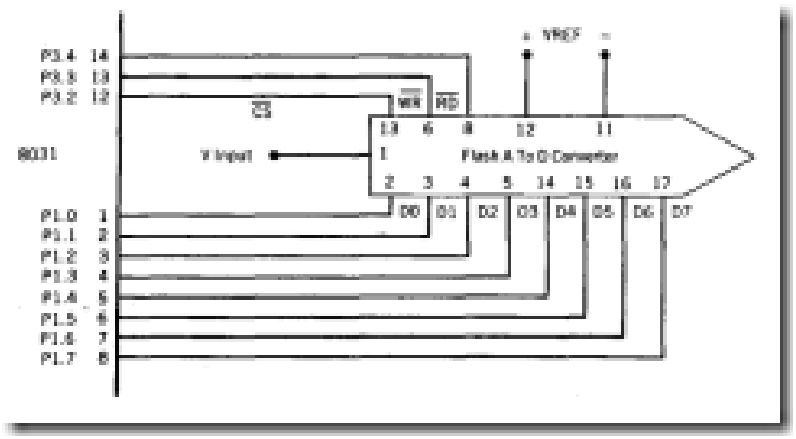
Data: Vin = Vref (-), data = 00h; Vin = Vref(+),data = FFh

Conversion time: 1 µs

Control sequence: $\overline{CS}$ then $\overline{WR}$ then $\overline{RD}$

An example circuit, using a generic flash converter, is shown in Figure 8. 11. Port I is used to read the byte value of the input analog voltage, and port 3 controls the Conversion.

FIGURE 8.11 A/D Converter Circuit for "Adconv" Program



A Conversion is started hy pulsing the write line low, and the data is read by bringing the read line low.

Our example involves the digitizing of an input waveform every l00d microseconds until io00d samples have been stored in external RAM.

Multiple Interrupts

The 8051 is equipped with two external interrupt input pins: $\overline{INT0}$ and INTI (P3.2 and P3.3). These are sufficient for small systems, but the need may arise for more than two interrupt points. There are many schemes available to multiply the number of interrupt points; they all depend upon the following strategies:

Connect the interrupt sources to a common line

Identify the interrupting source using software

Because the external interrupts are active low, the connections from the interrupt source to the $\overline{INTX}$ pin must use open-collector or tri-state devices.

An example of increasing the $\overline{INT0}$ from one to eight points is shown in Figure 8. 12.

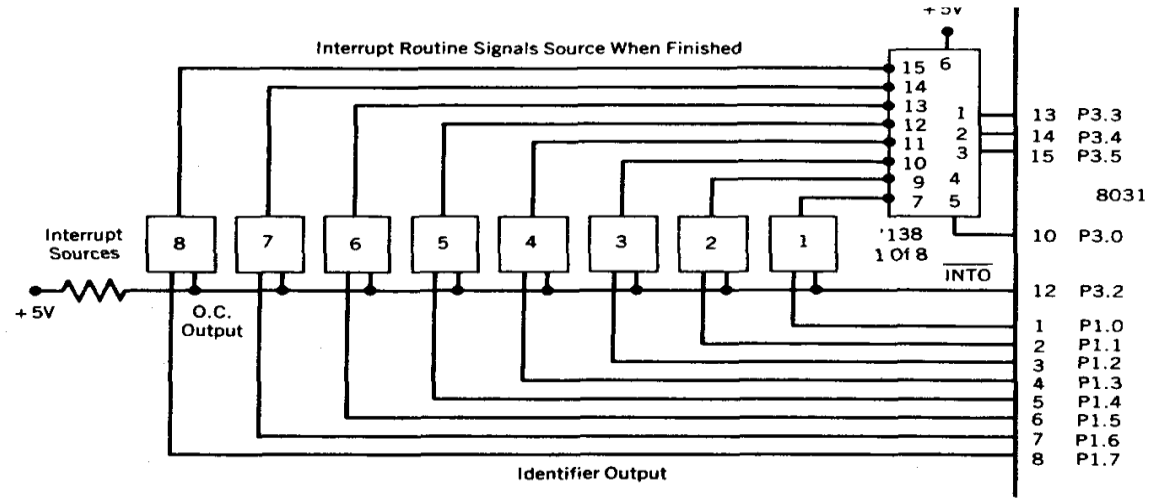
Each source goes to active low when an interrupt is desired. A corresponding pin on port I receives the identity of the interrupter. Once the interrupt program has handled the interrupt situation, the interrupter must receive an acknowledgment so that the interrupt line for that source can be Rought back to a high state. Port 3 pins 3.3, 3.4, and 3.5 supply, via a 3-to-8 decoder, the acknowledgment feedback signal to the pRoper interrupt source. The decoder is enabled by port pin 3.0.

Multiple and simultaneous interrupts can be handled by the program in as complex a manner as is desired. If there is no particular urgency attached to any of the interrupts then they can be handled as the port I pins are scanned sequentially for a low.

A simple priority system can be established whereby the most important interrupt sources are examined in the priority order, and the associated interrupt program is run until finished. An elaborate priority system involves ordering the priority of each source. The elaborate system acknowledges an interrupt immediately, thus resetting that source's interrupt line, and begins executing the particular interrupt program for that source. A new interrupt from a higher priority source forces the current interrupt program to be suspended and the new interrupter to be serviced.

To acknowledge the current interrupt in anticipation of another, it is necessary to also re-arm the $\overline{INTX}$ interrupt by issuing a "dummy" RETI instruction. The mechanism for accomplishing this task is illustrated in the program named "hipri." First, a low priority scheme is considered.

FIGURE Multiple-Source Interrupt Circuit Used in "Lepri" and "Hipri" pograms



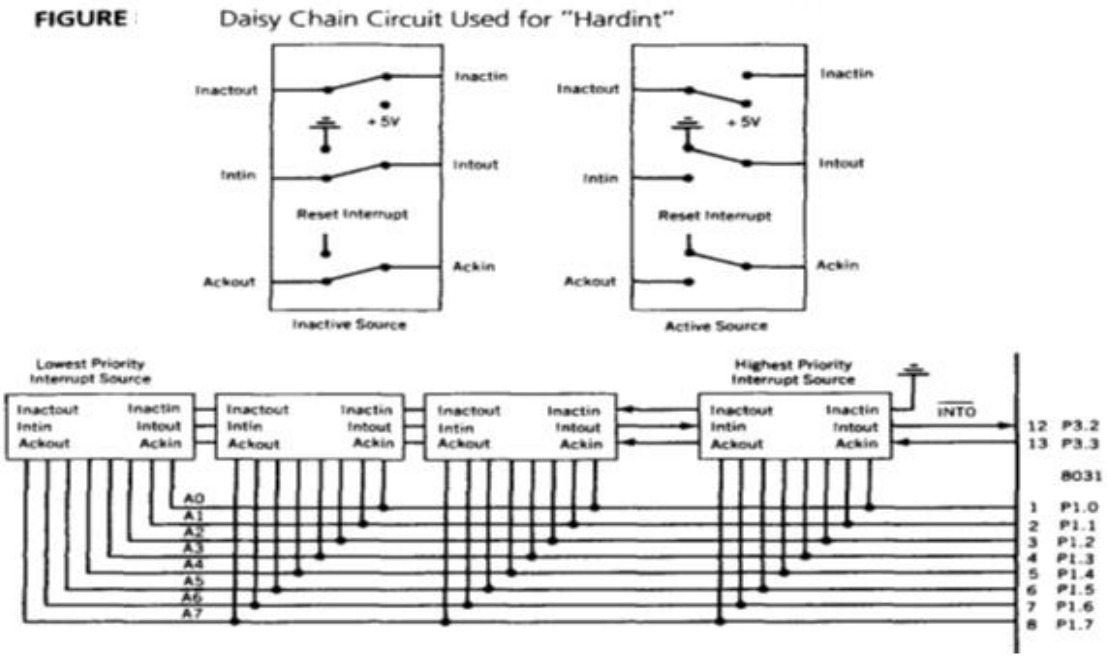
Hardware Circuits for Multiple Interrupts

Solutions to the expanded interrupt problem proposed to this point have emphasized using a minimal amount of external circuitry to handle multiple, overlapping interrupts. A hardware strategy, which can be expanded to cover up to 256 interrupt sources, is shown in Figure 8.13. This circuit is a version of the "daisy chain" approach, which has long been popular.

The overall philosophy of the design is as follows:

1. The most important interrupt source is physically connected first in the chain, with those of lesser importance next in line. Lower priority interrupt sources are "behind" (connected further from $\overline{INT0}$) those of a higher priority.

2. Each interrupting source can disable all signals from sources that are wired behind it. All sources that lose the INACTOUT signal (a low level) from the source(s) ahead of it will place their source address buffer in a tri-state mode until INACTOUT is restored.



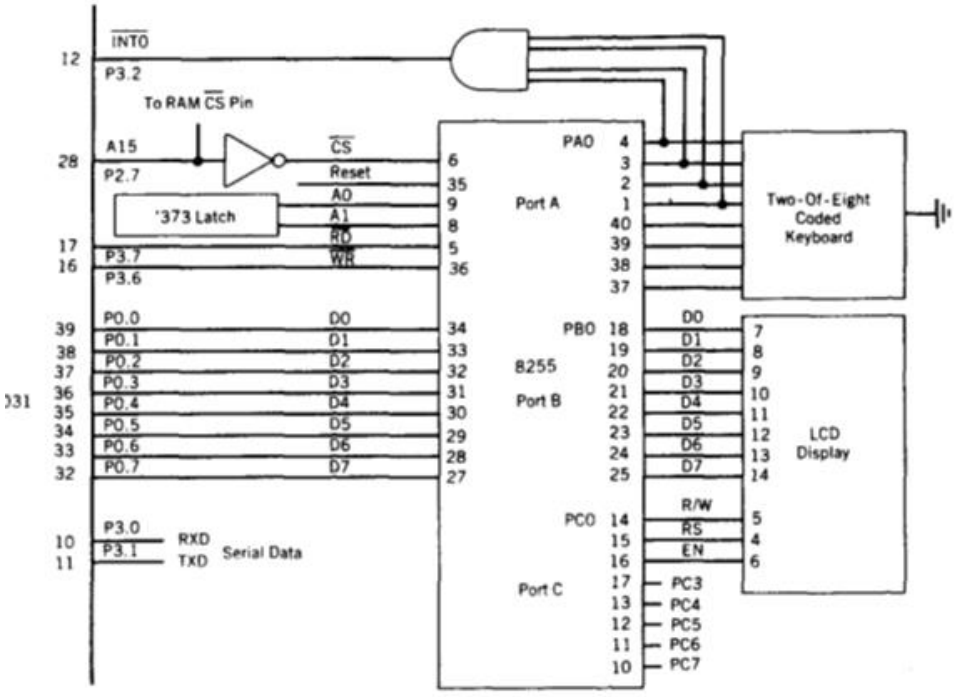
3. A requesting source pulls its $\overline{INT0}$ UT line low and places its 8-bit identifier on the tri-state bus connected to port I . The interrupt routine at the $\overline{INT0}$ vector location reads Pl and, using a lookup table, finds the address of the subroutine that handles that interrupt. The address is placed on the stack and a RETI executed to go to that routine and re-arm the interrupt structure.

4. The interrupt subroutine generates an ACKIN signal (a low-level pulse) to the source from the 805 I at the end of the subroutine; the source then removes $\overline{INT0}$ UT and the 8-bit source address. When an interrupt is acknowledged, the interrupting source must bring the $\overline{INT0}$ UT line high for at least one machine cycle so that the 8051 interrupt structure can recognize the next high-to-low transition on $\overline{INT0}$ .The software is very simple for this scheme. Any interrupt received is always of higher priority than the one now running, and the source address on port I enables rapid access to the interrupt subroutine.

Accomplishing this interrupt sequence requires that the source circuitry be complex or that the source contain some intelligence such as might be provided by a microcontroller.

The additional source hardware will entail considerable relative expense for each source. As the number of interrupt sources increases, system costs rise rapidly. At some point the designer should consider another microcontroller that has extensive interrupt capability.

FIGURE A Multi-Tasking Circuit Using Memory-mapped I/O



8255 I/O Ports Mapped At 8000, 8001, And 8002

Figure Multi-tasking circuit using memory mapping i/o