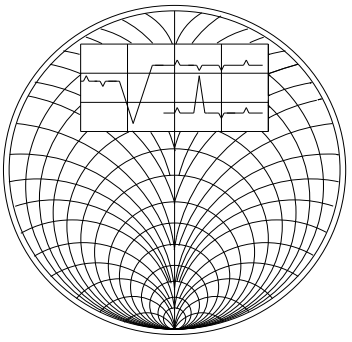
MICRO-51 EB

User Manual

Version 4.0

Technical Clarification /Suggestion :
✍ / ☎
Technical Support Division,
Vi Microsystems Pvt. Ltd.,
Plot No :75,Electronics Estate,
Perungudi,Chennai - 600 096,INDIA.
Ph: 91- 44-2496 3142, 91-44-2496 1852,
Mail : service@vimicrosystems.com,
Web : www.vimicrosystems.com



PREFACE

The ***MICRO-51 EB User Manual / Student Workbook*** has been devised as a User Manual common for both the Micropower-i trainer based on Intel 8031 / 8051 Microcontroller Micro-51 and Micro-51EB. It shall serve as a valuable aid to a learner, new to the software, hardware and interface programming aspects of the 8031 / 8051.

The manual has been formulated with consideration to the capabilities of a beginner and has a systematic organisation of all the chapter. Commencing with an introduction, each of the chapters has a whole complement of examples and exercises.

Chapter - 1 Serves as an introduction to the 8031 / 8051.

Chapter - 2 Which has been further segmented into various sections deals with the programming features of 8031 / 8051 under arithmetic, data transfer, stack operations, Bit manipulations, flags and such other heads.

Chapter - 3 gives you typical examples on hardware programming with the peripherals to further enhance your capability.

Chapter - 4 is on application examples based on the interfacing of different peripherals available on the trainer, which will give you an idea of the ways in which a microcontroller may be used to serve a particular purpose.

Chapter - 5 It Deals the ***"System Oriented Functions"*** for making a program as small and sense.

Chapter - 6 It gives details above the On-Chip interrupts of 8031 / 8051.

Chapter - 7 It is discussion on using serial communication for data transfer using our DATACOM.

While working through the exercises, please refer to the appendices given at the back of this manual for information on instruction set of 8031 / 8051. To promote your skills to higher levels, you can consult the titles quoted in the bibliography of references.

You shall no doubt emerge as a number one programmer, after working with the trainer along with this manual in all sincerity. Suggestions to enhance the substances provided in this manual are invited.

Write to:

The Customer-Support Division,
Vi Microsystems Pvt. Ltd.,
Plot No.75, Electronics Estate,
Perungudi, Chennai - 600 096.

Phone : **(044) 2496 1842, 2496 1852.**
Fax : **(044) 2496 1536.**
E-mail : **sales@vimicrosystems.com**

TABLE OF CONTENTS

LIST OF APPENDICES

APPENDIX

CHAPTER 1

8031/8051 PROGRAMMING OVERVIEW

OBJECTIVES:

a) To introduce the user to the concept of Microcontroller.

b) To introduce the user to the Basic Architecture of Intel 8031/8051.

c) To familiarise the user with the Memory, I/O and Register Architecture and the Addressing modes available on 8031/8051.

1.1 INTRODUCTION:

A Microcontroller consists of a powerful CPU tightly coupled with memory (RAM, ROM or EPROM), various I/O features such as Serial port(s), Parallel port(s), Timer/Counter(s), Interrupt Controller, Data Acquisition Interfaces - Analog to Digital Converter(ADC), Digital to Analog Converter(DAC), everything integrated onto a single Silicon chip.

It does not mean that any microcontroller should have all the above said features onchip. Depending on the need and area of application for which it is designed, the onchip features present in it may or may not include all the individual sections said above.

Any microcomputer system requires memory to store a sequence of instructions making up a program, parallel port or serial port for communicating with an external system, timer/counter for control purposes like generating time delays, baud rate for the serial port, apart from the controlling unit called the Central Processing Unit.

If a system is developed with a microprocessor, the designer has to go for external memory such as RAM, ROM or EPROM and peripherals and hence the size of the PCB (Printed Circuit Board) will be large enough to hold all the required peripherals. But, the microcontroller has got all these peripheral facilities on a single chip. So, development of a similar system with a microcontroller reduces PCB size and cost of the design.

One of the major differences between a microcontroller and a microprocessor is that a controller often deals with bits, not bytes as in the real world applications. For example, switch contacts can only be open or close, indicators should be lit or dark and motors can be either turned on or off and so forth.

INTEL MCS-51 FAMILY:

Intel has introduced a family of Microcontrollers called the **MCS-51**. This family comprises the 8031, 8051, 8052, 8751 and 8752 Microcontrollers. Our trainer kit supports all these microcontrollers. 8051, an 8-bit Single chip Microcontroller has got a powerful CPU optimised for control applications, 64K Program Memory address space, 64K Data Memory address space, 4K bytes of on-chip ROM (Read Only Memory), 128 bytes of on-chip RAM (Read/Write Memory), Four 8-bit bidirectional Parallel ports, one full-duplex (it can transmit and receive simultaneously) Serial port, two 16-bit timer/counters and an extensive interrupt structure.

8031 is the **ROMless version of 8051** whereas **8751** is the **EPROM version of 8051.** This is the main difference between 8031, 8051 and 8751. Since we have not used the On-chip ROM or EPROM in Micro-51, Micro-51 EB and Micropower-i based 8031/51 Piggy board, any of these three Microcontrollers can be used.

For individual line and control applications, 8051 is best suited as it incorporates a special set of instruction which are capable of bit addressing the onchip I/O features of 8051.

The Major Features of 8051 Are:

* 8-bit CPU optimised for control applications.
* Extensive Boolean processing (single-bit logic) capabilities.
* 64K Program Memory address space.
* 64K Data Memory address space.
* 4K bytes of on-chip Program Memory (in 8051 and 8751 only).
* 128 bytes of on-chip Data RAM.
* 32 bi-directional and individually addressable I/O lines.
* Two 16-bit Timer / Counters.
* Full Duplex UART (Universal Asynchronous Receiver / Transmitter).
* 6-source / 5-vector interrupt structure with two priority levels.
* On-chip Oscillator and Clock circuitry.

APPLICATIONS OF MICROCONTROLLERS:

Microcontrollers are designed for use in sophisticated real-time applications such as industrial control, instrumentation and intelligent computer peripherals.

Microcontrollers with ADC finds usage in data acquisition systems and closed-loop analog controllers. It permits considerable system integration by combining analog and digital I/O processing in the single chip.

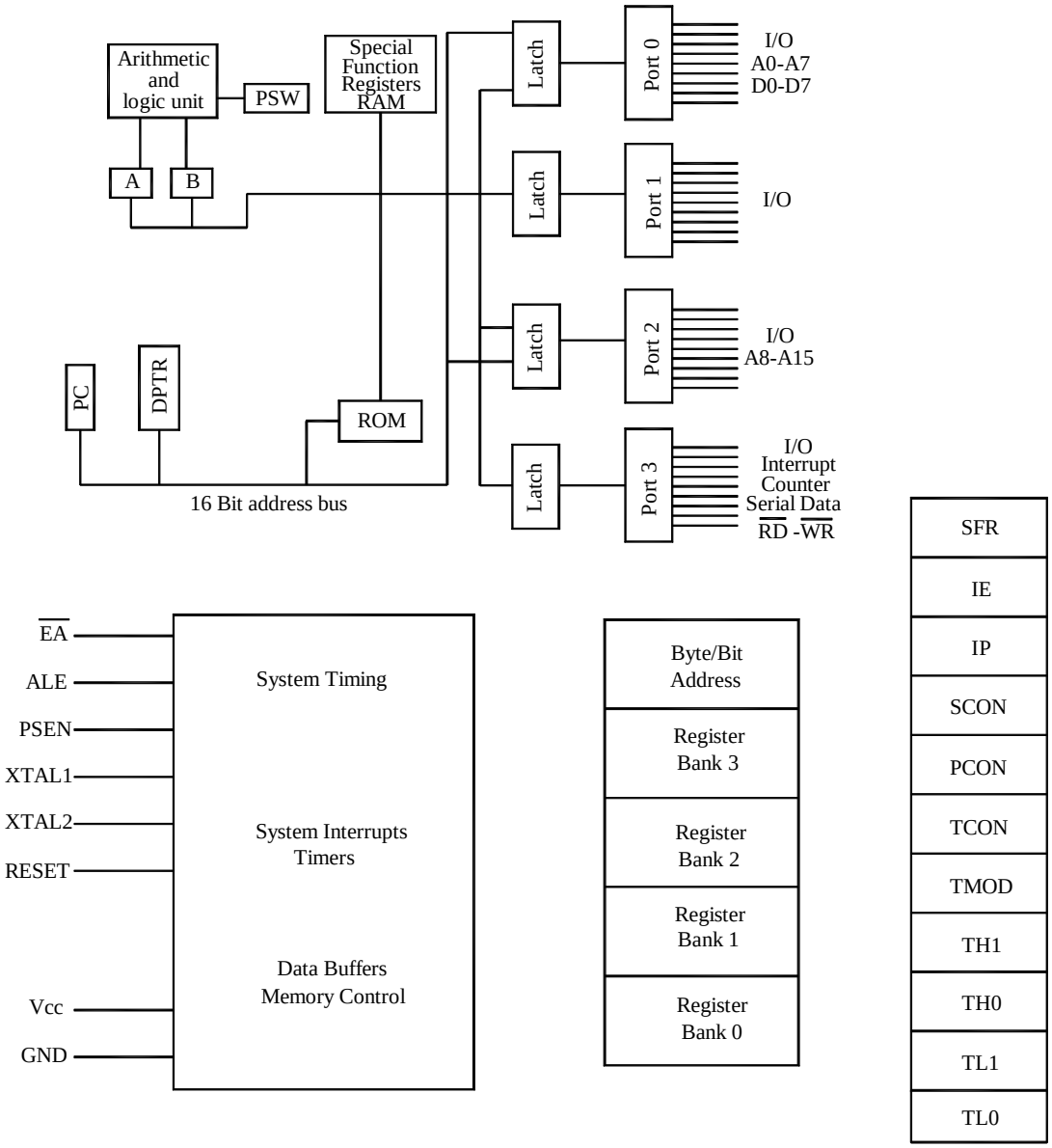
They are used in industrial applications to control Motor, Robotics, discrete and continuous process control, in missile guidance and control, in Medical instrumentation, Oscilloscopes, Telecommunications, Automobiles, for scanning a keyboard, driving an LCD (Liquid Crystal Display), for frequency measurements, period measurements, pulse width measurements and so on.

NOTE:

As it becomes unwidely to be constantly referring the three microcontrollers namely 8031, 8051 and 8751 by their individual names, we will adopt a convention of referring to them generically as **"8051"**, unless a specific name is being referred to.

1.2 THE 8051 MICROCONTROLLER ARCHITECTURE

The architecture of the 8051 family of microcontrollers is referred to as the MCS-51 architecture (Micro Controller Series-51), or sometimes simply as MCS-51.The microcontrollers have an 8-bit data bus. They are capable of addressing 64K of program memory and a separate 64K of data memory. The block diagram of 8051 microcontroller is shown below.



The 8051 have 4K of code memory implemented as on-chip *Read Only Memory* (ROM). The 8051 have 128 bytes of internal *Random Access Memory* (RAM). The 8051 has two timer/counters, a serial port, 4 general purpose parallel input/output ports, and interrupt control logic with five sources of interrupts. Besides internal RAM, the 8051 have various Special *Function Registers* (SFR), which are the control and data registers for on-chip facilities. The SFRs also include the accumulator, the B register, and the *Program Status Word* (PSW), which contains the CPU flags. Programming the various internal hardware facilities of the 8051 is achieved by placing the appropriate control words into the corresponding SFRs. The 8031 are similar to the 8051, except it lacks the on-chip ROM.

1.2.1 VARIOUS 8051 MICROCONTROLLERS

The 8051 is available in different memory types, such as UV-EPROM, flash, and NV-RAM, all of which have different part numbers. The UV-EPROM version of the 8051 is the 8751. Many companies including Philips, Atmel, Analog Devices & Cygnal manufactures the flash ROM version. The Atmel flash 8051 is called AT89C51 & the Analog Devices version is ADuC812. The NV-RAM version of the 8051 made by Dallas semiconductor is called DS5000. There is also the OTP (One time programmable) version of the 8051 made by various manufacturers. All these versions of 8051 (MCS-51 series) are called as '8051 Derivatives'.

MCS 51 Series of Microcontrollers are

INTEL	**–**	**805X, 803X, 875X, 835X, 8X152, 8X251**
ATMEL	**–**	**835X, 875X, 895X, T8X251, AT205X**
PHILIPS	**–**	**805X, 835X, 875X, 895X**
CYGNAL	**–**	**8051FXXX**
TI	**–**	**MSC121X**
ANALOG DEVICES	**–**	**ADuC812, 814, 816, 824, 831, 834, 844,848**

1.2.2 PIN DESCRIPTION OF THE 8051

Although 8051 family members (e.g. 8751, 89C51) come in different packages, such as DIP (dual in-line package), QFP (quad flat package), and LLC (Leadless chip carrier), they all have 40 pins that are dedicated for various function such as I/O, RD, WR, address, data and interrupts. It must be noted that some companies provide a less pin version of the 8051 with a reduced number of I/O ports for less demanding applications & user flexibility. ATMEL 892051 is the 20 pin 8051 derivative IC with less digital I/O lines & reduced flash memory compared to its 40 pin AT89C51 series. However, since the vast majority of developers use the 40-pin DIP package chip. And now RISC versions of 8051 ICs are also available with ATMEL & they called it as AVR microcontrollers & available size from 8-pin package.

Examining the following figure, note that of the 40 pins a total of 32 pins are set aside for the four ports P0, P1, P2 and P3, where each ports takes 8 pins. The rest of the pins are designated as Vcc, GND, XTAL1, XTAL2, RST, EA, ALE, and PSEN. Of these 8 pins, all members of the 8051 and 8031 families use six of them. In other words, they must be connected in order for the system to work, regardless of whether the microcontroller is of the

8051 or 8031 family. The other two pins PSEN- and ALE are used mainly in 8031 based systems. We first describe the function of each pin.

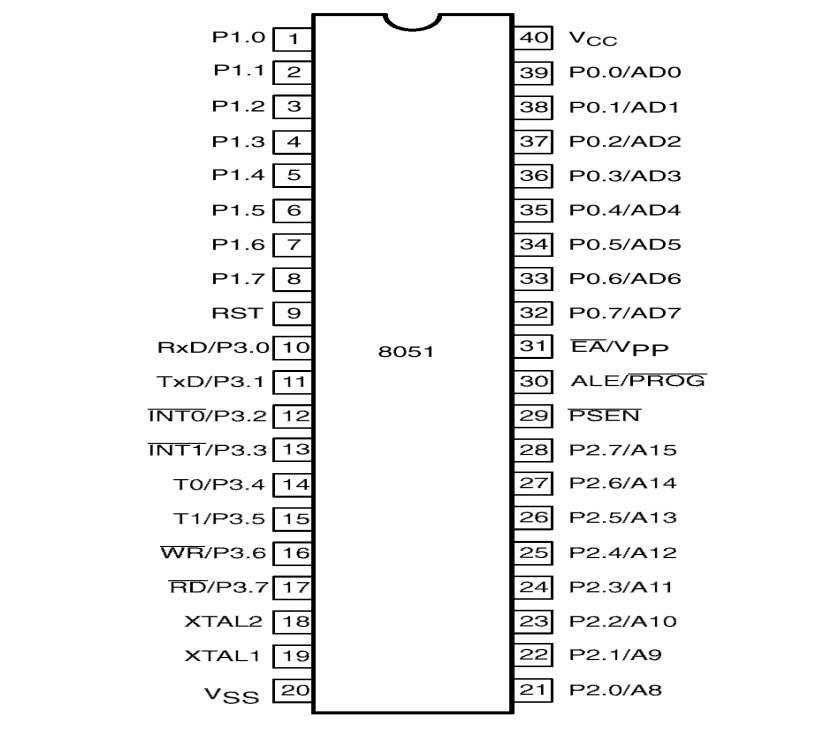


Fig 2.2: 8051 PIN Diagram

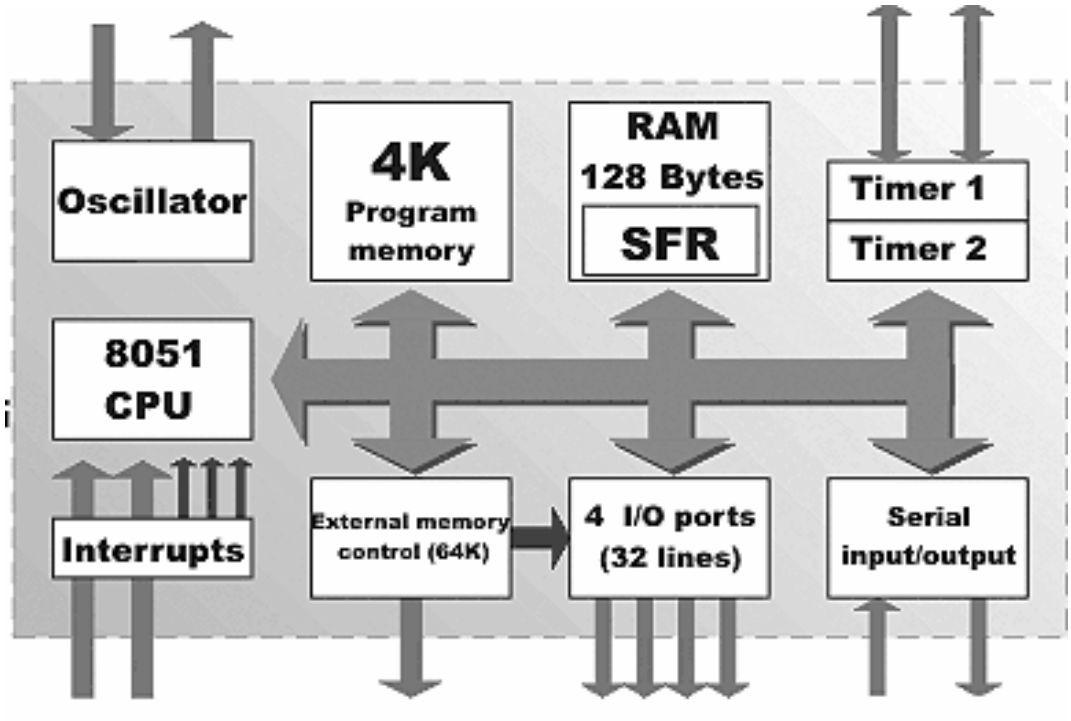


Fig 2.3: 8051 Simplified Diagram

1–8: Port 1: Each of these pins can be used as either input or output according to your needs. Also, pins 1 and 2 (P1.0 and P1.1) have special functions associated with Timer 2.

9: Reset Signal: High logical state on this input halts the MCU and clears all the registers. Bringing this pin back to logical state zero starts the program a new as if the power had just been turned on. In another words, positive voltage impulse on this pin resets the MCU. Depending on the device's purpose and environs, this pin is usually connected to the push-button, reset-upon-start circuit or a brown out reset circuit.

10-17: Port 3:as with Port 1, each of these pins can be used as universal input or output. However, each pin of Port 3 has an alternative function:

- Pin 10: **RXD** - serial input for asynchronous communication or serial output for synchronous communication.
- Pin 11: **TXD** - serial output for asynchronous communication or clock output for synchronous communication
- Pin 12: **INT0*** - input for interrupt 0
- Pin 13: **INT1*** - input for interrupt 1
- Pin 14: **T0** - clock input of counter 0
- Pin 15: **T1** - clock input of counter 1
- Pin 16: **WR*** - signal for writing to external (add-on) RAM memory.
- Pin 17: **RD*** - signal for reading from external RAM memory

18-19: X2 and X1: Input and output of internal oscillator. Quartz crystal controlling the frequency commonly connects to these pins. Capacitances within the oscillator mechanism optimal voltage

20: GND: Ground

21- 28: Port 2: if external memory is not present, pins of Port 2 act as universal input/output. If external memory is present, this is the location of the higher address byte, i.e. addresses A8 – A15. It is important to note that in cases when not all the 8 bits are used for addressing the memory (i.e. memory is smaller than 64kB), the rest of the unused bits are not available as input/output.

29: PSEN*: MCU activates this bit (brings to low state) upon each reading of byte instruction) from program memory. If external ROM is used for storing the program, PSEN- is directly connected to its control pins.

30: ALE: before each reading of the external memory, MCU sends the lower byte of the address register (addresses A0 – A7) to port P0 and activates the output ALE. External Chip (eg: 74HC373), memorizes the state of port P0 upon receiving a signal from ALE pin, and uses it as part of the address for memory chip. During the second part of the MCU cycle, signal on ALE is off, and port P0 is used as *Data Bus*. In this way, by adding only one integrated circuit, data from port can be multiplexed and the port simultaneously used for transferring both addresses and data.

31: EA*: Bringing this pin to the logical state zero designates the ports P2 and P3 for transferring addresses regardless of the presence of the internal memory. This means that even if there is a program loaded in the MCU it will not be executed, but the one from the external ROM will be used instead. Conversely, bringing the pin to the high logical state causes the controller to use both memories, first the internal, and then the external (if present).

32-39: Port 0: Similar to Port 2, pins of Port 0 can be used as universal input/output, if external memory is not used. If external memory is used, P0 behaves as address output (A0 – A7) when ALE pin is at high logical level, or as data output (Data Bus) when ALE pin is at low logical level.

40: VCC: Power +5V

1.2.3 COMPARISON OF 8051 FAMILY MEMBERS

Features	8031	8051	8052
ROM (on-chip program space in bytes)	0K	4K	8K
RAM (Bytes)	128	128	256
Timers	2	2	3
I/O Pins	32	32	32
Serial Port	1	1	1
Interrupt Sources	6	6	8

1.2.4 8051 OSCILLATOR AND CLOCK

The manufacturers make available 8051 designs that can run at specified maximum and minimum frequencies, typically 1 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.

The oscillator formed by the crystal, capacitors and on – chip inverter generates a pulse train at the frequency of the crystal.

The time to execute the instruction is found by using the expression,

$$T(inst) = (C * 12) / (crystal\ frequency)$$

Presently PHILIPS 8051 flash microcontrollers are working with the double speed of normal 8051 chip (40MHz operation) and\ CYGNAL 8051 series working with a speed of 100 MIPS.

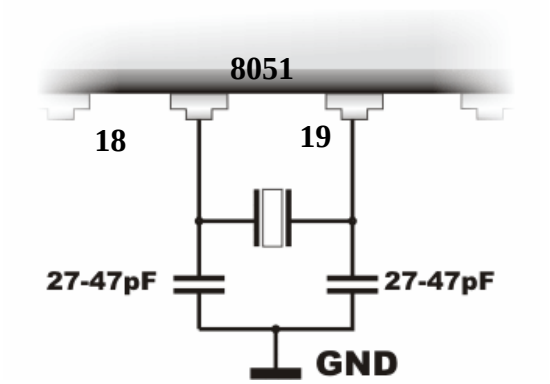


Fig 2.4: 8051 Oscillator Connection Circuit

It must be noted that there are various speed of the 8051 families. Speed refers to the maximum oscillator frequency connected to XTAL. For example, a 12 MHz chip must be connected to a crystal with 12MHz frequency or less. Likewise a 20KHz microcontroller requires a crystal frequency of no more than 20MHz. When the 8051 is connected to a crystal oscillator and is powered up, we can observe the frequency on the XTAL2 pins using the oscilloscope.

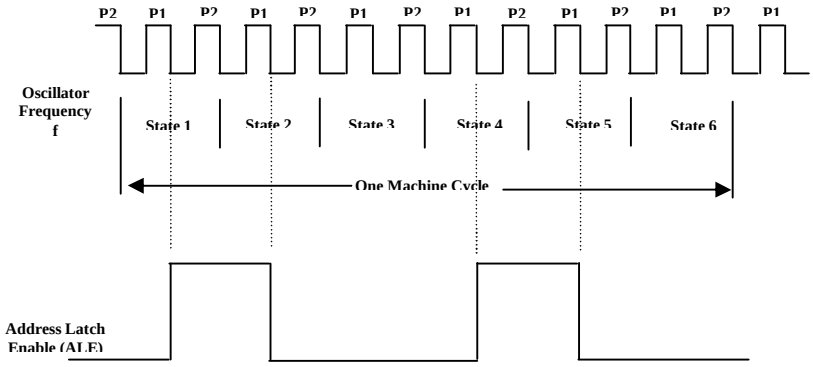


Fig 2.5: 8051 Timing Diagram

1.3 MEMORY ORGANIZATION

During the runtime, microcontroller uses two different types of memory: one for holding the program being executed (ROM memory), and the other for temporary storage of data and auxiliary variables (RAM memory). Depending on the particular model from 8051 family, this is usually few kilobytes of ROM and 128/256 bytes of RAM. This amount is built-in and is sufficient for common tasks performed "independently" by the MCU. However, 8051 can address up to 64KB of external memory.

1.3.1 MEMORY ARCHITECTURE

(i) VON-NEUMAN ARCHITECTURE

Von Neumann architectures are computer architectures that use the same storage device for both instructions and data. By treating the instructions in the same way as the data, the machine could easily change the instructions. In other words the machine was reprogram able. Because the machine did not distinguish between instructions and data, it allowed a program to modify or replicate a program.

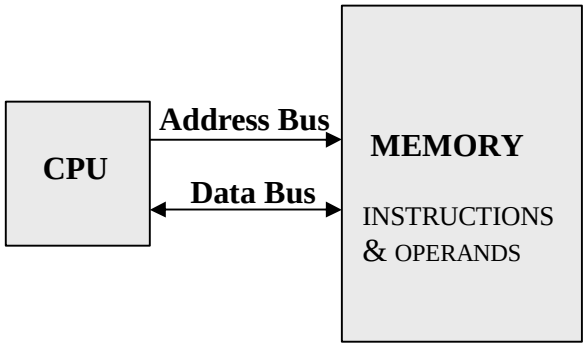


Fig 3.1: Von-Neumann Architecture

(ii) HARVARD ARCHITECTURE

The term **Harvard architecture** originally referred to computer architectures that uses physically separate storage devices for their instructions and data. **Harvard architecture** has separate data and instruction busses, allowing transfers to be performed simultaneously on both busses.

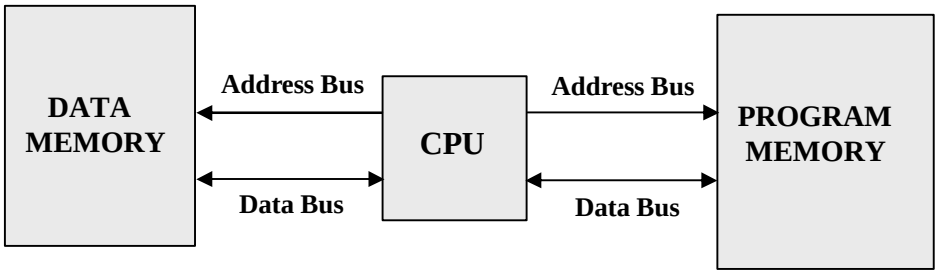


Fig 3.2: Harvard Architecture

The Harvard architecture executes instructions in fewer instruction cycles than the Von Neumann architecture. This is because a much greater amount of instruction parallelism is possible in the Harvard architecture. Parallelism means that fetches for the next instruction can take place during the execution of the current instruction, without having to either wait

for a "dead" cycle of the instruction's execution or stop the processor's operation while the next instruction is being fetched.

1.3.3 ROM MEMORY (PROGRAM MEMORY)

8051 have built-in ROM, although there are substantial variations. With some models internal memory cannot be programmed directly by the user. Instead, the user needs to precede the program to the manufacturer, so that the MCU can be programmed appropriately in the process of fabrication. Obviously, this option is cost-effective only for large series. Fortunately, there are MCU models ideal for experimentation and small specialized series. Many manufacturers deliver controllers that can be programmed directly by the user. These come in an EPROM version or EEPROM version or OTP or FLASH type.

The program memory of 8051 is shown in figure.

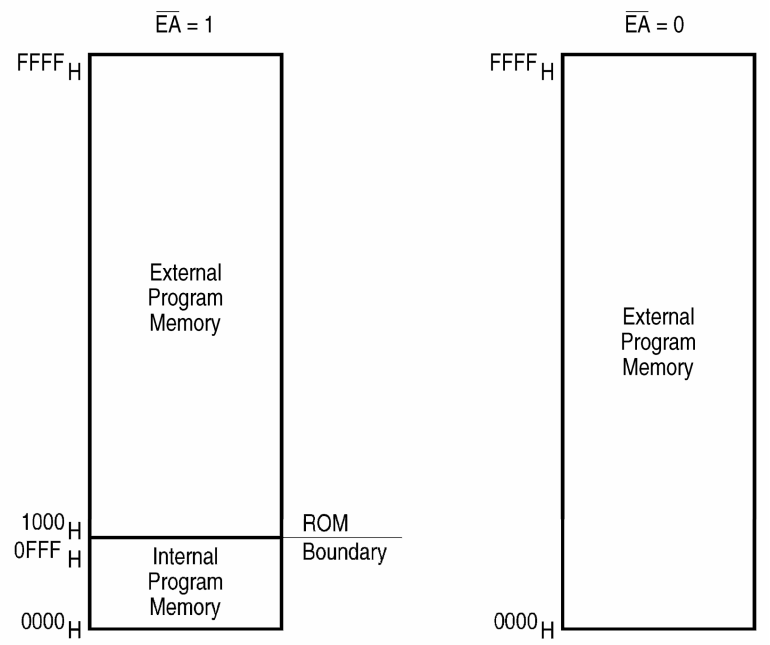


Fig 3.3: 8051 ROM (Program Memory)

1.3.4 RAM MEMORY (DATA MEMORY)

As previously stated, RAM is used for storing temporary data and auxiliary results generated during the runtime. Apart from that, RAM comprises a number of registers: hardware counters and timers, I/O ports, buffer for serial connection, etc. With older versions, RAM spanned 256 locations, while new models feature additional 128 registers. First 256 memory locations form the basis of RAM (addresses 0 – FFh) of every 8051 MCU. Locations that are available to the user span addresses from 0 to 7Fh, i.e. first 128 registers, and this part of RAM is split into several blocks as can be seen in the figure below.

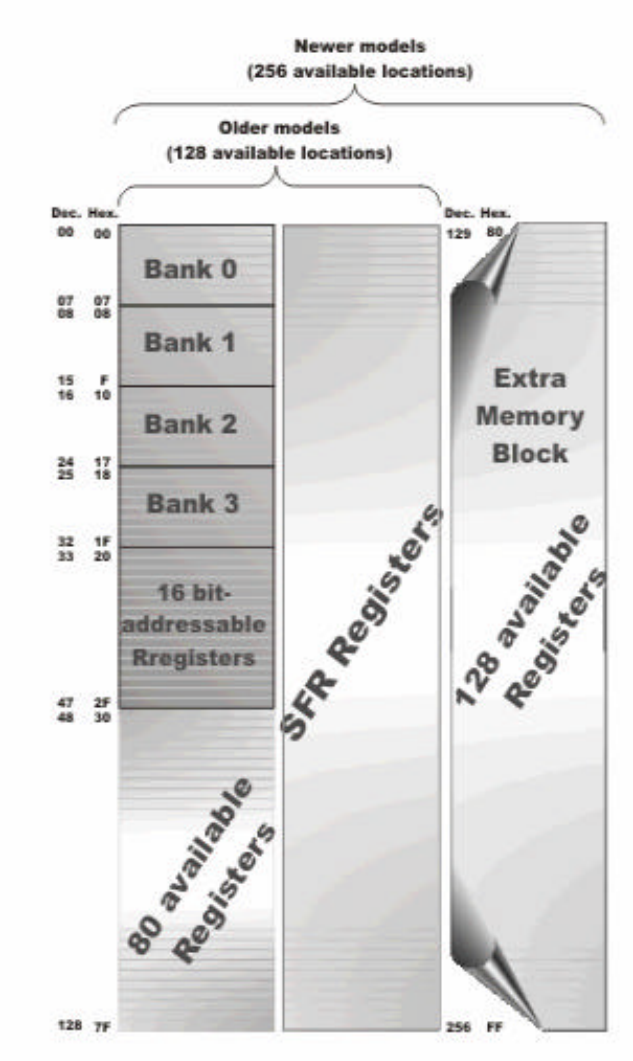


Fig 3.4: RAM ORGANIZATION

First block comprises 4 "banks" of 8 registers each, location 0 through 1Fh(32 bytes. The device after reset default to register bank0. To use other register banks the user must select them in software. Each register bank contains 8 one byte registers, 0 through 7.

Second memory block (range 20h – 2Fh) is bit-addressable, meaning that every belonging bit has its own address (0 to 7Fh). Since the block comprises 16 of these registers, there is a total of 128 addressable bits. (Bit 0 of byte 20h has bit address 0, while bit 7 of byte 2Fh has bit address 7Fh). Each of the 16 bytes in this segment can also be addressed as a byte.

Third is the group of available registers at addresses 30h – 7Fh (total of 80 locations) without special features or a preset purpose.

To satisfy the programmers' ever-increasing demands for RAM, latest 8051 models were added an extra memory block of 128 locations. The problem lies in the fact that the electronics, which addresses RAM, employs 1 byte (8 bits), reaching only the first 256 locations. Therefore, a little trick had to be applied in order to keep the existing 8-bit architecture for the sake of compatibility with older models. The idea is to make the additional memory block share the addresses with the existent locations intended for SFR registers (80h - FFh). For distinguishing these two physically separate memory areas, different methods of addressing are used: if SFR registers are in question, direct addressing is used; for extra RAM locations, indirect addressing is used.

The 8051 can addressed up to 64K bytes of Data Memory external to the chip. The MOVX instruction is used to access the external memory. Figure shows the 8051 data memory organization.

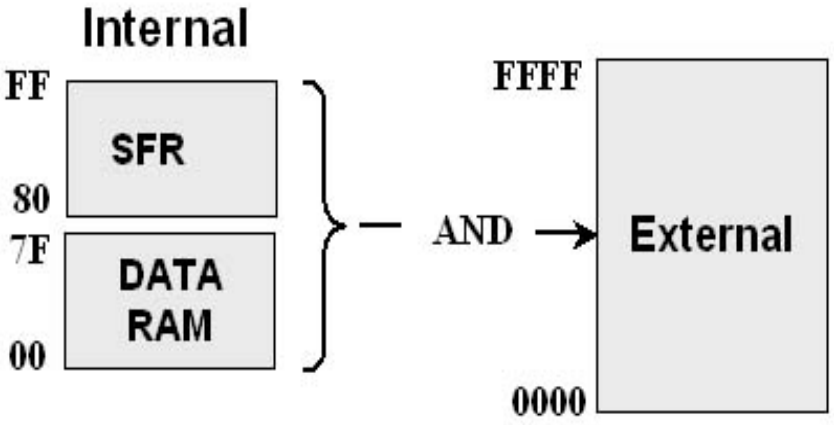


Fig 3.5: 8051 RAM(Data Memory)

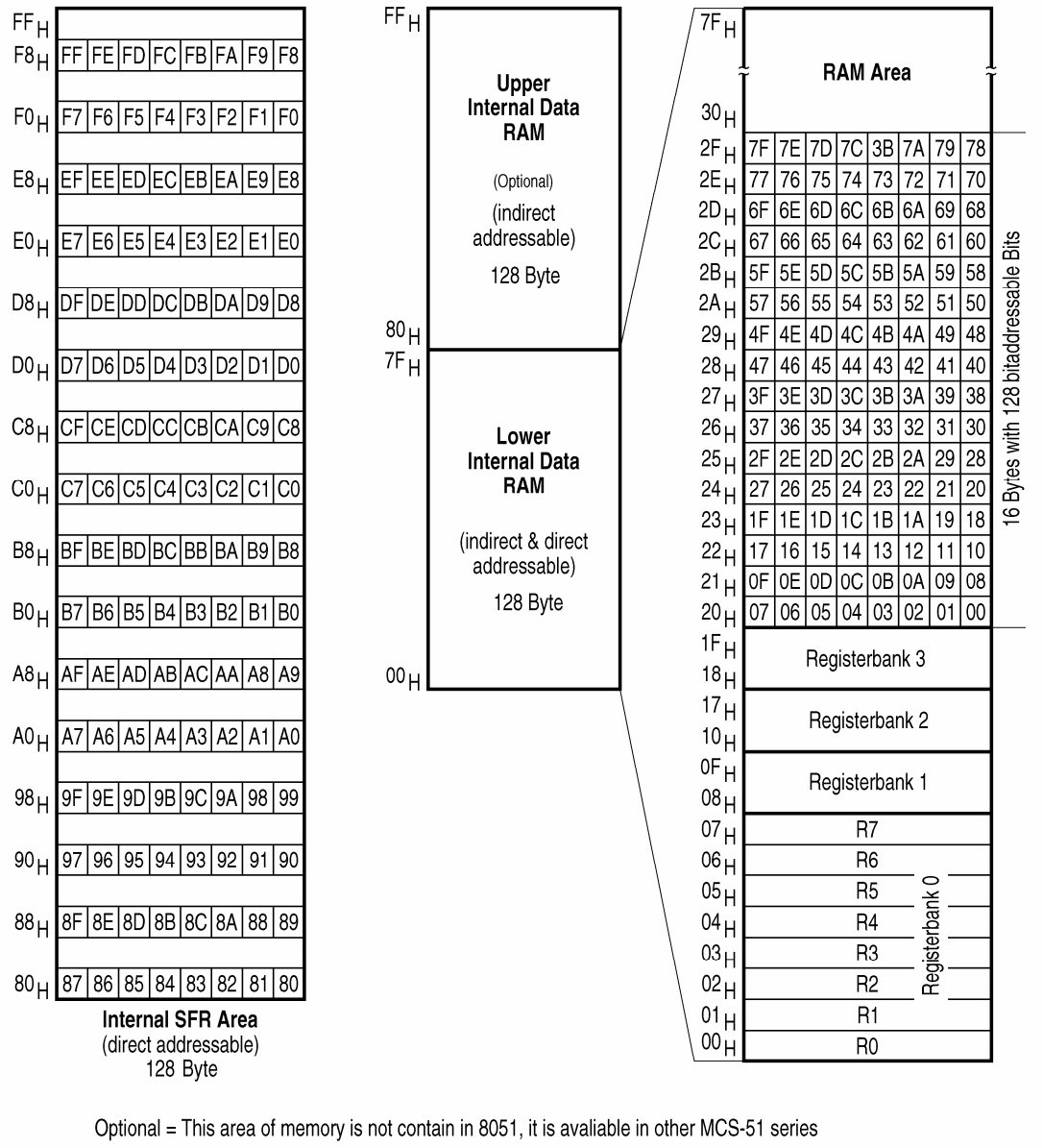


Fig 3.6: Internal RAM Architecture (MCS-51)

1.3.5 MEMORY CAPACITY

The number of bits that a semiconductor memory chip can store is called the chip capacity. The memory capacity of a memory IC chip is always given in bits, the memory capacity of a computer system is given in bytes.

SPEED

One of the most important characteristics of a memory chip is the speed at which its data can be accessed. To access the data, the address is presented to the address pins, the READ pin is activated, and after a certain amount of time has elapsed, the data shows up at the data pins. The speed of the memory chip is commonly referred to as its access time. The table serves as a reference for memory calculations.

X	**2^x**
10	1K
11	2K
12	4K
13	8K
14	16K
15	32K
16	64K
17	128K
18	256K
19	512K
20	1M
21	2M
22	4M
23	8M
24	16M

1.3.6 MEMORY TYPES

i. ROM

ROM is a type of memory that does not lose its contents when the power is turned off. For this reason, ROM is also called nonvolatile memory. There are different types of ROM, such as PROM, EPROM, EEPROM, OTP ROM, and FLASH EEPROM.

ii. EPROM

In EPROM, one can program the memory chip and erase it thousands of times. This is especially necessary during the development of the prototype of a embedded systems based design. A widely used EPROM is called UV-EPROM where UV stands for ultra-violet. All UV-EPROM chips have a window that is used to shine UV radiation to erase its contents. The main problem, and indeed the major disadvantage of UV-EPROM is that it cannot be programmed while in the system board. To find a solution, EEPROM was invented.

iii. FLASH MEMORY

Flash EPROM has become a popular user programmable memory chip since the early 90s. In this the process of erasure of the entire contents takes less than a second, or one might say in a flash, hence its name flash EEPROM. Flash memories increases the performance of the computer, since flash memory is semiconductor memory with access time in the range of 100 ns compared with disk access time in the range of tens of milliseconds.

iv. RAM

RAM memory is called volatile memory since cutting off the power to the IC will result in the loss of data. The three types of RAM are: static RAM, NV-RAM (non-volatile RAM), and dynamic RAM. Storage cells in static RAM memory are made of flip-flops and therefore do not require refreshing in order to keep their data. The problem with the use of flip-flops for storage cells is that each cell requires at least 6 transistors to build, and the cell holds only 1 bit of data. NV-RAM combines the best of RAM and ROM: the read and write ability of RAM, plus the nonvolatile of ROM. DRAM uses capacitors as storage cells. The major advantages of DRAM are high density (capacity), cheaper cost per bit, and lower power consumption per bit. The disadvantage is that it must be refreshed periodically, due to the fact that capacitor cell loses its charge.

1.3.7 MEMORY EXPANSION

In case the built-in amount of memory (either RAM or ROM) is not sufficient for your needs, there is always an option of adding two external 64KB memory chips. When added, they are addressed and accessed via I/O ports P0 and P2.

8051 MCU has two separate read signals, RD and $\overline{\text{PSEN}}$. The first one is active when reading byte from the external data memory (RAM), and the second one is active when reading byte from the external program memory (ROM). Both signals are active on low logical level. The following image shows a typical scheme for such expansion using separate chips for RAM and ROM, known as *Harvard architecture*.

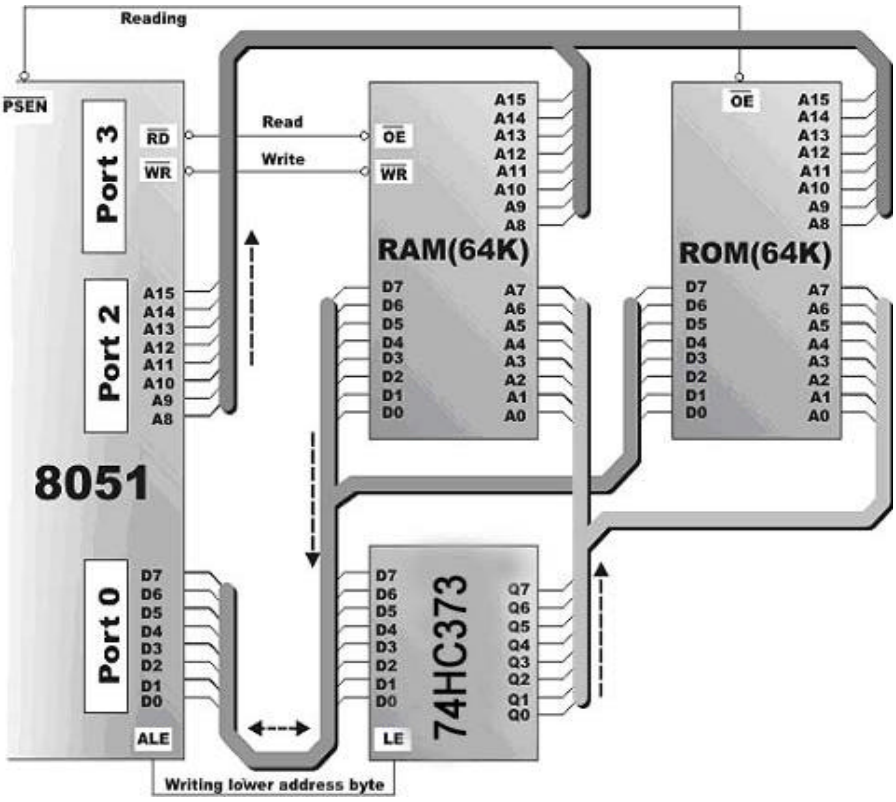


Fig 3.7: Interfacing External Memory with 8051

1.4 8051 REGISTERS

1.4.1 SFR (SPECIAL FUNCTION REGISTERS)

SFR can be seen as a sort of control panel for managing and monitoring the microcontroller. Every register and each of the belonging bits has its name, specified address in RAM and strictly defined role (e.g. controlling the timer, interrupt, serial connection, etc). Although there are 128 available memory slots for allocating SFR registers.. The rest has been left open intentionally to allow future upgrades while retaining the compatibility with earlier models. This fact makes possible to use programs developed for obsolete models long ago.

TABLE OF SPECIAL FUNCTION REGISTERS

NAME	FUNCTION	HEX ADDRES	Bit-Addressable
A	Accumulator	E0	Yes
B	Arithmetic	F0	Yes
DPTR	Data Pointer (2 Bytes)	---	---
DPH	Addressing external memory	83	No
DPL	Addressing external memory	82	No
IE	Interrupt enable control	A8	Yes
IP	Interrupt priority	B8	Yes
P0	Input/output port latch	80	Yes
P1	Input/output port latch	90	Yes
P2	Input/output port latch	A0	Yes
P3	Input/output port latch	B0	Yes
PCON	Power control	87	No
PSW	Program status	D0	Yes
SCON	Serial port control	98	Yes
SBUF	Serial port data buffer	99	No
SP	Stack pointer	81	No
TMOD	Timer/counter mode control	89	Yes
TCON	Timer/counter control	88	Yes
TL0	Timer 0 low byte	8A	No
TH0	Timer 0 high byte	8B	No
TL1	Timer 1 low byte	8C	No
TH1	Timer 1 high byte	8D	No

1.4.2 ACCUMULATOR

Accumulator is a general-purpose register, which stores runtime results. Before performing any operation upon an operand, operand has to be stored in the accumulator. Results of arithmetical operations (performed by ALU) are also stored in the accumulator. When transferring data from one register to another, it has to go through the accumulator.

Due to its versatile role, this is the most frequently used register, essential part of every MCU.

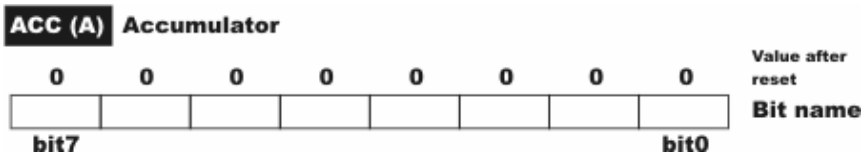


Fig 4.1 : Accumulator

1.4.3 B REGISTER

Used along with the Accumulator for multiplication/division. This B register provides temporary storage space for the 16 bit results in multiplication & division operation. Instructions of multiplication and division can be applied only to operands located in registers A and B. Other instructions can use this register as a secondary accum80ulator (A).

1.4.4 P0, P1, P2, P3 - I/O PORTS

8051 has 4 ports, with each ports have 8 bit length. All the ports are bit and byte addressable. Every port bit corresponds to one of the pins on the casing, thus controlling the voltage on output (0 or 5V). Vice versa, while reading, voltage on input pins is interpreted into bit logic on port. A 5V at the input makes the port bit into logic 1(HIGH) and 0V into logic 0 (LOW).

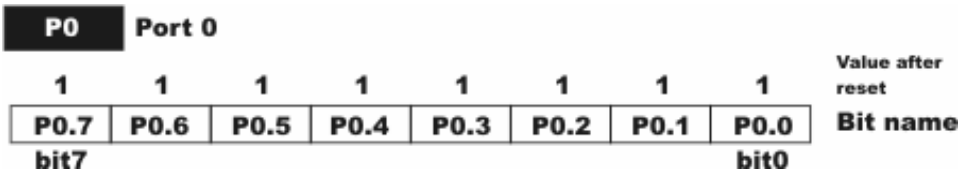


Fig 4.2: I/O PORT (Port 0)

1.4.5 DATA POINTER

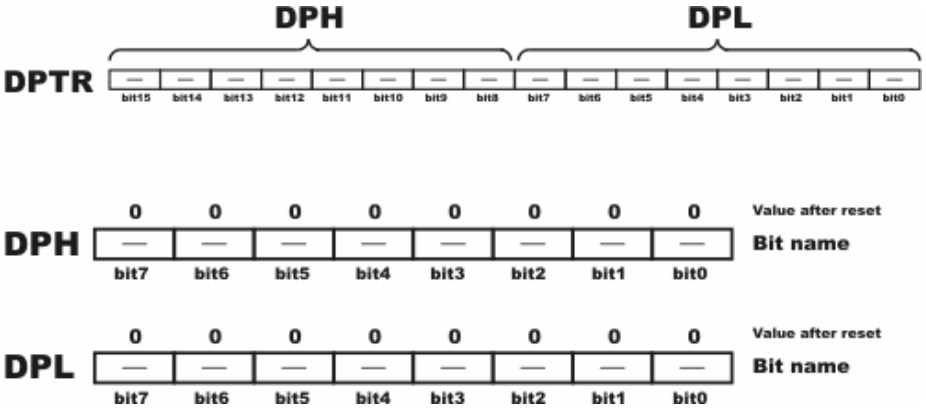


Fig 4.3: DATA POINTER

The Data pointer register is made up of two 8 bit registers, named DPH (Data Pointer High) and DPL (Data Pointer Low). These registers are used to give addresses of the internal or external memory.The DPTR is under the control of program. DPTR is also manipulated as one 16 bit register, DPH & DPL are each assigned an address. Data Pointer's 16 bits are used for addressing external memory. Since this is the only 16-bit register available to programmer, it is commonly used for temporary storage of data and runtime results.

1.4.6 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 register used to access the stack is called Stack Pointer. The 8 bit stack pointer register is used by the 8051 to hold an internal RAM address that is called then *top of the stack.*

Upon reset (or turning the power on), the stack pointer contains the value 07h.This means that RAM location 08h is the first location being used for the stack by the 8051. If another value is written to SP, entire Stack moves to the new specified location.

The stack pointer increments before storing the data on the stack As retrieved from the stack the SP is decremented by one.The storing of CPU register in the stack is called a PUSH, and loading the contents of the stack back into a CPU register is called a POP.The number in Stack Pointer points to the location of the last "valid" address within the Stack. With the beginning of every new routine, Stack Pointer increases by 1; upon return from routine, SP decreases by 1.

Fig 4.4: STACK POINTER

For example:

```
MOV  SP,#22H ; Copy the immediate data 22H to SP
MOV  R1,#33H ; Copy the immediate data 33H to R1
PUSH 00H     ; SP = 23H, Address 23H contains the number 33H
PUSH 00H     ; SP = 24H, Address 24H contains the number 33H
POP  80H     ; SP = 23H, Port 0 latch now contains the number 33H
```

1.4.7 PROGRAM COUNTER (PC)

Used to access code memory. Program counter always points to the address of the next instruction in memory to be executed. Some of the instructions in the 8051 are 1 byte while the others are two bytes or three bytes.

```
        MOV     A,#01H          PC = ?
        MOV     DPTR,#4500H
        MOVX    @DPTR,A         PC = ?
STOP:   SJMP    STOP
```

1.4.8 PROGRAM STATUS WORD

The program status word (PSW) register is an 8 bit register. It is also referred to as the flag register. It contains the math flags, user program flag F0,and the register select bits that identify which of the four general purpose register banks is currently in use by the program. PSW is one of the most important SFR registers, and is used for managing program during the runtime. ALU automatically makes changes to certain bits of this register.

- **P (bit 0)** - *Parity bit.* If numeral in accumulator is even, bit is automatically set (1), otherwise it's cleared (0). It is commonly used in data transfers via serial connection.
- **(bit 1)** - This bit is intended for the upcoming MCU models and shouldn't be used.
- **OV (bit 2)** - *Overflow bit.* If result of arithmetical operation exceeds 255 (decimal), OV is set (1), otherwise it's cleared (0).
- **RS1, RS0 (bits 3 and 4)** - Register select. Masking these bits stores registers R0 - R7 into one of the 4 banks in RAM, according to the following table.

RS1	RS2	Location in RAM
0	0	Bank 0 00h-07h
0	1	Bank 1 08h-0Fh
1	0	Bank 2 10h-17h
1	1	Bank 3 18h-1Fh

- **F0 (bit 5)** - *Flag 0.* An all-purpose user flag can be used by the user.
- **AC (bit 6)** - *Auxiliary Carry Flag*, used only for operations with BCD (Binary Coded Decimal).
- **CY (bit 7)** - *Carry Flag.* Auxiliary (ninth) bit for arithmetical and shift operations.

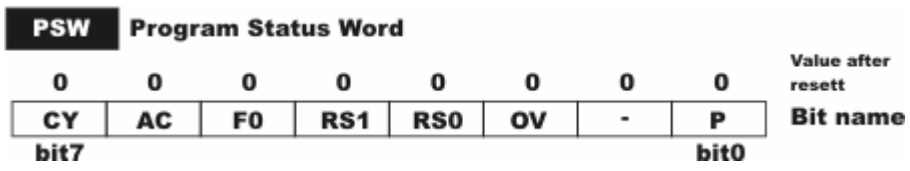


Fig 4.5: Flag Register

1.4.9 COUNTERS & TIMERS:

The 8051 MCU clock employs a quartz crystal. As this frequency is highly stable and accurate, it is ideal for time measuring. This is where the timer takes part; properly programmed, value of timer register will increase or decrease with every MCU clock impulse. Since one instruction takes 12 oscillator cycles to complete, the math is easy.

8051 has two timers/counters marked as T0, T1. Their purpose is to measure time and count external occurrences, but can also be used as clock in serial communication purpose called as, *Baud Rate*.

1.4.10 Timer T0

As shown in the image below, T0 consists of two registers - TH0 and TL0, for storing higher and lower byte of a 16-bit binary numeral.

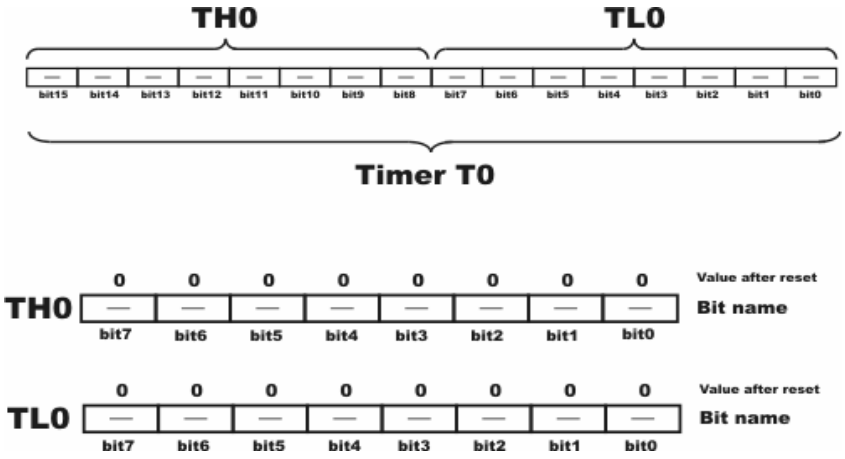
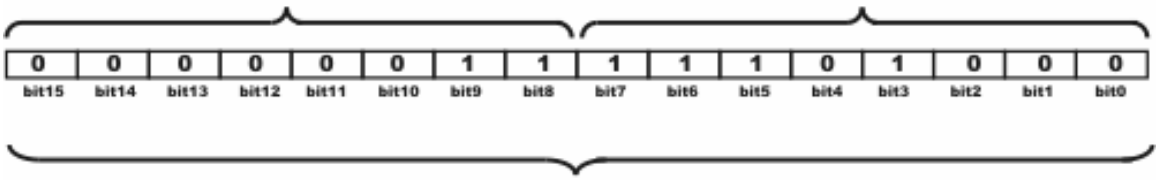


Fig 4.6: Timer 0 Register

For example, if T0 = 0, both registers will have value of zero. If T0 has value of 1000d (3E8H), TH0 (higher byte) will hold decimal value of 3 (03H), and TL0 (lower byte) will hold decimal value of 232 (E8H). See the image below.

TH0 = 03H (03D) **TL0 = E8H (232D)**



TIMER T0 = 3E8H (1000)

Fig 4.7: Example of Timer 0

Formula for calculating the value of 16-bit register is simple:

TH0 * 256 + TL0 = T

On our previous example:

3 * 256 + 232 = 1000

Timers are technically 16-bit registers, thus the maximal value they can hold is 65,535 (FFFFH). If this number is exceeded, timer will automatically reset and start from zero. This situation is known as *overflow*.

Two registers tightly connected to Timer T0 are TMOD and TCON.

1.4.11 Timer T1

This is the "twin brother" of Timer T0. It can fulfill same roles, it is also controlled by TMOD and TCON, and has 4 different modes of work.

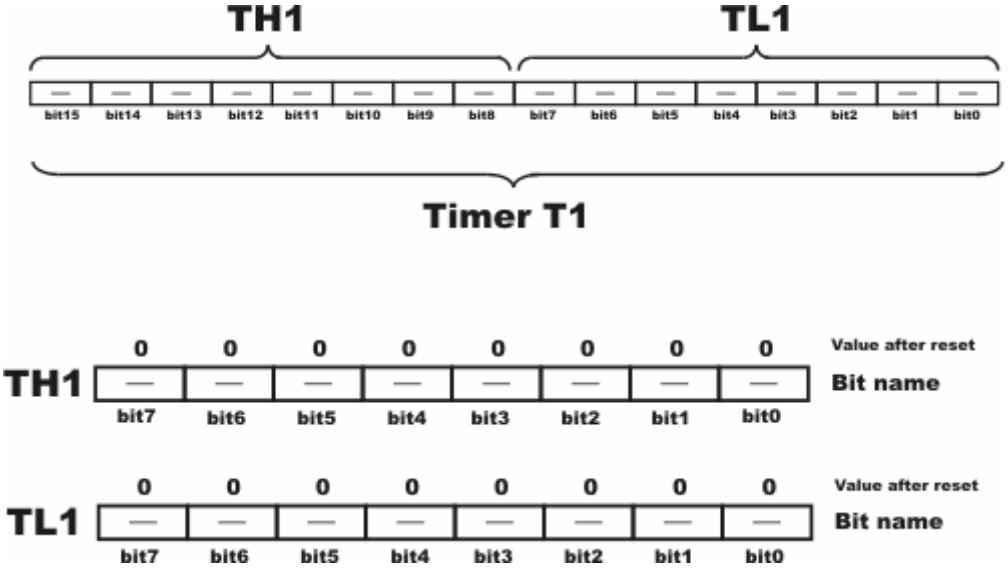


Fig 4.8: Timer 1 Register

1.4.12 TMOD - Timer Mode

TMOD is dedicated to the two timers T0 & T1 and can be considered to be two duplicate 4-bit registers, each of which controls the action of one of the timers. TCON has control bits and flags for the timers in the upper nibbles, and control bits and flags for the external interrupts in the lower nibble.

This register sets mode for timers T0 and T1. As shown in the image below, lower 4 bits (bit 0 - bit 3) are associated with T0, while the higher 4 bits (bit4 - bit7) are associated with T1.

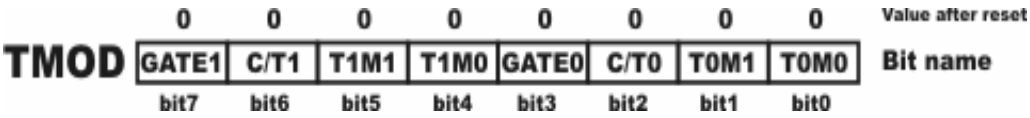


Fig 4.9: TMOD Register

The following table gives details on bits 0 – 7:

Bit	Bit Name	Purpose	Timer
7	GATE1	1 Timer works only if INT1 (P3.3) is set 0 Timer works regardless of INT1 (P3.3)	T1
6	C/T1-	1 Timer counts impulses on T1 (P3.5) 0 Timer counts impulses of internal oscillator	T1
5	T1M1	Timer mode	T1
4	T1M0	Timer mode	T1
3	GATE0	1 Timer works only if INT0 (P3.2) is set 0 Timer works regardless of INT0 (P3.2)	T0
2	C/T0-	1 Timer counts impulses on T0 (P3.4) 0 Timer counts impulses of internal oscillator	T0
1	T0M1	Timer mode	T0
0	T0M0	Timer mode	T0

Four bits from the previous table determine the operating mode of timers T0 and T1. There are 4 of these modes, and each will be covered in details.

T0M1	T0M0	Mode	Description
0	0	0	13-bit Timer
0	1	1	16-bit Timer
1	0	2	8-bit *auto-reload*
1	1	3	*Split* mode

i. Mode 0 (13-bit Timer)

This mode is an antiquity kept just for the sake of compatibility with older MCUs. When activated, whole higher byte TH0 and only the first 5 bits of lower byte TL0 are accessible. Thus, with Mode 0, Timer T0 uses only 13 of its 16 bits. How does it work? On each impulse, lower register is changed (the "trimmed" one). When TL0 is filled after 32 impulses, it is automatically reset, and TH0 is increased by one. This process repeats itself until 8192 (2^13 bits) impulses are registered, upon which both registers are reset to zero.

ii. Mode 1 (16-bit "manual reload" Timer)

Mode 1 uses all bits of registers TH0 and TL0, and is commonly used. Counting process is same as with Mode 0, except the timer reaches value of 65,536 (max for 16 bits) before reset.

iii. Mode 2 (8-bit "auto reload" Timer)

In "**a*uto reload*"** mode, only one of two registers is used for counting; however, it does not start from zero, but from a specified value stored in the other register (0-255).Advantages of this mode will be illustrated on the following example: suppose that there is a need to report every 55th impulse of the clock. If Mode 0 or Mode 1 were used, you would need to store 200 (decimal) into T0, and then continually check for the overflow (exceeding 255 decimal). Upon hit, value of 200 would need to be written to T0 again. In Mode 2, MCU performs this task automatically. Namely, TL0 works as an 8-bit timer, while TH0 stores the starting value, specifically 200 in our example. When TL0 is filled, instead of reset, it will load value from TH0. Thus, to register every 55th impulse, all you need to do is write 200 to TH0, and set the Timer Mode 2.

iv. Mode 3 ("Split" Timer)

When Timer T0 is configured to Mode 3, you actually get an additional timer. In this mode, registers TH0 and TL0 act as separate 8-bit timers: TH0 substitutes Timer 0, while TL0 substitutes Timer 1. Consequently, all control bits associated with the original Timer 1 (16-bit register consisting of TH1 and TL1) are now in control of newly created "Timer 1". This means that, although it can be set to any mode (Mode 1, 2, or 3), the original Timer 1 cannot be stopped anymore, because there is simply no control bit to do it. In this mode, it will be constantly active in the background.

1.4.13 TCON - Timer Control

TCON is another register in direct control of the timers.

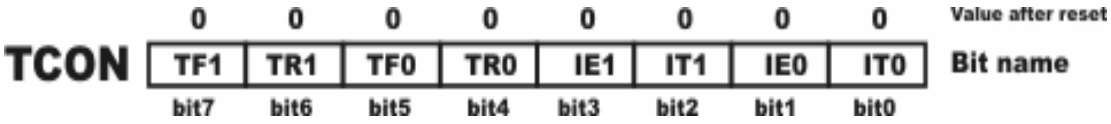


Fig 4.10: TCON Register

Of the 8 bits, TCON uses only 4 bits for controlling the timers, while the other 4 are associated with interrupts.

Bit	Bit Name	Purpose	Timer
7	TF1	This bit is automatically set in case of overflow in Timer T1	T1
6	TR1	**1** - Timer T1 is on **0** - Timer T1 is off	T1
5	TF0	This bit is automatically set in case of overflow in Timer T0	T0
4	TR0	**1** - Timer T0 is on **0** - Timer T0 is off	T0

1.4.14 UART (Universal Asynchronous Communication)

One of the things that makes this MCU so powerful is the hardware integrated UART, better known as serial port. It is a duplex port capable of sending and receiving data simultaneously. UART represents an elegant solution: programmer just needs to set the mode and the rate of transfer. Register SBUF holds data to be sent to line, and the same register accepts data from the line. Controller takes care of all the details of transfer with no room for error.

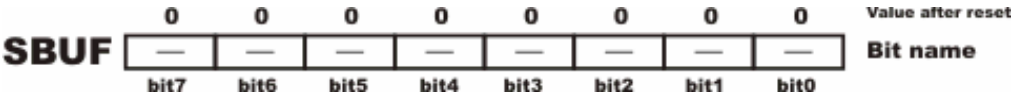


Fig 4.11: Serial Buffer

Before using the serial port, it should be appropriately configured. SFR register SCON (*Serial Control*) is in control of the transfer parameters: size of one serial "word" in bits, baud rate, and the source of impulses for synchronization.

1.4.15 CONNECTION TO RS232

The 8051 has two pins that are used specifically for transferring and receiving data serially. These two pins are called TxD and RxD and are art of the port 3 groups. Pin 11 of the 8051 (P3.1) is assigned to TxD and pin 10 (P3.0) is designated as RxD. These pins are TTL compatible; therefore, they require a line driver to make them RS232 compatible. One such line driver is the MAX232 chip. MAX232 converts from RS232 voltage levels to TTL voltage levels, and vice versa. One advantage of this chip is that it uses a +5 V power source.

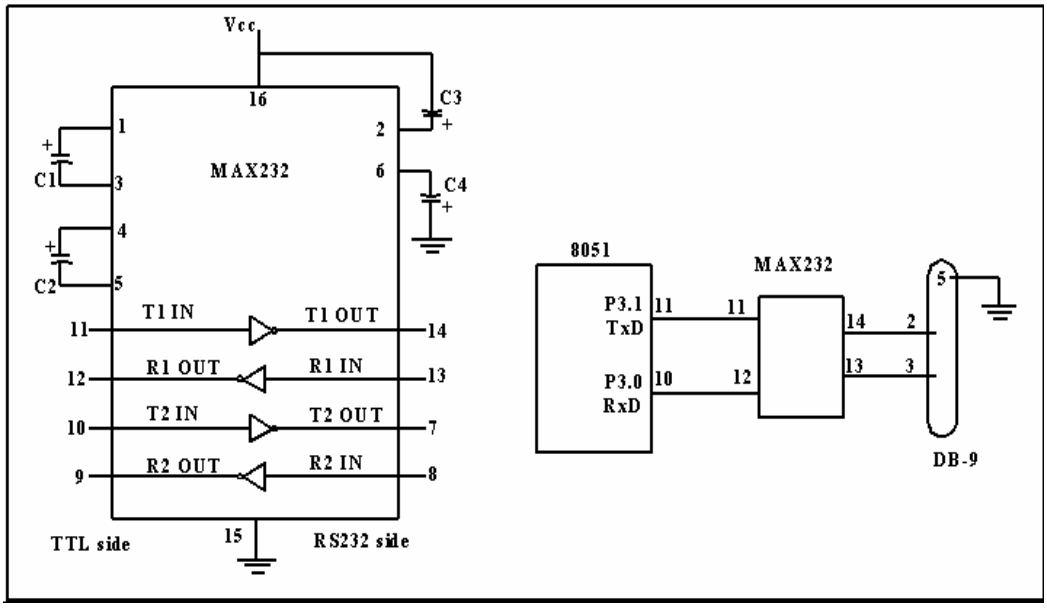


Fig 4.12: 8051 Interfacing with MAX232

1.4.16 SCON (Serial Port Control Register)

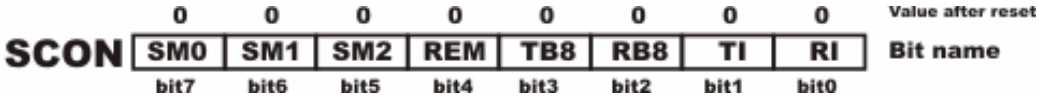


Fig 4.13: SCON Register

Bit Name	Bit Address	Purpose
SM0	9Fh	Mode of work
SM1	9Eh	Mode of work
SM2	9Dh	Enables connecting multiple MCU's
REM	9Ch	When set, enables receiving data
TB8	9Bh	9th bit for sending data in modes 2 and 3
RB8	9Ah	9th bit for sending data in modes 2 and 3
TI	99h	Bit is automatically set when the whole byte is sent
RI	98h	Bit is automatically set when the whole byte is received

SM0	SM1	Mode	Description	Baud Rate
0	0	0	8-bit Shift register	Quartz frequency / 12
0	1	1	8-bit UART	Determined by timer T1 or T2
1	0	2	9-bit UART	Quartz frequency / 32
1	1	3	9-bit UART	Determined by timer T1 or T2

As shown in the table, combination of bits SM0 and SM1 determines the mode of work for serial port. Obviously, baud rate in modes 0 and 2 is fixed, and can be adjusted in modes 1 and 3. In addition, baud rate in modes 1, 2, and 3 is doubled if bit SMOD in register PCON is set.

i. Mode 0

Mode 0 is not used for standard serial communication, but serves to provide additional I/O pins. External shift registers convert the data into binary sequence, which is then serially transferred to the controller. Although there is no limit on the number of I/O ports that can be gained in this manner, the existing 32 I/O lines are sufficient for most of the tasks, making this mode rarely used.

ii. Mode 1

This is the standard RS-232 mode for serial transfer of 8-bit data. Sequence of ten bits is sent via pin TXD or received via pin RXD in the following order: one start bit (always 0), followed by 8 data bits (LSB bit is first), and one stop bit (always 1). Start bit is not registered anywhere as its sole purpose is to start the mechanism for receiving data. When

data is received, stop bit is automatically copied to bit RB8 in register SCON. In order to connect the controller to RS-232 line, this "raw" serial data needs to be inverted - the designated drivers carry this out automatically.

iii. Modes 2 and 3

These modes are frequently used for speedy transfers at short range (Mode 2) and for standard RS-232 transfers with parity bit (Mode 3). Both modes transfer 9-bit data in the following order: one start bit (always 0), followed by 8 data bits (LSB bit is first), 9th bit which is copied from the bit TB8 before transfer, and to the bit RB8 after transfer, and finally one stop bit (always 1).

Other bits of register SCON have following roles:

Bit	Purpose
SM2	This bit is used if multiple microcontrollers exchange data using the same line. Otherwise, it needs to be cleared to provide normal functioning of the communication
REN	Needs to be set to enable receiving data via serial communication
TB8	Auxiliary 9th bit in 9-bit transfer (modes 2 and 3)
RB8	Similar to TB8, but on receiving. When accepting 9-bit data, it stores value of the ninth bit.
TI	This bit is automatically set when the last bit of one byte has been proceeded to the line. In this way, processor "knows" that the line is free for sending another byte.
RI	Similar to TI, but on receiving. It is a "doorbell" of a kind, which indicates that one byte has been received, and that it should be read before another one arrives.

1.4.17 BAUD RATE

To allow data transfer between the PC and an 8051 system without any error, we must make sure that the baud rate of the 8051 system matches the baud rate of the PC's COM port. The 8051 transfers and receives data serially at many different baud rates. The baud rate in the 8051 is programmable. The 8051 divide the crystal frequency by 12 to get the machine cycle frequency.

In the case of XTAL = 11.0592 MHz, the machine cycle frequency is 921.6 KHz (11.0592 MHz / 12 = 921.6 KHz). The 8051's serial communication UART circuitry divides the machine cycle frequency of 921.6 KHz by 32 once more before it is used by timer 1 to set the baud rate. When timer 1 is used to set the baud rate it must be programmed in mode 2.

Baud Rate in modes 0 and 2 depends solely on the frequency of quartz crystal. Timers T1 and/or T2 determine baud Rate in modes 1 and 3. Timer T1 is most commonly used in "Auto-Reload" mode (TMOD = 0010xxxx). In this case, rate is determined by the frequency of overflow occurrence, and can be calculated according to the formula:

$$\text{Baud Rate} = \frac{\text{oscillator frequency}}{\text{384 * (256 - TH1)}}$$

If bit SMOD in register PCON is set, rate will be doubled:

$$\text{Baud Rate} = \frac{\text{oscillator frequency}}{\text{192 * (256 - TH1)}}$$

To determine the value that must be placed in TH1 to generate a given baud rate, we may use the following equation (assuming PCON.7 is clear).

TH1 = 256 - ((Crystal / 384) / Baud)

If PCON.7 is set then the baud rate is effectively doubled, thus the equation becomes:

TH1 = 256 - ((Crystal / 192) / Baud)

For example, if we have an 11.0592Mhz crystal and we want to configure the serial port to 19,200 baud we try plugging it in the first equation:

TH1 = 256 - ((Crystal / 384) / Baud)
TH1 = 256 - ((11059000 / 384) / 19200)
TH1 = 256 - ((28,799) / 19200)
TH1 = 256 - 1.5 = 254.5

As you can see, to obtain 19,200 baud on a 11.059Mhz crystal we'd have to set TH1 to 254.5. If we set it to 254 we will have achieved 14,400 baud and if we set it to 255 we will have achieved 28,800 baud. But to achieve 19,200 baud we simply need to set PCON.7 (SMOD). When we do this we double the baud rate and utilize the second equation mentioned above. Thus we have: 0

TH1 = 256 - ((Crystal / 192) / Baud)
TH1 = 256 - ((11059000 / 192) / 19200)
TH1 = 256 - ((57699) / 19200)
TH1 = 256 - 3 = 253

Here we are able to calculate a perfect TH1 value. Therefore, to obtain 19,200 baud with an 11.059MHz crystal we must:

1. Configure Serial Port mode 1 or 3.
2. Configure Timer 1 to timer mode 2 (8-bit auto-reload).
3. Set TH1 to 253 to reflect the correct frequency for 19,200 baud.
4. Set PCON.7 (SMOD) to double the baud rate.

To get baud rates compatible with the PC, we must load TH1 with the values shown in table below:

Baud Rate	TH1 (Decimal)	TH1 (Hex)
9600	-3	FD
4800	-6	FA
2400	-12	F4
1200	-24	E8

i. Writing to the Serial Port

Once the Serial Port has been properly configured as explained above, the serial port is ready to be used to send data and receive data. To write a byte to the serial port one must simply write the value to the SBUF (99H) SFR. For example, if you wanted to send the letter "A" to the serial port, it could be accomplished as easily as:

```
MOV SBUF, #'A'
```

Upon execution of the above instruction the 8051 will begin transmitting the character via the serial port. Obviously transmission is not instantaneous--it takes a measurable amount of time to transmit. And since the 8051 does not have a serial output buffer we need to be sure that a character is completely transmitted before we try to transmit the next character.

The 8051 lets us know when it is done transmitting a character by setting the TI bit in SCON. When this bit is set we know that the last character has been transmitted and that we may send the next character, if any. Consider the following code segment:

```
      CLR  TI          ;Be sure the bit is initially clear
      MOV  SBUF, #'A'  ;Send the letter 'A' to the serial port
LOOP:JNB  TI, LOOP     ;Pause until the TI bit is set.
```

The above three instructions will successfully transmit a character and wait for the TI bit to be set before continuing. The last instruction says "Jump if the TI bit is not set to LOOP, means "the same address of the current instruction." Thus the 8051 will pause on the JNB instruction until the TI bit is set by the 8051 upon successful transmission of the character.

ii. Reading From the Serial Port

To read a byte from the serial port one just needs to read the value stored in the SBUF (99h) SFR after the 8051 has automatically set the RI flag in SCON.

For example, if your program wants to wait for a character to be received and subsequently read it into the Accumulator, the following code segment may be used:

```
LOOP:  JNB   RI,LOOP  ;Wait for the 8051 to set the RI flag
       MOV   A,SBUF   ;Read the character from the serial port
```

The first line of the above code segment waits for the 8051 to set the RI flag; again, the 8051 sets the RI flag automatically when it receives a character via the serial port. So as long as the bit is not set the program repeats the "JNB" instruction continuously.

Once the RI bit is set upon character reception the above condition automatically fails and program flow falls through to the "MOV" instruction which reads the value.

1.4.18 INTERRUPTS

An interrupt is an external or internal event that interrupts the microcontroller to inform it that a device needs its service. A single microcontroller can serve several devices. There are two ways to do that: interrupts or polling. In the interrupt method, whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal. Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device. The program, which is associated with the interrupt, is called the interrupt service routine. Although polling can monitor the status of several devices and serve each of them as certain conditions are met, it is not efficient as far as microcontroller is concerned. The advantage of interrupt is that each device get the attention of the microcontroller based on the priority assigned to it. In interrupt method the time taken by the microcontroller is less when compared to polling.

8051 supports total of 6 interrupt sources, meaning that it can recognize up to 6 different events that can interrupt regular program execution. Each of these interrupts can be individually enabled or disabled by configuring the register IE. Clearing the bit EA in the same register can disable also whole system of interrupts.

i. INTERRUPT SERVICE ROUTINE

For every interrupt, there must be an interrupt service routine (ISR). When an interrupt is invoked, the microcontroller runs the interrupt service routine. For every interrupt, there is a fixed location in memory that holds the address of its ISR.

Interrupt	**ROM location (HEX)**	**Pin**
Reset	0000	9
External hardware interrupt 0 (INT 0)	0003	
Timer 0 interrupt (TF 0)	000B	P3.2 (12)
External hardware interrupt 1 (INT 1)	0013	
Timer 1 interrupt (TF 1)	001B	
Serial COM interrupt (R1 and T1)	0023	P3.3 (13)

ii. IE (Interrupt Enable)

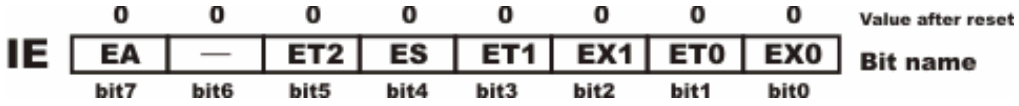


Fig 4.14:Interrupt Enable Register

Following table describes the bits of register IE. Same rule applies to all bits - logical state of 1 enables the appropriate interrupt. ET2 is not used in 8051. It is available in derivatives of 8051 versions where Timer 2 is available.

Bit	Purpose
EA	Enables/disables all interrupt sources
ET2	Timer T2 interrupt
ES	UART and SPI interrupts
ET1	Timer T1 interrupt
EX1	External interrupt: pin INT1
ET0	Timer T0 interrupt
EX0	External interrupt: pin INT0

To enable an interrupt, bit D7 of the IE register (IE) must be set to high to allow the rest of register to take effect. If EA* is high, interrupts are enabled and will be responded to if their corresponding bits in IE are high. If EA* is low, no interrupt will be responded to, even if the associated bit in the IE register is high.

1.4.19 INTERRUPT FLAG SFR REGISTER BIT

The list of all interrupt flags are listed below: Four of the interrupt flags are held in the TCON register while the SCON has the RI and TI flags.

Interrupt	Flag	SFR Register Bit
External 0	IE0	TCON.1
External 1	IE1	TCON.3
Timer 0	TF0	TCON.5
Timer 1	TF1	TCON.7
Serial port	T1	SCON.1
Timer 2	TF2	T2SCON.7 (For 89C51)
Timer 2	EXF2	T2SCON.6 (For 89C51)

iv. INTERRUPT PRIORITY

When the 8051 is powered up, the priorities are assigned according to the table listed.

HIGHEST TO LOWEST PRIORITY

External Interrupt 0	(INT 0)	Highest
Timer Interrupt 0	(TF 0)	
External Interrupt 1	(INT 1)	
Timer Interrupt 1	TF 1)	
Serial Communication	RI + TI)	Lowest

From the table we see that if external hardware interrupts 0 and 1 are activated at the same time, external interrupt 0 is responded to first. Only after INT 0 has been serviced, INT 1 get the node, since INT 1 has the lowest priority. We can alter the sequence of the table by assigning a higher priority to any of the interrupts. This is done by a programming in a register called IP (Interrupt Priority) register. The bit arrangement of the IP register is shown below:

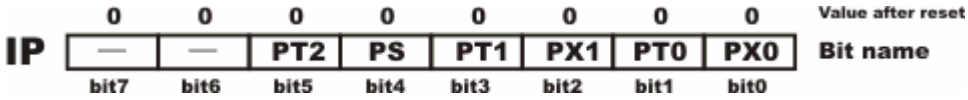


Fig 4.15:Interrupt Priority Register

Priority bit = 1 assign high priority. Priority bit = 0 assigns low priority

-- IP.7 reserved

-- IP.6 reserved

PT2 IP.5 Timer 2 interrupt priority bit (MCS-51 Series only)

PS IP.4 Serial port interrupt priority bit

PT1 IP.3 Timer 1 interrupt priority bit

PX1 IP.2 External interrupt 0 priority bit

PT0 IP.1 Timer 1 interrupt priority bit

PX0 IP.0 External interrupt 0 priority bit

Upon reset, the IP register contains all 0s, making the priority sequence based on the priority table. To give a higher priority to any of the interrupts, we make the corresponding bit in the IP register high. When two or more interrupt bits in the IP register are set to high, those interrupts having higher priority than others are serviced according to the sequence of the priority table.

Upon receiving an interrupt request, following scenario takes place:

1. Current instruction is executed first.

2. Address of the instruction that would be executed next if there was no interrupt request is put away to stack.

2. Depending on the interrupt in question, program counter will take value of one of possible 6 vectors (addresses) according to the table below.

Interrupt source	Vector (address in hex)
IE0	3H
TF0	0BH
IE1	13H
TF1	1BH
RI, TI, SPIF	23H
TF2, EXF2	2BH

   These addresses should hold the appropriate subroutines for handling the interrupts. In practice, instead of actual routines, they only point to the location of appropriate routines in the code.

3. Upon accomplishing the interrupt routine, address of the next instruction to be executed is retrieved from the stack, and the program proceeds from the location where it was interrupted.

1.5 ADDRESSING MODES & INSTRUCTION SET

1.5.1 ADDRESSING MODES

The CPU can access data in various ways. The data could be in a register, or in memory, or to be provided as an immediate value.These various ways of accessing data are called addressing modes. The various addressing modes are determined when it is designed and therefore cannot be changed by the programmer.

1.5.2 8051 ADDRESSING MODES

The 8051 provide a total of five distinct addressing modes. They are:

1. **Immediate**
2. **Direct**
3. **Register**
4. **Register indirect**
5. **Indexed**

We can use direct or register indirect addressing modes to access data stored either in RAM or registers of the 8051.

i. IMMEDIATE ADDRESSING MODE

Immediate addressing is so-named because the value to be stored in memory immediately follows the operation code in memory. That is to say, the instruction itself dictates what value will be stored in memory. Immediate data has to be preceded by "#" sign.

For example, the instruction: **MOV A, #20H**

This instruction uses Immediate Addressing because the Accumulator will be loaded with the value that immediately follows; in this case 20 (hexadecimal). Immediate addressing is very fast since the value to be loaded is included in the instruction. However, since the value to be loaded is fixed at compile-time it is not very flexible.

ii. DIRECT ADDRESSING

In Direct addressing the value to be stored in memory is obtained by directly retrieving it from another memory location.

For example: **MOV A, 30H**

This instruction will read the data from the Internal RAM address 30 (hexadecimal) and store it in the Accumulator.

Also, it is important to note that when using direct addressing any instruction, which refers to an address between 00h, and 7Fh is referring to Internal Memory. Any instruction, which refers to an address between 80h and FFh, is referring to the SFR control registers. Direct addressing is the only method of accessing the special function registers. The lower 128 bytes of internal RAM are also directly addressable.

iii. REGISTER ADDRESSING

Register addressing accesses the eight working registers (R0 - R7) of the selected register bank. This instruction put the operand in a register and manipulates it by referring to the register (by name) in the instruction & in this type of instruction the source & destination registers must match in size.

Example:

```
Mov   A, R0        ;A ↴ contents (R0)
Mov   R2, A        ;R2  ↴ contents (A)
ADD   A, R1        ;A ↴ contents (A) + contents (R1)
```

The least significant bit of the instruction op-code indicates which register is to be used. ACC, B, DPTR and CY, can also be addressed as registers.

iv. REGISTER INDIRECT ADDRESSING

Register indirect addressing is a very powerful addressing mode, which in many cases provides an exceptional level of flexibility. Indirect addressing is also the only way to access the extra 128 bytes of Internal RAM found on an 8052.

Indirect addressing appears as follows:

MOV A, @R0

This instruction causes the 8051 to analyze the value of the R0 register. The 8051 will then load the accumulator with the value from Internal RAM, which is found at the address indicated by R0.

For example, let's say R0 holds the value 40H and Internal RAM address 40H holds the value 67H. When the above instruction is executed the 8051 will check the value of R0. Since R0 holds 40H the 8051 will get the value out of Internal RAM address 40H (which holds 67H) and store it in the Accumulator. Thus, the Accumulator ends up holding 67H.

Indirect addressing always refers to Internal RAM; it never refers to an SFR. Indirect addressing only can access the upper half of the internal RAM. Access to the full 64 Kbytes of external data memory address space is accomplished by using the 16-bit data pointer. Execution of PUSH and POP instructions also uses register indirect addressing. The stack may reside anywhere in the internal RAM.

v. INDEX ADDRESSING

Indexed addressing mode is widely used in accessing data elements of look-up table entries located in the program ROM space of the 8051. Indexed addressing use a register for storing the pointer and another register for an offset. The Effective address is the sum of base & offset.

Eg:

```
MOVC A, @A+DPTR     ; A ↴ ext_code_mem [(A + DPTR)]
MOVC A, @A+PC       ; A ↴ ext_code_mem [(A + PC)]
```

1.5.3 8051 INSTRUCTION SET

The 8051, 8-bit microcontroller family instruction set includes 111 instructions, 49 of which are single-byte, 45 two-byte and 17 three-byte instructions. The instruction op-code format consists of a function mnemonic followed by a destination & source operand field. The instruction set is divided into four functional groups:

- **Data transfer**
- **Arithmetic**
- **Logic**
- **Control transfer**

i. DATA TRANSFER INSTRUCTIONS

Data transfer operations are divided into three classes:

- General - purpose
- Accumulator-specific
- Address-object

None of these operations affects the PSW flag settings except a POP or MOV directly to the PSW.

Examples

- **MOV A, #45 - Immediate Addressing Mode**
- **MOV A, R1 - Register Addressing Mode**
- **MOV 45h,A - Direct Addressing Mode**
- **MOV @R1, 32h - Indirect Addressing Mode**

ii. ARITHMETIC INSTRUCTIONS

The MCS-51 family microcontrollers have four basic mathematical operations. Only 8-bit operations using unsigned arithmetic are supported directly. The overflow flag, however, permits the addition and subtraction operation to serve for both unsigned and signed binary integers. Arithmetic can also be performed directly on packed BCD representations.

Examples

- **ADD A, #84 - Immediate Addressing Mode**
- **SUBB A, R2 - Register Addressing Mode**
- **ADD 73h,A - Direct Addressing Mode**
- **ADDC @R1, 25h - Indirect Addressing Mode**

iii. LOGIC INSTRUCTIONS

The MCS-51 family microcontrollers perform basic logic operations on both bit and byte operands.

BIT LEVEL (SINGLE OPERAND) OPERATIONS

In 8051 internal RAM and SFRs can be addressed by the address of each bit within a byte. This bit addressing is very convenient when we wish to alter a single bit of a byte. The ability to operate on individual bits creates the need for an area of RAM that contains data addresses that hold a single bit. The bit addresses are numbered from 00H to 7FH to represent the 128d bit addresses that exist from byte addresses 20H to 2FH.

- **CLR** sets A or any directly addressable bit to zero (0).
- **SETB** sets any directly bit-addressable bit to one (1).
- **CPL** is used to complement the contents of the A register without affecting any flag, or any directly addressable bit location.
- RL, RLC, RR, RRC, SWAP are the five operations that can be performed on A. RL, rotate left, RR, rotate right, RLC, rotate left through carry, **RRC**, rotate right through carry, and SWAP, rotate left four. For **RLC** and **RRC** the **CY** flag become equal to the last bit rotated out. **SWAP** rotates A left four places to exchange bits 3 through 0 with bits 7 through 4.

BYTE LEVEL (TWO-OPERAND) OPERATIONS

- **ANL** performs bit wise logical AND of two operands (for both bit and byte operands) and returns the result to the location of the first operand.
- **ORL** performs bit wise logical OR of two source operands (for both bit and byte operands) and returns the result to the location of the first operand.
- **XRL** performs logical Exclusive OR of two source operands (byte operands) and returns the result to the location of the first operand.

Example

- **ANL A, #45h - Immediate Addressing Mode**
- **ORL A, R2 - Register Addressing Mode**
- **XRL 52h, A - Direct Addressing Mode**
- **ANL @R3, 65h - Indirect Addressing Mode**

iv. CONTROL TRANSFER INSTRUCTIONS

There are three classes of control transfer operations: unconditional calls, returns, jumps, conditional jumps, and interrupts. All control transfer operations, some upon a specific condition, cause the program execution to continue a non-sequential location in program memory.

Example

- **CJNE A,#22H,loop - Immediate Addressing Mode**
- **DJNZ R1,loop - Register Addressing Mode**
- **DJNZ 30H,loop - Direct Addressing Mode**
- **JMP @A+DPTR - Indirect Addressing Mode**

Notes on Data Addressing Modes

Rn - Working register R0-R7

direct - 128 internal RAM locations, any l/O port,control or status register

@Ri - Indirect internal or external RAM location addressed by register R0 or R1

#data - 8-bit constant included in instruction

#data 16 - 16-bit constant included as bytes 2 and 3 of instruction

Bit - 128 software flags, any bit-addressable l/O pin, control or status bit

A - Accumulator

Notes on Program Addressing Modes:

addr16 - Destination address for LCALL and LJMP may be anywhere within the 64-Kbyte program memory address space.

addr11 - Destination address for ACALL and AJMP will be within the same 2-Kbyte page of program memory as the first byte of the following instruction.

rel - SJMP and all conditional jumps include an 8-bit offset byte. Range is + **127/– 128** bytes relative to the first byte of the following instruction.

1.5.4 INSTRUCTION SET SUMMARY:

Mnemonic		Description	Byte	Cycle
Arithmetic Operations				
ADD	A,Rn	Add register to accumulator	1	1
ADD	A,direct	Add direct byte to accumulator	2	1
ADD	A @Ri	Add indirect RAM to accumulator	1	1
ADD	A,#data	Add immediate data to accumulator	2	1
ADDC	A,Rn	Add register to accumulator with carry flag	1	1
ADDC	A,direct	Add direct byte to A with carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with carry flag	1	1
ADDC	A, #data	Add immediate data to A with carry flag	2	1
SUBB	A,Rn	Subtract register from A with borrow	1	1
SUBB	A,direct	Subtract direct byte from A with borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with borrow	1	1
SUBB	A,#data	Subtract immediate data from A with borrow	2	1
INC	A	Increment accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
DEC	A	Decrement accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
INC	DPTR	Increment data pointer	1	2
MUL	AB	Multiply A and B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal adjust accumulator	1	1

+

Mnemonic		Description	Byte	Cycle
Logic Operations				
ANL	A,Rn	AND register to accumulator	1	1
ANL	A,direct	AND direct byte to accumulator	2	1
ANL	A,@Ri	AND indirect RAM to accumulator	1	1
ANL	A,#data	AND immediate data to accumulator	2	1
ANL	direct,A	AND accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to accumulator	1	1
ORL	A,direct	OR direct byte to accumulator	2	1
ORL	A,@Ri	OR indirect RAM to accumulator	1	1
ORL	A,#data	OR immediate data to accumulator	2	1
ORL	direct,A	OR accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive OR register to accumulator	1	1
XRL	A direct	Exclusive OR direct byte to accumulator	2	1
XRL	A,@Ri	Exclusive OR indirect RAM to accumulator	1	1
XRL	A,#data	Exclusive OR immediate data to accumulator	2	1
XRL	direct,A	Exclusive OR accumulator to direct byte	2	1
XRL	direct,#data	Exclusive OR immediate data to direct byte	3	2
CLR	A	Clear accumulator	1	1
CPL	A	Complement accumulator	1	1
RL	A	Rotate accumulator left	1	1
RLC	A	Rotate accumulator left through carry	1	1
RR	A	Rotate accumulator right	1	1
RRC	A	Rotate accumulator right through carry	1	1
SWAP	A	Swap nibbles within the accumulator	1	1
Data Transfer				
MOV	A,Rn	Move register to accumulator	1	1
MOV	A,direct	Move direct byte to accumulator	2	1
MOV	A,@Ri	Move indirect RAM to accumulator	1	1

Mnemonic		Description	Byte	Cycle
MOV	A,#data	Move immediate data to accumulator	2	1
MOV	Rn,A	Move accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct byte	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri, #data	Move immediate data to indirect RAM	2	1
MOV	DPTR, #data16	Load data pointer with a 16-bit constant	3	2
MOVC	A,@A + DPTR	Move code byte relative to DPTR to accumulator	1	2
MOVC	A,@A + PC	Move code byte relative to PC to accumulator	1	2
MOVX	A,@Ri	Move external RAM (8-bit addr.) to A	1	2
MOVX	A,@DPTR	Move external RAM (16-bit addr.) to A	1	2
MOVX	@Ri,A	Move A to external RAM (8-bit addr.)	1	2
MOVX	@DPTR,A	Move A to external RAM (16-bit addr.)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with accumulator	1	1
XCH	A,direct	Exchange direct byte with accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with accumulator	1	1
XCHD	A,@Ri	Exchange low-order nibble indir. RAM with A	1	1

Boolean Variable Manipulation

CLR	C	Clear carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set carry flag	1	1
SETB	bit	Set direct bit	2	1
CPL	C	Complement carry flag	1	1
CPL	bit	Complement direct bit	2	1

Mnemonic		Description	Byte	Cycle
ANL	C,bit	AND direct bit to carry flag	2	2
ANL	C,/bit	AND complement of direct bit to carry	2	2
ORL	C,bit	OR direct bit to carry flag	2	2
ORL	C,/bit	OR complement of direct bit to carry	2	2
MOV	C,bit	Move direct bit to carry flag	2	1
MOV	bit,C	Move carry flag to direct bit	2	2

Program and Machine Control

ACALL	addr11	Absolute subroutine call	2	2
LCALL	addr16	Long subroutine call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute jump	2	2
LJMP	addr16	Long iump	3	2
SJMP	rel	Short jump (relative addr.)	2	2
JMP	@A + DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if accumulator is zero	2	2
JNZ	rel	Jump if accumulator is not zero	2	2
JC	rel	Jump if carry flag is set	2	2
JNC	rel	Jump if carry flag is not set	2	2
JB	bit,rel	Jump if direct bit is set	3	2
JNB	bit,rel	Jump if direct bit is not set	3	2
JBC	bit,rel	Jump if direct bit is set and clear bit	3	2
CJNE	A,direct,rel	Compare direct byte to A and jump if not equal	3	2
CJNE	A,#data,rel	Compare immediate to A and jump if not equal	3	2
CJNE	Rn,#data rel	Compare immed. to reg. and jump if not equal	3	2
CJNE	@Ri,#data,rel	Compare immed. to ind. and jump if not equal	3	2
DJNZ	Rn,rel	Decrement register and jump if not zero	2	2
DJNZ	direct,rel	Decrement direct byte and jump if not zero	3	2
NOP		No operation	1	1

1.6 8051 DERIVATIVES

1.6.1 DERIVATIVE ADVANTAGES

The 8051 microcontroller family remains one of the most popular processors in the world. Its ease of use and relatively high performance make it ideal for many applications, including portable and handheld products. The introduction of 8051 derivatives by several manufactures allows a way for existing 8051 designs to improve their power efficiency without a costly redesign. The benefits of the 8051 derivatives as listed below.

- A high performance CPU allows the processor clock to be slowed, resulting in the same level of performance at less power. Alternatively, the performance of an existing system can be increased without increasing power consumption.

- The high-speed microcontroller incorporates features such as watchdog timers, additional UARTs, and precision reset circuits. External components consume more power.

- The introduction of power management modes provides a low-power alternative to the Idle mode.

- Conventional 8051 architectures require the processor to operate at the maximum clock rate, even if only minimal processing power is required. The benefits of a programmable clock rate and high-performance core can be combined with the Stop mode to greatly reduce power consumption.

The most direct approach to decreasing power consumption of an 8051-based design is to improve the efficiency of the microcontroller. The original design of the 8051 was based on a 12-clock, 2-fetch-permachine cycle architecture. The high-speed microcontroller family, however, uses a 4- or 1-clock per machine cycle core. It is more computationally efficient and requires fewer clock cycles to execute an instruction, resulting in faster execution times and increased maximum clock rates.

Integrating peripherals on-chip is a method of power conservation. When driving a signal off-chip, the generating device must contend with the switching power required to drive the external loads and any DC losses. Microcontroller-based systems typically use a number of peripherals. These range from external UARTs and power-on reset circuitry to watchdog timers. One of the strengths of the 8051 product family is the large number of peripheral functions that are available on-chip. In addition to simplifying a design by eliminating components, integrated peripherals also can reduce power consumption.

Another 8051 feature that is not commonly perceived as a peripheral is program memory. All 8051 derivatives incorporate various amounts of on-chip program memory. This is desired by many system designers as a method of reducing the component count and board area, but it also improves battery life in portable systems. As mentioned previously, this will reduce power consumption by eliminating the need to drive an external bus. There is an additional power savings when using on-chip memory.

The use of on-chip data memory instead of external RAM will save power. The enlarged scratchpad of the 80C32 derivatives (256 bytes) is sufficient for stack operations and some data storage in small programs, eliminating the need for external RAM.

Another critical system component from a power standpoint is the clock management. The operating frequency of the microcontroller is the single largest factor affecting the power consumption of the device. Although the system clock frequency is primarily a hardware function, the 8051 has the ability to exercise limited control over it. These methods rely on slowing or halting the internal operating frequency of all or part of the device.

In the Power Management Modes 8051 derivative architecture has used two clock control modes: Idle and Stop. Idle mode halts operation of the CPU, but keeps the on-chip, general-purpose timers operational. In a power-sensitive application, these timers are used to periodically wake the device to perform a task or to poll if a task should be performed. This consumes a considerable amount of power, considering the timers are basically operating in a "standby" capacity.

Another interesting features found in derivatives are ADC, DAC and PWM integrated in on-chip itself. These features simplify the embedded system design and programming easily.

In this chapter we will discuss about the two widely used 8051 derivatives from PHILIPS & ANLOG DEVICES companies. First one is the 89C51 series from PHILIPS SEMICONDUCTOR where FLASH memory is important in embedded system design, and the second one from 812 series of microconverter from ANALOG DEVICES which incorporates built-in high speed ADC & DAC in a single chip.

1.6.2 PHILIPS 89C51 – FLASH 8051 Derivative

This device is a Single-Chip 8-Bit Microcontroller manufactured in advanced CMOS process and is a derivative of the 80C51 microcontroller family. The instruction set is 100% compatible with the 80C51 instruction set. The device also has four 8-bit I/O ports, three 16-bit timer/event counters, a multi-source, four-priority-level, nested interrupt structure, an enhanced UART and on-chip oscillator and timing circuits. The added features of the P89C51RB2/RC2/RD2 makes it a powerful microcontroller for applications that require pulse width modulation, high-speed I/O and up/down counting capabilities such as motor control.

This device executes one machine cycle in 6 clock cycles (12 clock mode ICs are also available), hence providing twice the speed of a conventional 80C51. The configuration bit lets the user select conventional 12 clock timing if desired. Three different versions of 89C51are available listed below:

P89C51RB2 --- 16KB FLASH / 512 B RAM
P89C51RC2 --- 32KB FLASH / 512 B RAM
P89C51RD2 --- 64KB FLASH / 1KB RAM

1.6.3 IMPORTANT FEATURES

• 80C51 Central Processing Unit
• On-chip Flash Program Memory with In-System Programming(ISP)
• 6 clocks per machine cycle operation (standard)
• 12 clocks per machine cycle operation (optional)
• Speed up to 20 MHz with 6 clock cycles per machine cycle (40 MHz equivalent performance); up to 33 MHz with 12 clocks per machine cycle
• RAM expandable externally to 64 kB
• 4 level priority interrupt
• 7 interrupt sources
• Four 8-bit I/O ports
• Full-duplex enhanced UART
• Power control modes

– Clock can be stopped and resumed
– Idle mode
- Power down mode

• Programmable clock out
• Second DPTR register
• Programmable Counter Array (PCA)
• PWM
• Capture/compare

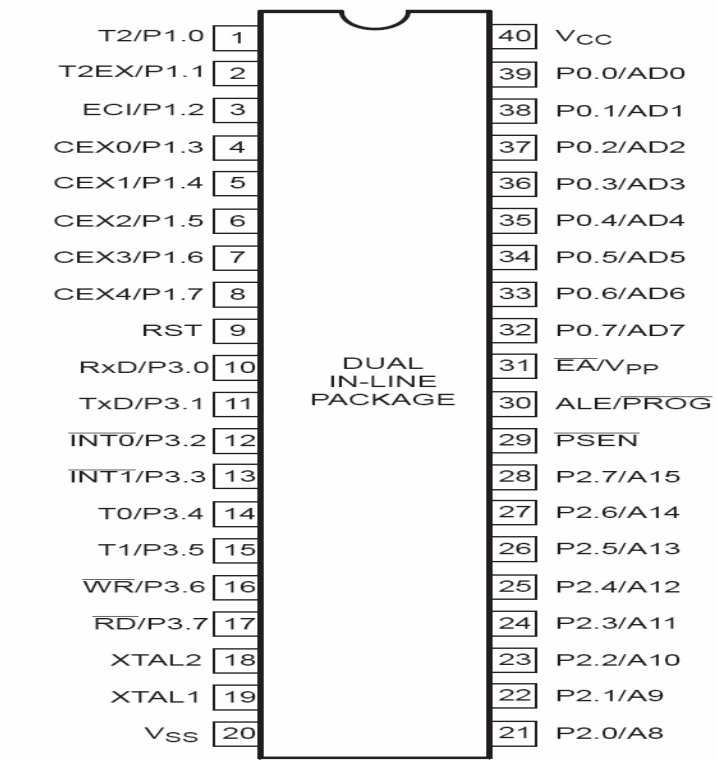


Fig 6.1: P89C51 PIN Details

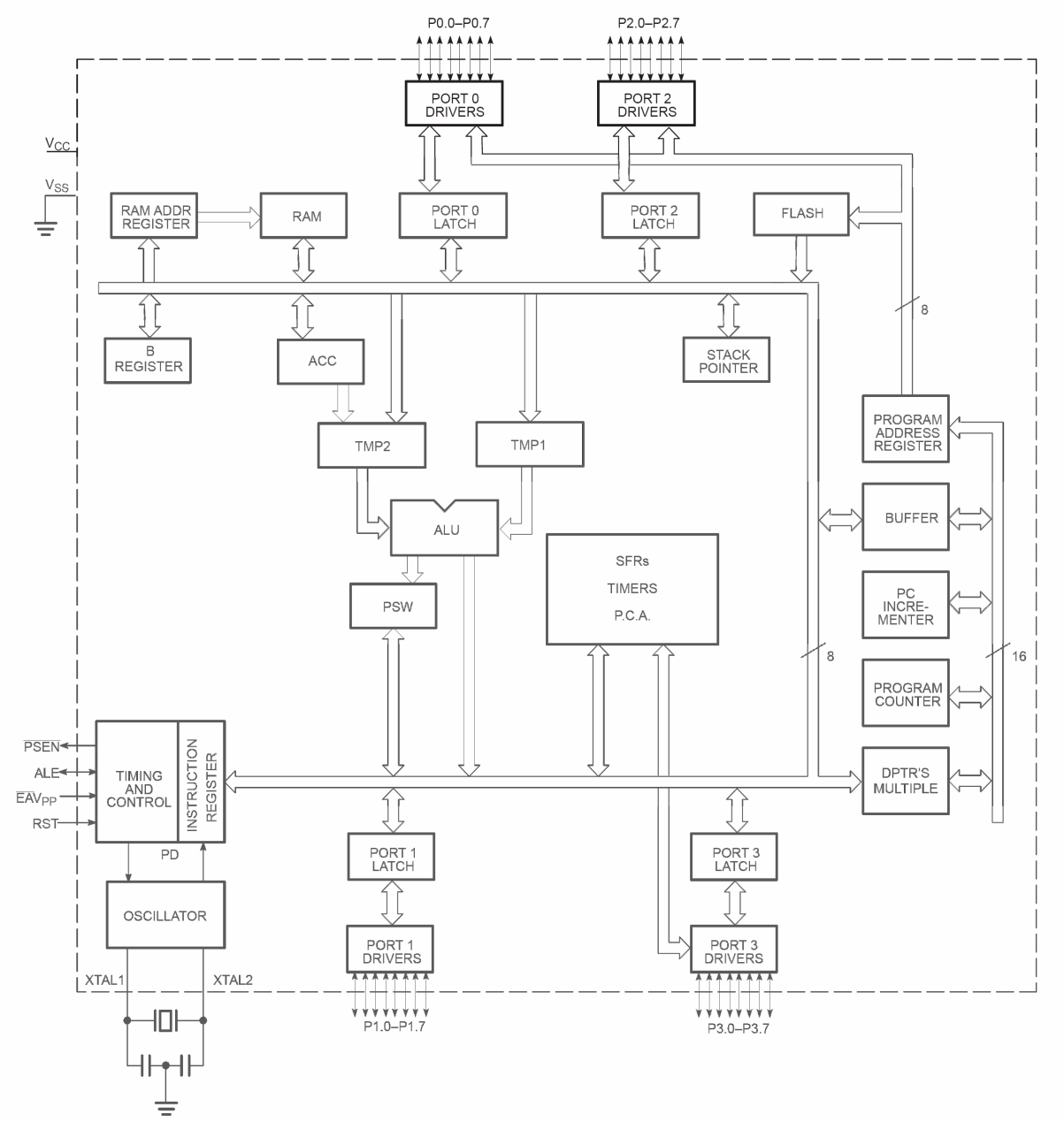


Fig 6.2: P89C51 BLOCK DIAGRAM

1.6.4 ARCHITECTURE DETAILS

The 89C51 pin details and block diagram are shown above. The architecture features are same as the 8051 controller, and so here we will only explained the additional features available in the chip.

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier. The pins can be configured for use as an on-chip oscillator. This device is configured to operate using 6 clock periods per machine cycle. It may be optionally configured to operate at 12 clocks per machine cycle. Once 12 clock mode has been configured, it cannot be changed back to 6 clock mode.

i. LOW POWER MODES

Stop Clock Mode

The static design enables the clock speed to be reduced down to 0 MHz (stopped). When the oscillator is stopped, the RAM and Special Function Registers retain their values. This mode allows step-by-step utilization and permits reduced system power consumption by lowering the clock frequency down to any value. For lowest power consumption the Power Down mode is suggested.

Idle Mode

In the idle mode, the CPU puts itself to sleep while all of the on-chip peripherals stay active. The instruction to invoke the idle mode is the last instruction executed in the normal operating mode before the idle mode is activated. The CPU contents, the on-chip RAM, and all of the special function registers remain intact during this mode. The idle mode can be terminated either by any enabled interrupt, or by a hardware reset which starts the processor in the same manner as a power-on reset.

Power-Down Mode

To save even more power, a Power Down mode can be invoked by software. In this mode, the oscillator is stopped and the instruction that invoked Power Down is the last instruction executed. The on-chip RAM and Special Function Registers retain their values down to 2.0 V and care must be taken to return VCC to the minimum specified operating voltages before the Power Down Mode is terminated. Either a hardware reset or external interrupt can be used to exit from Power Down. Reset redefines all the SFRs but does not change the on-chip RAM. An external interrupt allows both the SFRs and the on-chip RAM to retain their values. To properly terminate Power Down, the reset or external interrupt should not be executed before VCC is restored to its normal operating level and must be held active long enough for the oscillator to restart and stabilize (normally less than 10 ms).

ii. THE ADDITIONAL TIMER – TIMER 2

Timer 2 is a 16-bit Timer/Counter which can operate as either an event timer or an event counter, as selected by C/T2* in the special function register T2CON. Timer 2 has three operating modes: Capture, Auto-reload (up or down counting), and Baud Rate Generator, which are selected by bits in the T2CON.

Timer 2 Operating Modes

Timer 2 operating modes are shown in the below table.

RCLK + TCLK	CP/$\overline{RL2}$	TR2	MODE
0	0	1	16-bit Auto-reload
0	1	1	16-bit Capture
1	X	1	Baud rate generator
X	X	0	(off)

iii. Special Function Registers

The additional SFR available in 89C51 is listed below.

CCAP0H	Module 0 Capture High	FAH	
CCAP1H	Module 1 Capture High	FBH	
CAP2H	Module 2 Capture High	FCH	
CCAP3H	Module 3 Capture High	FDH	
CCAP4H	Module 4 Capture High	FEH	
CCAP0L	Module 0 Capture Low	EAH	
CCAP1L	Module 1 Capture Low	EBH	
CCAP2L	Module 2 Capture Low	ECH	
CCAP3L	Module 3 Capture Low	EDH	
CCAP4L	Module 4 Capture Low	EEH	
CCAPM0	Module 0 Mode		DAH
CCAPM1	Module 1 Mode		DBH
CAPM2	Module 2 Mode		DCH
CCAPM3	Module 3 Mode		DDH
CCAPM4	Module 4 Mode		DEH
CCON	PCA Counter Control D8H		
CH	PCA Counter High	F9H	
CL	PCA Counter Low	E9H	
CMOD	PCA Counter Mode		D9H
AUXR	Auxiliary	8EH	
AUXR1	Auxiliary 1	A2H	
IPH	Interrupt Priority High	B7H	
RCAP2H	Timer 2 Capture High	CBH	
RCAP2L	Timer 2 Capture Low CAH		
SADDR	Slave Address	A9H	
SADEN	Slave Address Mask	B9H	
T2CON	Timer 2 Control	C8H	
T2MOD	Timer 2 Mode Control	C9H	
TH2	Timer High 2	CDH	
TL2	Timer Low 2	CCH	
WDTRST	Watchdog Timer Reset	A6H	

iv. Interrupt Sources

The P89C51 has a 7 source four-level interrupt structure. There are 3 SFRs associated with the four-level interrupt. They are the IE, IP, and IPH. The IPH (Interrupt Priority High) register makes the four-level interrupt structure possible. The IPH is located at SFR B7H.

PRIORITY BITS		INTERRUPT PRIORITY LEVEL
IPH.x	IP.x	
0	0	Level 0 (lowest priority)
0	1	Level 1
1	0	Level 2
1	1	Level 3 (highest priority)

The priority scheme for servicing the interrupts is the same as that for the 80C51, except there are four interrupt levels rather than two as on the 80C51. The bit status of IE & IP registers is already explained in 8051.

SOURCE	POLLING PRIORITY	VECTOR ADDRESS
X0	1	03H
T0	2	0BH
X1	3	13H
T1	4	1BH
PCA	5	33H
SP	6	23H
T2	7	2BH

Interrupt Table

v. Programmable Counter Array (PCA)

The Programmable Counter Array available on the 89C51is a special 16-bit Timer that has five 16-bit capture/compare modules associated with it. Each of the modules can be programmed to operate in one of four modes:

Rising and/or falling edge capture
Software timer
High-speed output
Pulse Width Modulator

Each module has a pin associated with it in port 1. The basic PCA configuration is shown in Figure.

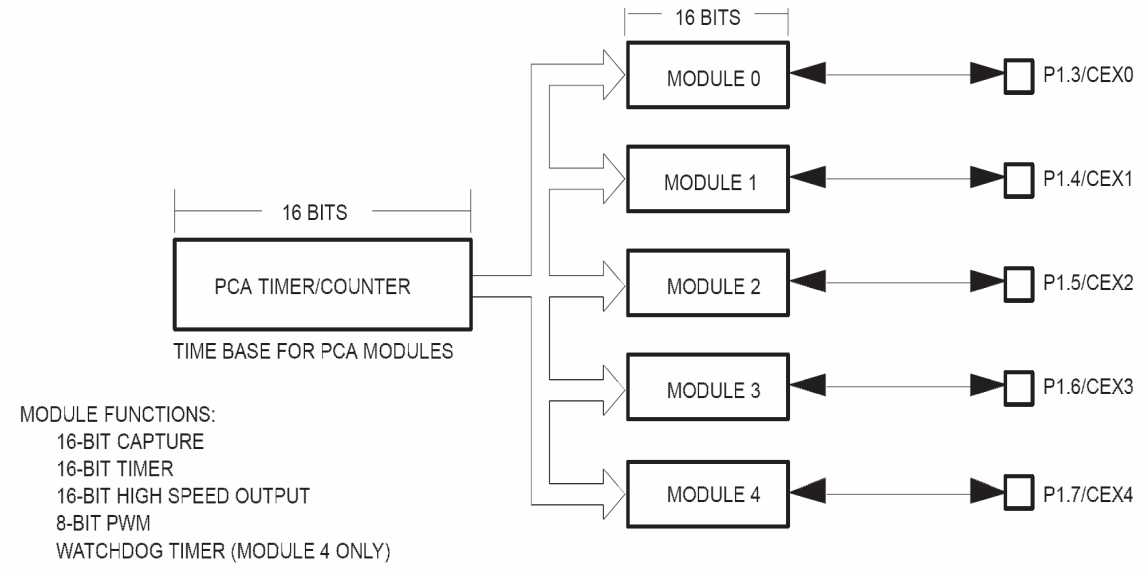


Fig 6.3: Programmable Counter Array (PCA)

vi. FLASH EPROM MEMORY

The P89C51 Flash memory augments EPROM functionality with in-circuit electrical erasure and programming. The Flash can be read and written as bytes. The Chip Erase operation will erase the entire program memory. The Block Erase function can erase any Flash block. In-system programming and standard parallel programming are both available. On-chip erase and write timing generation contribute to a user friendly programming interface. The P89C51 Flash reliably stores memory contents even after 10,000 erase and program cycles. The P89C51 uses a +5 V VPP supply to perform the Program/Erase algorithms.

FLASH MEMORY FEATURES

- Flash EPROM internal program memory with Block Erase.
- Internal 1 kB fixed boot ROM, containing low-level in-system programming routines and a default serial loader. The Boot ROM can be turned off to provide access to the full 64 kB Flash memory.
- Boot vector allows user provided Flash loader code to reside anywhere in the Flash memory space. This configuration provides flexibility to the user.
- Default loader in Boot ROM allows programming via the serial port without the need for a user provided loader.
- Up to 64 kB external program memory if the internal memory is disabled (EA = 0).
- Programming and erase voltage +5 V.
- Read/Programming/Erase:

  – Byte-wise read (100 ns access time).
  – Byte Programming (20 ?s).
  – Typical erase times:
  – Block Erase (8 kB or 16 kB) in 3 seconds.
  – Full Erase (64 kB) in 3 seconds.

- Parallel programming compatible hardware interface to programmer.
- In-system programming facility.
- Programmable security for the code in the Flash.
- 10,000 minimum erase/program cycles for each byte.
- 10-year minimum data retention.

FLASH ORGANIZATION

The P89C51 contains 64K bytes of Flash program memory. This memory is organized as 5 separate blocks. The first two blocks are 8 kB in size, filling the program memory space from address 0 through 3FFF hex. The final three blocks are 16 kB in size and occupy addresses from 4000 through FFFF hex.

Boot ROM

When the microcontroller programs its own Flash memory, all of the low level details are handled by code that is permanently contained in a 1 kB Boot ROM that is separate from the Flash memory. The Boot ROM overlays the program memory space at the top of the address space from FC00 to FFFF hex, when it is enabled. The Boot ROM may be turned off so that the upper 1 kB of Flash program memory are accessible for execution.

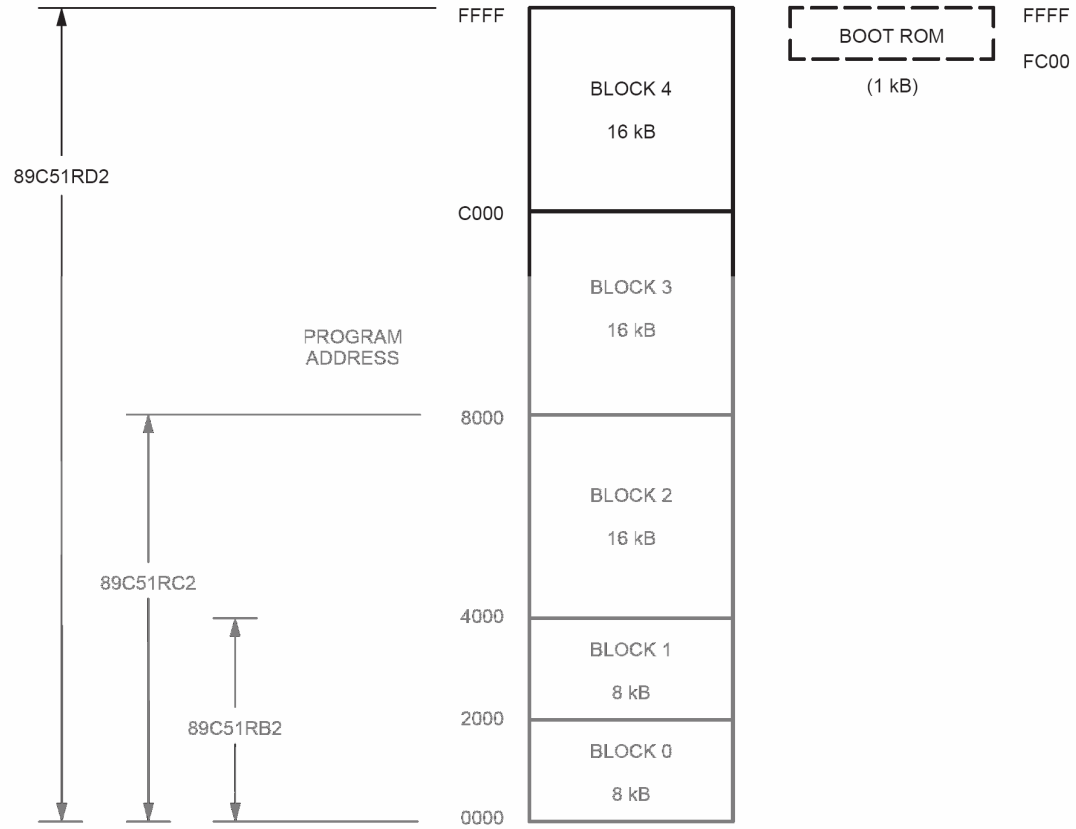


Fig 6.4: Flash Memory Configuration

Power-On Reset Code Execution

The P89C51 contains two special Flash registers: the BOOT VECTOR and the STATUS BYTE. At the falling edge of reset, the P89C51 examines the contents of the Status Byte. If the Status Byte is set to zero, power-up execution starts at location 0000H, which is the normal start address of the user's application code. When the Status Byte is set to a value other than zero, the contents of the Boot Vector is used as the high byte of the execution

address and the low byte is set to 00H. The factory default setting is 0FCH, corresponds to the address 0FC00H for the factory masked-ROM ISP boot loader. A custom boot loader can be written with the Boot Vector set to the custom boot loader.

In-System Programming (ISP)

The In-System Programming (ISP) is performed without removing the microcontroller from the system. The In-System Programming (ISP) facility consists of a series of internal hardware resources coupled with internal firmware to facilitate remote programming of the P89C51 through the serial port. This firmware is provided by Philips and embedded within each P89C51 device. For In-System Programming facility WinISP, a software utility is available from Philips. The security feature protects against software piracy and prevents the contents of the Flash from being read. The Security Lock bits are located in Flash. The P89C51 has three programmable security lock bits that will provide different levels of protection for the on-chip code and data.

SECURITY LOCK BITS[1]			PROTECTION DESCRIPTION
LB1	LB2	LB3	
X	X	X	MOVC instructions executed from external program memory are disabled from fetching code bytes from internal memory.
1	X	X	Block erase is disabled. Erase or programming of the status byte or boot vector is disabled.
X	1	X	Verify of code memory is disabled.
X	X	1	External execution is disabled.

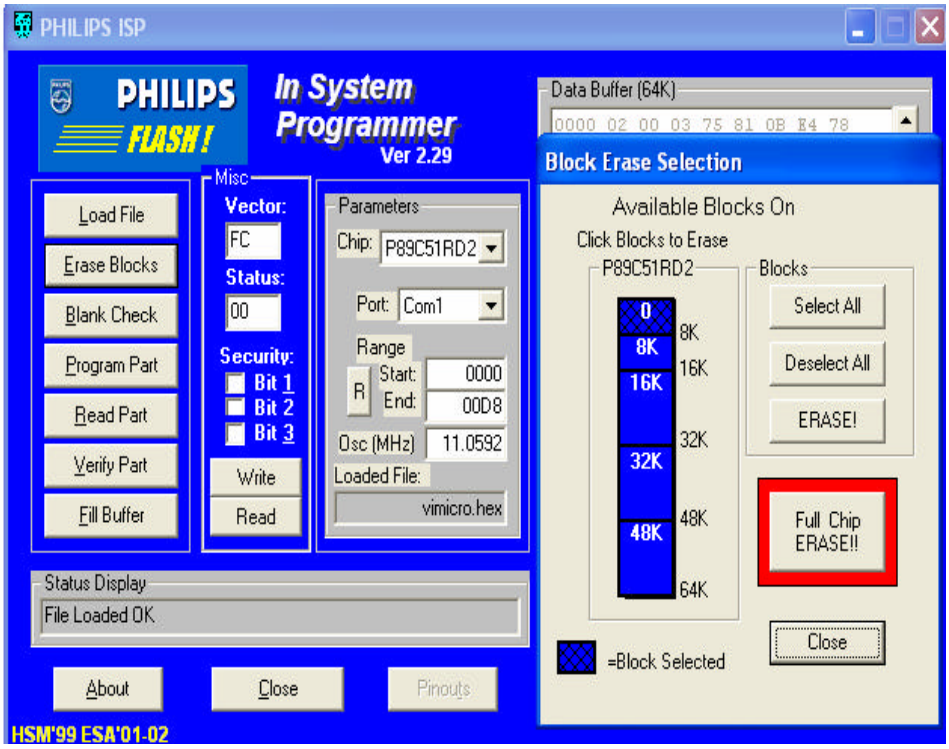


Fig 6.5: WINISP Software

PROGRAMMING STEPS

1. Connect a serial cable to the board and one of your PC's serial ports.
2. Power up the ISP board.
3. Boot up your computer, start MS Windows and the WinISP software.
4. In WinISP, select the chip type, clock frequency (89C1RD2 / 11.0592 MHz for VPC89C51 board) and the serial port number the board is connected to.
5. Force the Board into ISP mode. To do this, give proper settings in the tiny switch (PSEN* makes to LOW & EA* makes to HIGH) & reset the hardware.
6. Verify the board will communicate with the software by pressing the Read button in Misc window in WINISP software. If proper configuration is established it will give the message " Boot Vector Read OK" in status window. Please verify the vector (FC) & Status (00) bits are correctly loaded.
7. The system is now ready. You can use the buttons on the left of the WinISP window to load hex files, program/erase/verify the part, set/reset the security bits etc. Before loading the program in to the chip you have to use the Block Erase function to erase the content of the memory if it is not blank.
8. The Board can be reset at any time to run the program.
9. Do not program the lock bits; this will disable chip access by WinISP.

CHAPTER - 2

SOFTWARE EXAMPLES

OBJECTIVES:

To introduce the user to,

a) The **Load and Exchange** operations available with Intel 8051.

b) The various **Arithmetic and Logical** operations.

c) The **Boolean Bit addressing** instructions.

d) The Flag Register (Program Status Word) and **how to test the flags** for a specific condition.

e) The manipulation of arrays of data.

f) The **Stack and Subroutines** of 8051.

g) The implementation of **software delays** using delay loops.

2.1 INTRODUCTION:

The Intel 8051's excellent and the most powerful instruction set offers possibilities in control areas, Serial I/O, arithmetic, byte and bit manipulation.

It has an instruction set comprising of 140 one-byte instructions, 91 two-byte instructions, 24 three-byte instructions; an instruction cycle time of 1 µs.(12 MHz clock) in which 58% of instructions are executed in 1 cycle, 40% in 2 cycles and 2% in 4 cycles (for Multiply and Divide); direct memory to memory (on-chip RAM) transfer; memory mapped type instructions in which a register's contents can be moved (MOV) directly to an external port or vice-versa; ability to move an I/O pin bit to another I/O pin bit (via the "Boolean Accumulator", the carry flag) (I/O pin bits have individual address).

The above mentioned and various other instructions of the 8051 will be reviewed in this section with the intent of simplifying them and grouping them into logical categories.

In this chapter, you will encounter experiments that will help you to enhance you ability as a software programmer. Only the basic instructions of 8051 are used in this chapter which will be easy for you to follow. However, if possible, they can be replaced by some other equivalent instructions.

When developing programs on your own, try to make it **SMALL AND EFFICIENT.**

SMALL in the sense, it should occupy less memory space. **EFFICIENT** in the sense, the desired task must be done in the most effective way by utilising the most efficient algorithm.

Here, we will examine the capabilities provided by the 8051 in terms of the following groups:

i) Data transfer Operations.

ii) Arithmetic and Logical Operations.

iii) Bit addressable operations.

iv) Flag Operations.

v) Branch Operations.

vi) Array Operations.

vii) Stack and Subroutines.

viii) Delay Loops.

You are requested to follow the guidelines given below, while going through this Chapter. It will be very helpful for your strong foundation in the field of programming.

i) Study the different categories of instructions explained here.

ii) Understand the examples and work them out using a trainer and check for specified results.

iii) Read through the exercises and try each and every program in your trainer. There is no other way to ensure that your program is correct.

iv) Write your programs efficiently, i.e. the task desired must be done using the simplest and most effective algorithm.

v) Write comments wherever possible, for it helps in debugging your program some time later.

vi) Your comments should not describe the purpose of the presence of that instruction in your program. The comments can be ungrammatical, but crisp and clear.

By the end of this Chapter, you will become more familiar with the programming methods, so that the next Chapter on hardware programming can be done very easily.

List of Symbols and Notations Used:

Sl. No.	NOTATION	DESCRIPTION
1	LSB	Least Significant Bit
2	MSB	Most Significant Bit
3	LSD	Least Significant Digit (Byte)
4	MSD	Most Significant Digit (Byte)
5	LN	Lower Nibble
6	HN	Higher Nibble
7	H	Hexa decimal
8	D	Decimal
9	B	Binary
10	(DPTR)	Contents of DPTR
11	[(DPTR)]	Contents of memory location held by DPTR register
12	(X)	Contents of X where X may be any register

NOTE:

i) All address and data are in hex unless otherwise specified.

ii) Flowcharts are given wherever necessary.

2.2 LOAD & EXCHANGE OPERATIONS:

Data Transfer instructions transfer data between registers, between a register and memory and between a register and an input/output device. This includes Load, Store and Exchange instructions. Of all the above, you have got both 8-bit as well as 16-bit operations.

The following are some examples to explain the different addressing methods to load data to registers and memory and also about the exchange of data between registers.

2.2.1 PROGRAM 1 - IMMEDIATE, REGISTER, DIRECT AND INDIRECT ADDRESSING:

OBJECTIVE:

To initialise data to registers and memory using Immediate, Register, Direct and Indirect addressing modes.

THEORY:

Initialisation of data to registers and memory is accomplished using the addressing mode which is highly dependent on the way the operand is available. The above said addressing modes will be used in the following program to initialise registers and memory. In the immediate addressing mode, the data operated upon is in the location immediately following the opcode. In the register addressing mode, the opcode itself specifies the register in operation. In the direct addressing mode, the operand is specified by an 8-bit address field in the instruction. In the Indirect addressing mode, the operand will be in memory whose address is held either by an 8-bit or 16-bit register.

EXAMPLE:

To initialise the following Registers and memory contents as specified below.

(R0) = 9A
(B) = 1F
(DPTR) = 4500
(R6) = 9A

PROGRAM:

```
        MOV     R0,#9A          ; (R0)  = 9A            - Immediate
        MOV     B,#1F           ; (B)   = 1F            - Immediate
        MOV     DPTR,#4500      ; (DPL) = 00;
                                ; (DPH) = 45            - Immediate
        MOV     R6,A            ; (R6) <-- (A)          - Register
        MOV     A,20            ; (A)  <-- (20)         - Direct
        MOVX    A,@DPTR         ; (A)  <-- ((DPTR))     - Indirect
        XCH     A,R0            ; (A)  <--> (R0)        - Register
HERE:   SJMP    HERE
```

OBJECT CODES:

The above program written in 8051 mnemonics should be decoded into its own binary equivalent for the microprocessor to understand and execute. These binary equivalents are called **OBJECT CODES** or simply **OPCODES**. The instruction set of 8051 is given in the appendices at the back of this manual for your reference.

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	78	MOV R0,#9A
4101	9A	
4102	75	MOV B,#1F
4103	F0	
4104	1F	
4105	90	MOV DPTR,#4500
4106	45	
4107	00	
4108	FE	MOV R6,A
4109	E5	MOV A,20
410A	20	
410B	E0	MOVX A,@DPTR
410C	C8	XCH A,R0
410D	80	HERE: SJMP HERE
410E	FE	

PROCEDURE:

i) Key in the opcodes from the address 4100.

ii) Single step through the program and check for the contents of register or memory location, after every individual instruction.

DISCUSSION:

The above specified instructions are some among the load and exchange group of 8051. In the above example, you stepped through a simple **'DO-NOTHING'** program. Note that while specifying 16-bit operands, the MSD is specified first followed by LSD.

2.2.2 PROGRAM 2 - INDEXED ADDRESSING:

OBJECTIVE:

To move a block (array) of data to another block to show how to employ indexed addressing to access data stored in the form of a continuous array in memory.

THEORY:

This addressing mode is widely used for accessing look-up tables from memory. A 16-bit base register (either DPTR or Program Counter) will hold the base of the table and the Accumulator will contain the table entry number. The address of the memory to be accessed is formed by adding the Accumulator data to the Base Pointer.

EXAMPLE:

Let the array start at 4501 and let the length of the array be in 4500. The contents of the array are as below.

(4500) = 06
(4501) = FF
(4502) = EF
(4503) = DF
(4504) = CF
(4505) = BF
(4506) = AF

The program should move the contents into an array starting at 4510. The result is,

(4510) = FF
(4511) = EF
(4512) = DF
(4513) = CF
(4514) = BF
(4515) = AF

PROGRAM:

```
        MOV     DPTR,#4500
        MOVX    A,@DPTR
        MOV     R0,A
        INC     DPTR
        MOV     R1,DPL
        MOV     R2,DPH
        MOV     DPTR,#4510
REPT:   PUSH    DPL
        PUSH    DPH
        MOV     DPL,R1
        MOV     DPH,R2
        MOVX    A,@DPTR
        INC     DPTR
        MOV     R1,DPL
        MOV     R2,DPH
        POP     DPH
        POP     DPL
        MOVX    @DPTR,A
        INC     DPTR
        DJNZ    R0,REPT
HERE:   SJMP    HERE
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	90	MOV DPTR,#4500
4101	45	
4102	00	
4103	E0	MOVX A,@DPTR
4104	F8	MOV R0,A
4105	A3	INC DPTR
4106	A9	MOV R1,DPL
4107	82	
4108	AA	MOV R2,DPH
4109	83	
410A	90	MOV DPTR, #4510

410B	45	
410C	10	
410D	C0	REPT: PUSH DPL
410E	82	
410F	C0	PUSH DPH
4110	83	
4111	89	MOV DPL, R1
4112	82	
4113	8A	MOV DPH,R2
4114	83	
4115	E0	MOVX A, @DPTR
4116	A3	INC DPTR
4117	A9	MOV R1, DPL
4118	82	
4119	AA	MOV R2, DPH
411A	83	
411B	D0	POP DPH
411C	83	
411D	D0	POP DPL
411E	82	
411F	F0	MOVX @DPTR, A
4120	A3	INC DPTR
4121	D8	DJNZ R0, REPT
4122	EA	
4123	80	HERE: SJMP HERE
4124	FE	

PROCEDURE:

i. Enter the opcodes from 4100 and the data from 4500.
ii. Execute the program.
iii. See that the data from 4501 to 4506 is moved to 4510.

DISCUSSION:

In this example, we have used indexed addressing for accessing an array of data using DPTR to hold the base address of the array. Here, MOVX @DPTR,A instruction is used as the data is accessed from a 16-bits wide external memory. If an 8-bit address is being used, MOVX @Ri,A instruction can be used for external memory access.

2.3 ARITHMETIC AND LOGICAL OPERATIONS:

Add, Subtract, Increment and Decrement operations, Multiply, Divide and Decimal Adjust instructions comprise the Arithmetic instructions in the instruction set of the 8051. The Rotate, And, Or, Exclusive Or, Complement, Swap and Clear instructions comprise the logical group. In the following examples, you will learn about most of these instructions.

2.3.1 PROGRAM 3 - 16-BIT ADDITION:

OBJECTIVE:

To perform 16-bit addition of two 16-bit data using immediate addressing and store the result in memory.

THEORY:

As there is only one 16-bit Register in 8051, 16-bit addition is performed by using ADDC instruction twice, i.e. adding LSD first and MSD next.

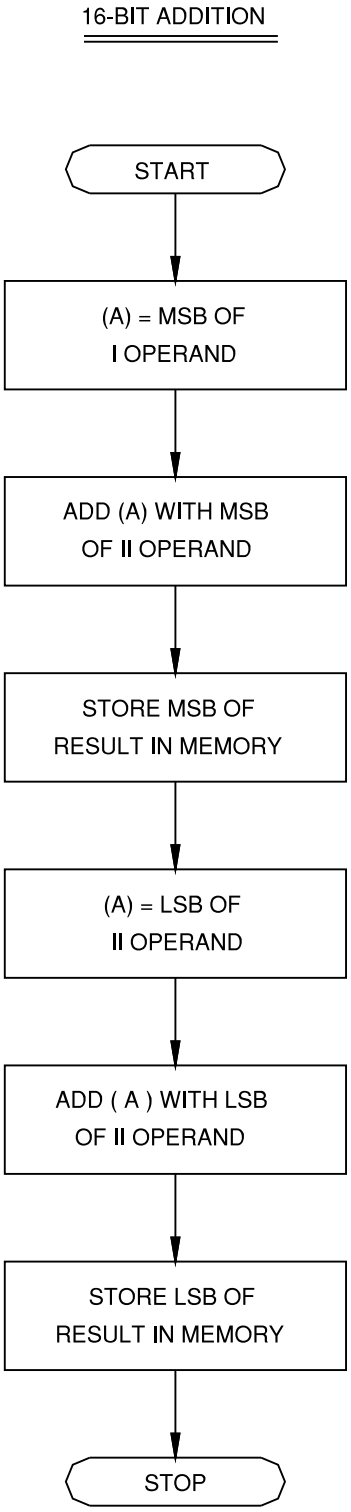
EXAMPLE:

The program is to add the 16-bit data 1234 with the data 5678 and store the result at the locations 4150 and 4151 using immediate addressing.

RESULT: [4150] = AC (LSB); [4151] = 68 (MSB).

DATAL1 = 34; DATAL2 = 78.
DATAM1 = 12; DATAM2 = 56.
DATAM1 - MSD OF DATA1,
DATAM2 - MSD OF DATA2,
DATAL1 - LSD OF DATA1,
DATAL2 - LSD OF DATA2.

FLOWCHART:


16-BIT ADDITION
START
(A) = MSB OF
I OPERAND
ADD (A) WITH MSB
OF II OPERAND
STORE MSB OF
RESULT IN MEMORY
(A) = LSB OF
II OPERAND
ADD (A) WITH LSB
OF II OPERAND
STORE LSB OF
RESULT IN MEMORY
STOP

PROGRAM:

```
        CLR     C
        MOV     A,#DATAL1
        ADDC    A,#DATAL2
        MOV     DPTR,#4150
        MOVX    @DPTR,A
        INC     DPTR
        MOV     A,#DATAM1
        ADDC    A,#DATAM2
        MOVX    @DPTR,A
HLT:    SJMP    HLT
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	C3	CLR C
4101	74	MOV A,#DATAL1
4102	34	
4103	24	ADD A,#DATAL2
4104	78	
4105	90	MOV DPTR,#4150
4106	41	
4107	50	
4108	F0	MOVX @DPTR,A
4109	A3	INC DPTR
410A	74	MOV A,#DATAM1
410B	12	
410C	34	ADDC A,#DATAM2
410D	56	
410E	F0	MOVX @DPTR,A
410F	80	HERE: SJMP HERE
4110	FE	

PROCEDURE:

i) Enter the above opcodes from 4100 in the trainer.

ii) Execute the program.

iii) Check for the result at 4150 and 4151.

iv) Change data and see if the result at 4150 changes accordingly.

DISCUSSION:

In this example, 16-bit addition has been directly performed. The user can write a program to perform 16-bit addition which can use any of the on-chip Register Banks. Also, in the above example, the data were loaded into the registers using immediate addressing mode. Try to do the same program by loading data from memory. Program for decimal addition can be written using 'DA A' in instruction of 8051.

EXERCISES:

i) Sample: [4500] = FF
[4501] = FF

Obtain the 16-bit sum of these two 8-bit data and store the result at memory locations 4510 and 4511.

Result: [4510] = 01 (MSB)
[4511] = FE (LSB)

ii) Sample: [4500] = FFFF
[4502] = FFFF

Obtain the sum and store the result starting from 4510.

Result: [4510] = 01 (MSB)
[4511] = FF
[4512] = FE (LSB)

2.3.2 PROGRAM 4 - 8-BIT SUBTRACTION:

OBJECTIVE:

To perform subtraction of two 8-bit data using immediate addressing and store the result in memory.

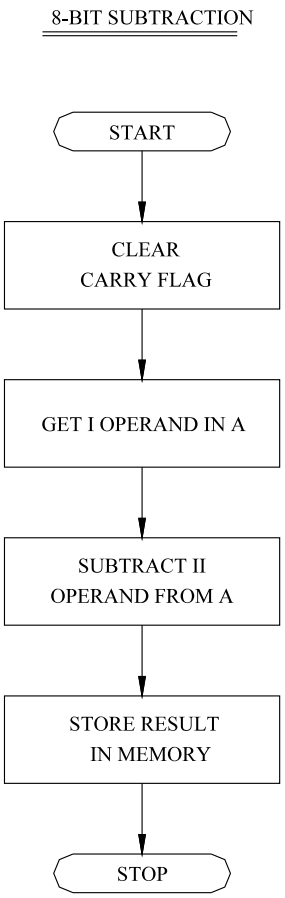
THEORY:

Using the accumulator, subtraction is performed and the result is stored. Immediate addressing is employed. The SUBB instruction writes the result in the accumulator.

EXAMPLE:

Sample data:	DATA1	=	20
	DATA2	=	10
Result:	[4500]	=	10

FLOW CHART:



PROGRAM:

```
        CLR     C
        MOV     A,#DATA1
        SUBB    A,#DATA2
        MOV     DPTR,#4500
        MOVX    @DPTR,A
HERE:   SJMP    HERE
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	C3	CLR C
4101	74	MOV A,#DATA1
4102	20	
4103	94	SUBB A,#DATA2
4104	10	
4105	90	MOV DPTR,#4500
4106	45	
4107	00	
4108	F0	MOVX @DPTR,A
4109	80	HERE: SJMP HERE
410A	FE	

PROCEDURE:

i. Enter the opcodes and data in the trainer.

ii. Execute the program and verify for results.

iii. Change data and see that correct results are obtained.

DISCUSSION:

In this example, we have used the SUBB instruction. The same can be performed using two's complement arithmetic using the CPL instruction and an ADD or INC instruction. A subtract is essentially a two's complement addition. Note the change in the carry flag during a borrow.

EXERCISES:

i. Subtract the contents of location 4500 from the contents of location 4501 and store the result at location 4600.

Sample data: [4500] = 56
[4501] = 6A

Result: [4600] = 14

ii. Perform the same subtraction using two's complement addition.

2.3.3 PROGRAM 5 - 8-BIT MULTIPLICATION:

OBJECTIVE:

To obtain the product of two 8-bit data using immediate addressing and store the result in memory.

THEORY:

The 8051 has a "MUL" instruction unlike many other 8-bit processors. MUL instruction multiplies the unsigned eight-bit integers in A and B. The low-order byte of the product is left in A and the high-order byte in B. If the product is > 255, the overflow flag is set. Otherwise it is cleared. The carry flag is always cleared.

EXAMPLE:

Let us multiply the contents of Registers A and B and store the 16-bit result at locations 4500 and 4501.

Sample data: DATA1 = 0A

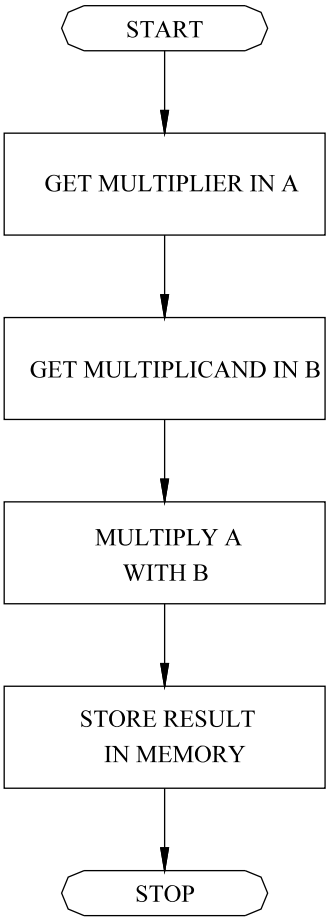
DATA2 = 88

[4500] = 50 (LSB)

[4501] = 05 (MSB)

FLOW CHART:

8-BIT MULTIPLICATION



PROGRAM:

```
        MOV     A,#DATA1
        MOV     B,#DATA2
        MUL     AB
        MOV     DPTR,#4500
        MOVX    @DPTR,A
        INC     DPTR
        MOV     A,B
        MOVX    @DPTR,A
HERE:   SJMP    HERE
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	74	MOV A,#DATA1
4101	0A	
4102	75	MOV B,#DATA2
4103	F0	
4104	88	
4105	A4	MUL AB
4106	90	MOV DPTR,#4500
4107	45	
4108	00	
4109	F0	MOVX @DPTR,A
410A	A3	INC DPTR
410B	E5	MOV A,B
410C	F0	
410D	F0	MOVX @DPTR,A
410E	80	HERE: SJMP HERE
410F	FE	

PROCEDURE:

i. Enter the above opcodes from 4100.

ii. Execute the program; see that the result is stored correctly.

iii. Change data and check if the results are correct each time.

DISCUSSION:

The MUL instruction and its usage have been demonstrated in this example. As mentioned earlier, the LSD of the product is in A and the MSD is in the B Register. Multiplication can also be performed by rotation because rotating a byte once towards left is equal to multiplying it by two.

EXERCISES:

i. Obtain the square of a number stored in memory.

Sample: [4500] = 0A

Result: [4600] = 64

ii. Obtain the fourth power of 08 using MUL instruction and store the result in memory.

Result: [4500] = 10 (MSB)
[4501] = 00 (LSB)

2.3.4 PROGRAM 6 - ONE'S AND TWO'S COMPLEMENT:

OBJECTIVE:

To obtain the one's and two's complement of an 8-bit number in Register A.

THEORY:

The one's complement of a number is obtained by inverting all the bits in that number , that is replacing all 1's by 0's and all 0's by 1's. The two's complement is the negative of that number.

EXAMPLE:

The number whose complements are needed is in register A. The result is in location 4200 and 4201.

Sample data: DATA = CC

Result: [4200] = 33 - One's Complement
[4201] = 34 - Two's Complement

PROGRAM:

```
        MOV     A,#DATA
        CPL     A
        MOV     DPTR,#4200
        MOVX    @DPTR,A
        INC     A
        INC     DPTR
        MOVX    @DPTR,A
HERE:   SJMP    HERE
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	74	MOV A,#DATA
4101	CC	
4102	F4	CPL A
4103	90	MOV DPTR,#4200
4104	42	
4105	00	
4106	F0	MOVX @DPTR,A
4107	04	INC A
4108	A3	INC DPTR
4109	F0	MOVX @DPTR,A
410A	80	HERE: SJMP HERE
410B	FE	

PROCEDURE:

i. Enter the opcodes and the data in the trainer.

ii. Execute the program and check for results.

iii Change data and check for the corresponding results.

DISCUSSION:

One's complement is nothing but the logical operation 'NOT'. In this example, the CPL instruction has been employed. Since the two's complement of a number is its one's complement + 1 , the INC instruction has been employed. It can also be performed by adding 1 to the one's complement number by using ADD instruction.

EXERCISE:

i) Obtain the one's and two's complement of the data 77 and store it in memory.

Result: One's complement [4500] = 88
Two's complement: [4501] = 89

2.3.5 PROGRAM 7 - SETTING BITS IN AN 8-BIT NUMBER:

OBJECTIVE:

To set specific bits of an 8-bit number.

THEORY:

Setting bits can be done by ORing that particular bit by 1. The following program explains how to set a particular bit in an 8-bit number by using the ORL instruction of 8051.

EXAMPLE:

In the following program, the contents of the Accumulator is ORed with an immediate data accordingly to set the required bits.

Sample Data : DATA1 = 2F
DATA2 = 45

Result : [4500] = 6F

PROGRAM:

```
        MOV     A,#DATA1
        ORL     A,#DATA2
        MOV     DPTR,#4500
        MOVX    @DPTR,A
HERE:   SJMP    HERE
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	74	MOV A,#DATA1
4101	2F	
4102	44	ORL A,#DATA2
4103	45	
4104	90	MOV DPTR,#4500
4105	45	
4106	00	
4107	F0	MOVX @DPTR,A
4108	80	HERE: SJMP HERE
4109	FE	

PROCEDURE:

i. Enter the opcodes and execute the program.
ii. Check whether the corresponding bits are set accordingly.

DISCUSSION:

The ORL instruction can be used to set a particular bit in the command or control registers of peripherals interfaced with the processor and can also be used to determine if the status read from peripherals is as expected.

EXERCISES:

i. Set alternate bits in an 8-bit number, starting from bit 0 and store the result at location 4200 in memory.
ii. Store successive powers of 2 from 0 to 7 in consecutive memory locations starting from 4300.

Result : [4300] = 01
[4301] = 02
[4302] = 04
[4303] = 08
[4304] = 10
[4305] = 20
[4306] = 40
[4307] = 80

2.3.6 PROGRAM 8 - MASKING BITS IN AN 8-BIT NUMBER:

OBJECTIVE:

To mask bits 0 and 7 of an 8-bit number and store the result in memory.

THEORY:

The ANL instruction of 8051 can be used to reset bits. ANDing with zero produces a cleared bit. ANDing with one does not change the status of the bit.

EXAMPLE:

To mask bits 0 and 7, the 8-bit data has to be ANDed with 7E, which is 01111110 in binary.

Sample data: DATA1 = 87
DATA2 = 7E
Result: [4500] = 06

PROGRAM:

```
        MOV     A,#DATA1
        ANL     A,#DATA2
        MOV     DPTR,#4500
        MOVX    @DPTR,A
HERE:   SJMP    HERE
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	74	MOV A,#DATA1
4101	87	
4102	54	ANL A,#DATA2
4103	7E	
4104	90	MOV DPTR,#4500
4105	45	
4106	00	
4107	F0	MOVX @DPTR,A
4108	80	HERE: SJMP HERE
4109	FE	

PROCEDURE:

i. Enter the opcodes from 4100 and execute the program.
ii. Check whether the result is 06 at 4500.

DISCUSSION:

The **ANL** instruction can be used to check whether a particular status is reached in the peripheral device just like the **ORL** instruction. The other logical instruction available in the instruction set of 8051 is the **XRL** [Exclusive-OR]. The **CLR** [Clear operand] instruction is also a logical instruction which can be used to initialise registers in counter operations.

EXERCISES:

i. Can the **XRL** instruction be used to clear an 8 bit register?

ii. Using **DA A** instruction, write a program for 8 bit **Decimal Addition.**

2.4 BIT ADDRESSABLE OPERATIONS:

One of the unique features of 8051 which makes it to serve as a powerful microcontroller in the real world applications is that it has got a number of bit addressable Special Function Registers which can be accessed using various Arithmetic, Logical, Data transfer and Boolean bit manipulations.

The details of various Bit addressable Special Function Registers and their addresses can be had from Chapter - 1 under the heading **"Registers of 8051."**

2.4.1 PROGRAM 9 - ARITHMETIC OPERATIONS:

OBJECTIVE:

To perform various Arithmetic Operations using Bit addressable Special Function Registers of 8051.

THEORY:

Specific bits of various bit addressable registers can be ANDed, ORed with Carry flag, CLeaRed, ComPLemented, SET and can be moved to Carry flag.

EXAMPLE:

In the following program, the contents of C(carry) is ANDed with ACC.7, ORed with IE.2. Also, the contents of SCON.5 is set and that of SCON.1 is cleared and the contents of SCON.1 is moved to the carry flag.

PROGRAM:

```
        MOV     A,#0FFH
        SETB    C
        ANL     C,ACC.7
        ORL     C,IE.2
        SETB    SCON.5
        CLR     SCON.1
        MOV     C,SCON.1
HLT:    SJMP    HLT
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	74	MOV A,#FF
4101	FF	
4102	D3	SETB C
4103	82	ANL C,ACC.7
4104	E0	
4105	72	ORL C,IE.2
4106	A8	

4107	D2	SETB SCON.5
4108	98	
4109	C2	CLR SCON.1
410A	98	
410B	A2	MOV C,SCON.1
410C	98	
410D	80	HLT: SJMP HLT
410E	FE	

PROCEDURE:

i. Enter the opcodes from 4100 and execute the program.

ii. Single step through the above program and check for register contents after every individual instruction.

DISCUSSION:

Only few instructions in bit manipulation are given in the Example Program. Try to utilise the maximum use of various Bit addressable instructions in 8051 instruction set for more efficiency. The Bit addressable registers can also be used with Branch instructions to check a specific condition which will be explained in the succeeding section under the name **'BRANCH INSTRUCTIONS'**.

EXERCISE:

i. Can the XRL instruction be used to XOR a Bit addressable register with carry flag?

2.5 BRANCH INSTRUCTIONS:

Programming requires conditional or unconditional transfer of instruction execution sequences. Depending upon the condition tested, the flags in the Program Status Word register are checked for the status and program execution address may change or a branch is performed. Instructions are also available in 8051 to test a particular bit of any bit addressable register whether it is set or not and change the program flow accordingly. Branch instructions in 8051 are listed below.

UNCONDITIONAL JUMPS:

SJMP	:	SHORT JUMP
LJMP	:	LONG JUMP
JMP		@A + DPTR : JUMP INDIRECT

CONDITIONAL JUMPS:

DJNZ	:	DECREMENT AND JUMP IF NOT ZERO
CJNE	:	COMPARE AND JUMP IF NOT EQUAL
JB	:	JUMP IF BIT SET
JBC	:	JUMP IF BIT IS SET AND CLEAR BIT
JC	:	JUMP IF CARRY IS SET
JNB	:	JUMP IF BIT NOT SET
JNC	:	JUMP IF CARRY NOT SET
JNZ	:	JUMP IF ACCUMULATOR NOT ZERO
JZ	:	JUMP IF ACCUMULATOR ZERO

In the unconditional jump instructions, it may be either long or short. If it is a long branch, it is prefixed by an L. Long branch instruction LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC with the second and third instruction bytes respectively. The destination may therefore be anywhere in the full 64K Program memory address space.

The conditional jumps are all relative. The unconditional jump instruction JMP @A+DPTR is a special instruction used for indirect jump whose branch address is calculated by adding the accumulator's 8-bit data with the contents of DPTR. Neither Accumulator nor DPTR is affected.

The following programs shall employ any of these instructions and transform your programming ability to higher levels.

2.5.1 PROGRAM 10 - SUM OF THE ELEMENTS IN AN ARRAY:

OBJECTIVE:

To add the numbers of an 8-bit array, the array length being the first element in the array and store the result in memory.

THEORY:

The numbers are stored in consecutive locations in memory. Addition has to be performed for the number of times decided by the length of the array. Since the sum may exceed 8-bits, it is necessary to monitor the carry flag status. Indexed addressing employing the 16-bit Base Register DPTR is used in this program.

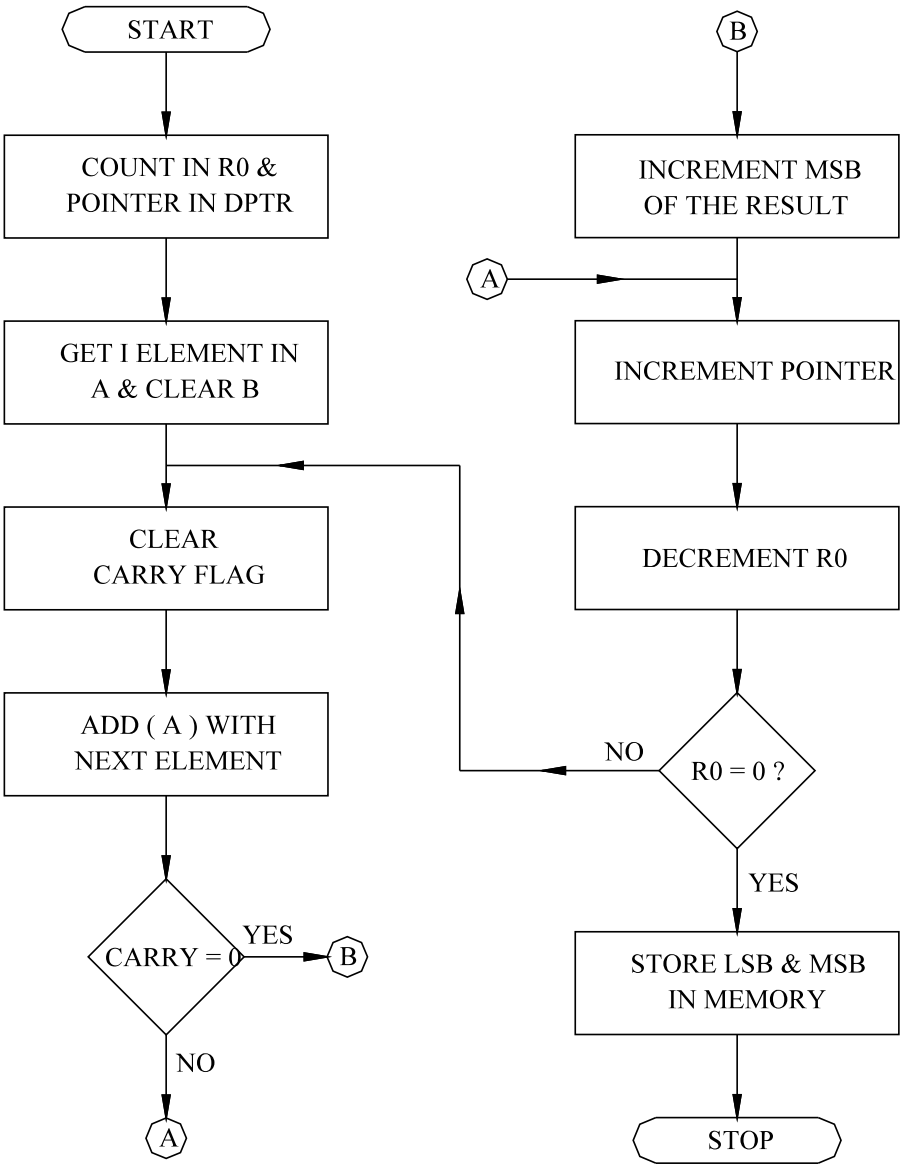
EXAMPLE:

The array starts at 4201. The address location 4200 contains the number of elements in that array. The result is to be stored at 4500 and 4501.

Sample Data:	[4200]	=	03
	[4201]	=	01
	[4202]	=	02
	[4203]	=	03
Result:	[4500]	=	00 (MSB)
	[4501]	=	06 (LSB)

FLOW CHART:

SUM OF THE ELEMENTS IN AN ARRAY



PROGRAM:

```
        MOV     DPTR,#4200      ;DPTR points to base of array
        MOVX    A,@DPTR
        MOV     R0,A            ;R0 has the length of array
        MOV     B,#0
        MOV     R1,B
ADD:    CLR     C
        INC     DPTR            ;Increment the Pointer
        MOVX    A,@DPTR         ;Get the element
        ADD     A,B             ;Do the partial addition
        MOV     B,A
        JNC     NC              ;Check for carry
        INC     R1              ;If carry set, then inc. MSB
NC:     DJNZ    R0,ADD          ;Repeat it till count = 0
        MOV     DPTR,#4500
        MOV     A,R1            ;Store the MSB first
        MOVX    @DPTR,A
        INC     DPTR
        MOV     A,B             ;Store the LSB at 4501
        MOVX    @DPTR,A
HLT:    SJMP    HLT
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	90	MOV DPTR,#4200
4101	42	
4102	00	
4103	E0	MOVX A,@DPTR
4104	F8	MOV R0,A
4105	75	MOV B,#0
4106	F0	
4107	00	
4108	A9	MOV R1,B
4109	F0	
410A	C3	ADD: CLR C
41OB	A3	INC DPTR
410C	E0	MOVX A,@DPTR
410D	25	ADD A,B
410E	F0	
410F	F5	MOV B,A
4110	F0	
4111	50	JNC NC
4112	01	
4113	09	INC R1
4114	D8	NC: DJNZ R0, ADD
4115	F4	
4116	90	MOV DPTR,#4500
4117	45	
4118	00	
4119	E9	MOV A,R1

411A	F0	MOVX @DPTR,A
411B	A3	INC DPTR
411C	E5	MOV A,B
411D	F0	
411E	F0	MOVX @DPTR,A
411F	80	HLT : SJMP HLT
4120	FE	

PROCEDURE:

i. Enter the above opcodes and data from 4100 and 4200 respectively.

ii. Execute the program.

iii. Check for result at 4500 and 4501.

iv. Change data at 4200 and check for results.

DISCUSSION:

Computation of the sum of the elements of an array employs the algorithm which is usually used to compute checksums in data storage. The calculated checksum while retrieving data is compared with the stored checksum to detect discrepancies if any. This would use the XRL instruction. Branch instructions SJMP, JNC and DJNZ have been employed in the above example.

EXERCISES:

i. Modify the program to employ JC.

ii. Modify the program to add 16-bit numbers stored in an array.

iii Add the two numbers 12345678 and 9ABCDEF0 using array addition.

Result = ACF13568.

2.5.2 PROGRAM 11 - MULTIPRECISION ADDITION (24-BIT):

OBJECTIVE:

To add two numbers that are more than 8-bit in length, the length being given as the number of 8-bits in that number.

THEORY:

The multi-byte addition program adds only in sets of 8-bits. The LSD of the two numbers are added first. Now, the next set of 8-bits is added, taking into consideration the status of carry due to the previous addition. Addition is done till the length of the number specified becomes zero.

EXAMPLE:

The length of both the numbers is 3 bytes. The length is loaded in R4. The first data is stored from 4200 with the LSB taking the first position. The second number starts at 4210.

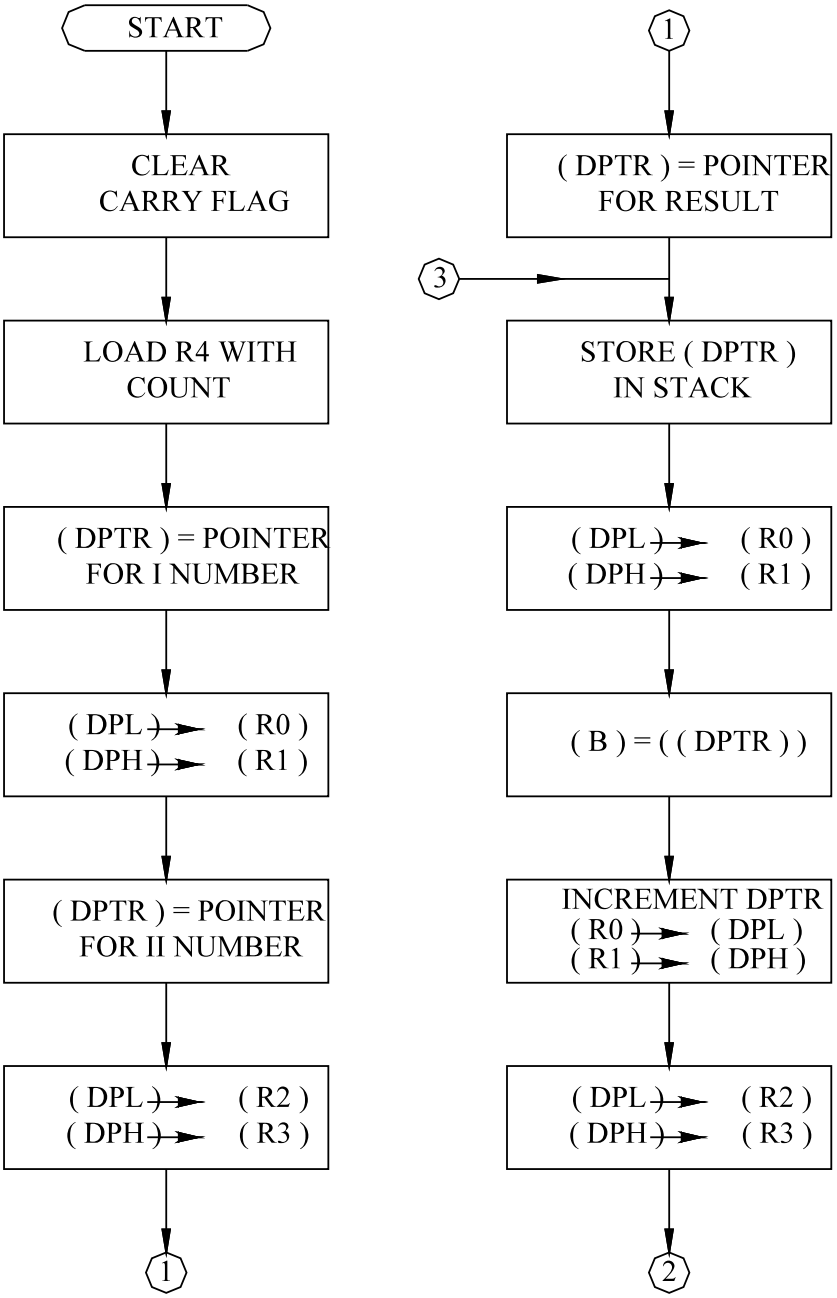
Data: [4200] = 29 - First number
[4201] = A4
[4202] = 02
[4210] = FB - Second number
[4211] = 37
[4212] = 28

```
02 A4 29 (+)
28 37 FB
---------
2A DC 24
---------
```

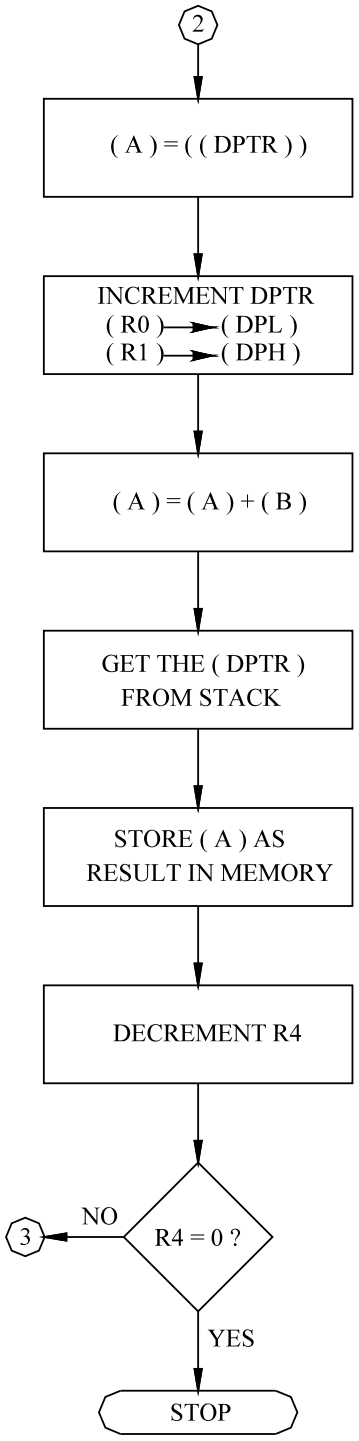
Result: [4500] = 24
[4501] = DC
[4502] = 2A

FLOW CHART:

MULTIPRECISION ADDITION (24 - BITS)



FLOW CHART:


2
(A) = ((DPTR))
INCREMENT DPTR
(R0) ⟶ (DPL)
(R1) ⟶ (DPH)
(A) = (A) + (B)
GET THE (DPTR)
FROM STACK
STORE (A) AS
RESULT IN MEMORY
DECREMENT R4
NO
3
R4 = 0 ?
YES
STOP

PROGRAM:

```
        CLR     C               ;Carry flag = 0
        MOV     R4,#3           ;Count
        MOV     DPTR,#4200      ;Pointer for first no.
        MOV     R0,DPL
        MOV     R1,DPH
        MOV     DPTR,#4210      ;Pointer for second no.
        MOV     R2,DPL
        MOV     R3,DPH
        MOV     DPTR,#4500      ;Pointer to store result
REPT:   PUSH    DPL
        PUSH    DPH
        MOV     DPL,R0
        MOV     DPH,R1
        MOVX    A,@DPTR
        MOV     B,A
        INC     DPTR
        MOV     R0,DPL
        MOV     R1,DPH
        MOV     DPL,R2
        MOV     DPH,R3
        MOVX    A,@DPTR
        ADDC    A,B
        INC     DPTR
        MOV     R2,DPL
        MOV     R3,DPH
        POP     DPH
        POP     DPL
        MOVX    @DPTR,A
        INC     DPTR
        DJNZ    R4,REPT
HLT:    SJMP    HLT
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	C3	CLR C
4101	7C	MOV R4,#3
4102	03	
4103	90	MOV DPTR,#4200
4104	42	

4105	00	
4106	A8	MOV R0,DPL
4107	82	
4108	A9	MOV R1,DPH
4109	83	
410A	90	MOV DPTR,#4210
410B	42	
410C	10	
410D	AA	MOV R2,DPL
410E	82	
410F	AB	MOV R3,DPH
4110	83	
4111	90	MOV DPTR,#4500
4112	45	
4113	00	
4114	C0	REPT: PUSH DPL
4115	82	
4116	C0	PUSH DPH
4117	83	
4118	88	MOV DPL,R0
4119	82	
411A	89	MOV DPH,R1
411B	83	
411C	E0	MOVX A,@DPTR
411D	F5	MOV B,A
411E	F0	
411F	A3	INC DPTR
4120	A8	MOV R0,DPL
4121	82	

4122	A9	MOV R1,DPH
4123	83	
4124	8A	MOV DPL,R2
4125	82	
4126	8B	MOV DPH,R3
4127	83	
4128	E0	MOVX A,@DPTR
4129	35	ADDC A,B
412A	F0	
412B	A3	INC DPTR
412C	AA	MOV R2,DPL
412D	82	
412E	AB	MOV R3,DPH
412F	83	
4130	D0	POP DPH
4131	83	
4132	D0	POP DPL
4133	82	
4134	F0	MOVX @DPTR,A
4135	A3	INC DPTR
4136	DC	DJNZ R4,REPT
4137	DC	
4138	80	HLT: SJMP HLT
4139	FE	

PROCEDURE:

i. Enter the opcodes and data into the trainer.
ii. Execute the program.
iii. Check for results at 4500.
iv. Change data at 4200 and 4210 and check for appropriate results from 4500.

DISCUSSION:

The multi-byte addition and subtraction programs provide much help during long arithmetic table operations. Take a careful look at the logic and try 32-bit, 48-bit additions and subtractions.

EXERCISES:

i. Subtract two multiple - word numbers. The length of the array is given at memory location 4550 the array itself starts from 4551 and 4560 respectively. The result replaces the number starting from memory location 4551. Subtract the number string in 4560 from the one starting at 4551.

Sample problem:

Data : [4550] = 04 - Count.
[4551] = C3 - First number.
[4552] = A7
[4553] = 5B
[4554] = 2F
[4560] = B8 - Second number.
[4561] = 35
[4562] = DF
[4563] = 14

Result: [4551] = 0B
[4552] = 72
[4553] = 7C
[4554] = 1A

```
  2F  5B  A7  C3
  14  DF  35  B8  (-)
 ----------------
  1A  7C  72  0B
 ----------------
```

ii. Add the four numbers CA, 43, 56 and 19 and store the result at memory locations 4550 and 4551.

Result : [4550] = 7C
[4551] = 01

CA + 43 + 56 + 19 = 017C

iii. Do a decimal multi-byte addition in 32-bit and store the result in memory.

Data	:	[4550]	= 04	-	Count
		[4551]	= 99	-	First number
		[4552]	= 99		
		[4553]	= 99		
		[4554]	= 99		
		[4561]	= 99	-	Second number
		[4562]	= 99		
		[4563]	= 99		
		[4564]	= 99		
Result	:	[4570]	= 98		
		[4571]	= 99		
		[4572]	= 99		
		[4573]	= 99		
		[4574]	= 01		

2.5.3 PROGRAM 12 - 8-BIT DIVISION:

OBJECTIVE:

To divide an 8-bit number by another 8-bit number and store the quotient and remainder in memory.

THEORY:

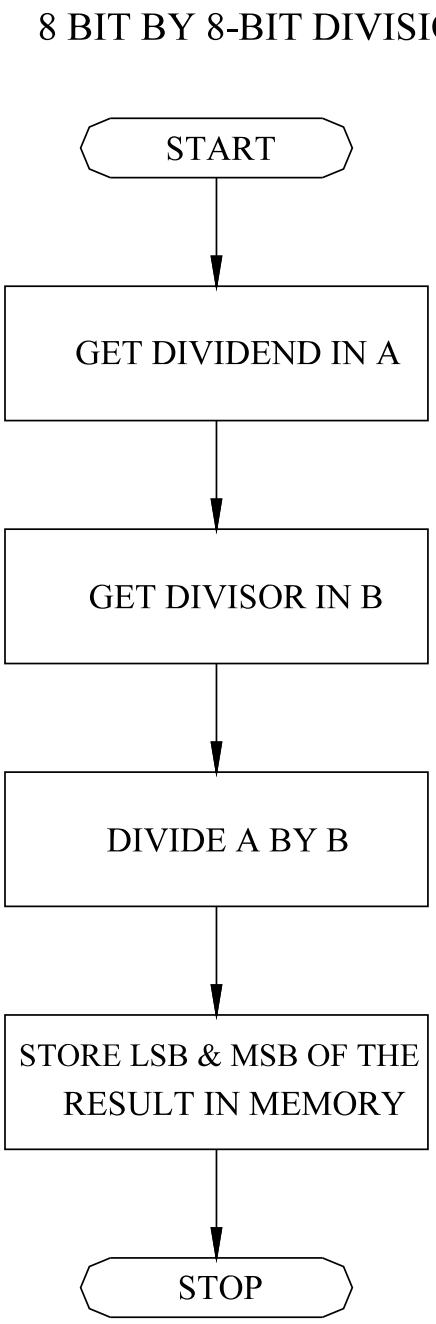
The 8051 has a "DIV" instruction unlike many other 8-bit processors. DIV instruction divides the unsigned eight-bit integer in A by the unsigned 8-bit integer in register B. The Accumulator receives the integer part of the quotient and register B receives the integer remainder. The carry and OV flags will be cleared.

EXAMPLE:

Let the divisor and dividend be in registers B and A respectively.

Data	:	DATA1	= 65	-	Dividend
		DATA2	= 08	-	Divisor
Result	:	[4500]	= 0C	-	Quotient
		[4501]	= 05	-	Remainder

FLOW CHART:


8 BIT BY 8-BIT DIVISION
START
GET DIVIDEND IN A
GET DIVISOR IN B
DIVIDE A BY B
STORE LSB & MSB OF THE RESULT IN MEMORY
STOP

PROGRAM:

```
        MOV     A,#DATA1        ;Load Acc. with Dividend.
        MOV     B,#DATA2        ;Load Reg. B with Divisor.
        DIV     AB
        MOV     DPTR,#4500
        MOVX    @DPTR,A         ;Store quotient at 4500
        INC     DPTR
        MOV     A,B             ;Store remainder at 4501
        MOVX    @DPTR,A
HLT:    SJMP    HLT
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	74	MOV A,#DATA1
4101	65	
4102	75	MOV B,#DATA2
4103	F0	
4104	08	
4105	84	DIV AB
4106	90	MOV DPTR,#4500
4107	45	
4108	00	
4109	F0	MOVX @DPTR,A
410A	A3	INC DPTR
410B	E5	MOV A,B
410C	F0	
410D	F0	MOVX @DPTR,A
410E	80	HERE: SJMP HERE
410F	FE	

PROCEDURE:

i. Enter the opcodes and data.

ii. Execute the program.

iii. Check for results at 4500 and 4501.

iv. Change data and verify the results.

DISCUSSION:

Though DIV instruction is available in the 8051 instruction set, division can also be performed by using the rotate instructions. Dividing by 8 can be performed by rotating the contents of Accumulator 3 times towards right because rotating a byte once towards right is equal to dividing it by two.

EXERCISES:

i Divide the 16-bit unsigned number at memory locations 4550 and 4551 by the 8-bit unsigned number at memory location 4552 and store quotient and remainder starting from 4553.

Sample problem:

Data : [4550] = 63
[4552] = 21.
[4551] = 52;

Result : [4553] = 03 Quotient
[4554] = 02
[4555] = 10 Remainder

ii. Divide 14 by 04 using Rotate Right instruction and store the result at 4550.

[Hint: Rotating a number right once is equivalent to dividing it by 2]

Sample problem:

Data : [4500] = 15 (Dividend)
[4501] = 02 (Divisor)
Result : [4550] = 0A - Quotient
[4551] = 01 - Remainder

2.6 CODE CONVERSION:

The basic instructions learnt so far include Data transfer, Bit addressable, Arithmetic and logical instructions. This section on code conversion makes use of any or all of these instructions.

Code conversion is very essential in micro computing because many peripherals that are to be dealt which may provide data in ASCII, BCD or various special codes. Each one of the special codes which is different from the binary concept of 8051 must be converted first to binary, for the microprocessor to process it. After processing, the binary form must again be converted back to the special code of that peripheral, before transferring it to the peripheral.

In the following examples, you will learn about conversion between different codes, using certain algorithms. These algorithms can be changed if desired.

2.6.1 PROGRAM 13 - ASCII TO DECIMAL CONVERSION:

OBJECTIVE:

To convert the ASCII number in the accumulator to its equivalent decimal number.

THEORY:

Conversion of an ASCII number to decimal is very simple because all the decimal numbers form a sequence in ASCII. Any ASCII number can be converted to decimal just by subtracting 30 from it.

ASCII Number(Hex)	Decimal Equivalent
30	00
31	01
32	02
33	03
34	04
35	05
36	06
37	07
38	08
39	09

EXAMPLE:

Let the ASCII number be in Register A and the result be stored at 4500.

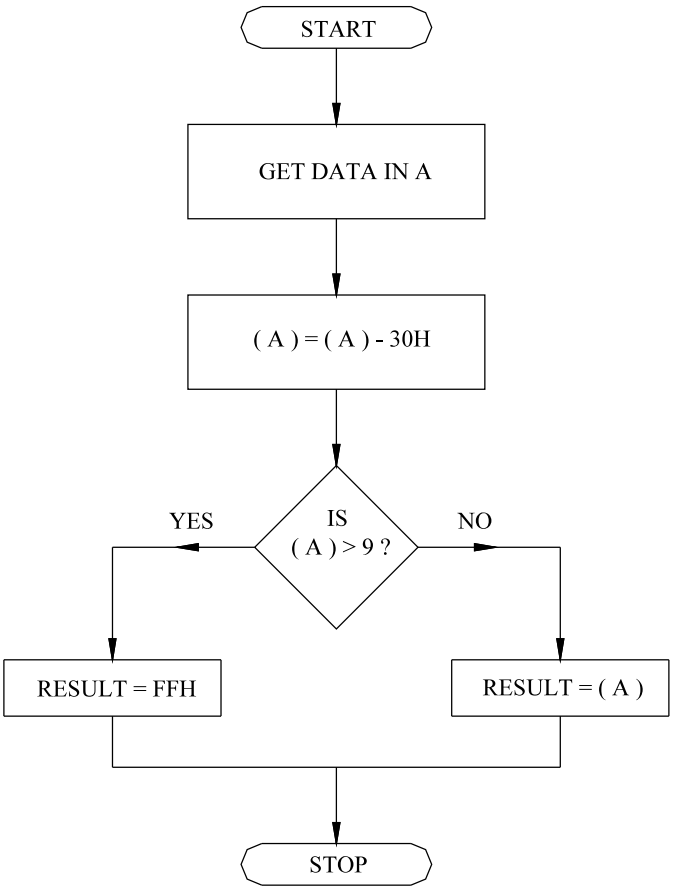
Data : DATA = 35
Result : [4500] = 05

Data : DATA = 3B
Result : [4500] = FF

If the decimal number got is not a valid one, FF will be stored as the result to indicate the error.

FLOW CHART:

ASCII TO DECIMAL CONVERSION



PROGRAM:

```
        MOV     DPTR,#4500
        MOV     A,#DATA     ;Get ASCII number
        CLR     C           ;Clear carry flag
        SUBB    A,#30       ;Subtract [A] by 30
        CLR     C
        SUBB    A,#0A       ;Check if the no. is decimal
        JC      STR
        MOV     A,#0FF      ;Else store FF at 4500
        SJMP    L1
STR:    ADD     A,#0A
L1:     MOVX    @DPTR,A     ;Store the result if valid
HLT:    SJMP    HLT
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	90	MOV DPTR,#4500
4101	45	
4102	00	
4103	74	MOV A,#DATA
4104	35	
4105	C3	CLR C
4106	94	SUBB A,#3
4107	30	
4108	C3	CLR C
4109	94	SUBB A,#0A
410A	0A	
410B	40	JC STR
410C	04	
410D	74	MOV A,#0FF
410E	FF	
410F	80	SJMP L1
4110	02	

4111	24	STR: ADD A,#0A
4112	0A	
4113	F0	L1: MOVX @DPTR,A
4114	80	HLT: SJMP HLT
4115	FE	

DISCUSSION:

In this program, the basic idea is to subtract ASCII "0" for all ASCII digits. This conversion is perfect only for decimal digits. There is a break between ASCII "9" and ASCII "A" which must be taken into account during ASCII to Hex conversions or vice versa. The ASCII is very useful to display data on CRT displays or to print on an ASCII Printer.

EXERCISES:

i) Convert the decimal number at 4550 to its equivalent ASCII and store it at 4551. Store FF if the input data is not a valid ASCII.

Sample problems:

Data : [4550] = 03
Result : [4551] = 33
Data : [4550] = 10
Result : [4551] = FF - (Error)

ii. Convert the hex number at 4550 to its equivalent ASCII at 4552.

Sample problems:

Data : [4550] = 0F
Result : [4552] = 46
Data : [4550] = 09
Result : [4552] = 39
Data : [4550] = 10
Result : [4552] = FF - (Error)

iii. Do ASCII to hex conversion of data at 4550 and store result at 4551.

Sample Problems:

Data : [4550] = 38
Result : [4551] = 08
Data : [4550] = 41
Result : [4551] = 0A
Data : [4550] = 3C
Result : [4551] = FF (Error)

2.6.2 PROGRAM 14 - WORD DISASSEMBLY:

OBJECTIVE:

Split the contents of 4500 into two nibbles (4-bit numbers) and store the HN at 4501 and the LN at 4502.

THEORY:

The data is fetched from memory and SWAP instruction of 8051 is used which will interchange the low and high-order nibbles of the Accumulator. The HN of the result is masked off and is stored in memory. Then, the HN in the original data is masked off and is stored in memory.

EXAMPLE:

The contents of 4500 is 56. The result stored at 4501 is 05 and at 4502 is 06.

Data : [4500] = 56
Result : [4501] = 05
[4502] = 06

PROGRAM:

```
        MOV     DPTR,#4500
        MOVX    A,@DPTR     ;Get the data from memory
        MOV     B,A         ;B has the original data
        SWAP    A
        ANL     A,#0F
        INC     DPTR
        MOVX    @DPTR,A     ;Store HN
        MOV     A,B         ;Get original data
        ANL     A,#0F       ;Mask HN of original data
        INC     DPTR
        MOVX    @DPTR,A     ;Store LN
HLT:    SJMP    HLT
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	90	MOV DPTR,#4500
4101	45	
4102	00	
4103	E0	MOVX A,@DPTR
4104	F5	MOV B,A
4105	F0	
4106	C4	SWAP A
4107	54	ANL A,#0F
4108	0F	
4109	A3	INC DPTR
410A	F0	MOVX @DPTR,A
410B	E5	MOV A,B
410C	F0	
410D	54	ANL A,#0F
410E	0F	
410F	A3	INC DPTR
4110	F0	MOVX @DPTR,A
4111	80	HLT: SJMP HLT
4112	FE	

PROCEDURE:

i. Key in the opcodes from the address specified.

ii. Enter data at 4500.

iii. Execute the program and check for results.

iv. Change data and try again.

DISCUSSION:

Word disassembly has been included under code conversion since it is another nomenclature for packed to unpacked decimal conversion. This is used in many I/O interfacing techniques (displays).

EXERCISES:

i) Do a word assembly of data at locations 4550 and 4551 and store the result at 4560.

Sample Problems:

Data : [4550] = 07
[4551] = 09
Result : [4560] = 79
Data : [4550] = 09
[4551] = 09
Result : [4560] = 99

2.6.3 PROGRAM 15 - HEX TO DECIMAL CONVERSION:

OBJECTIVE:

To obtain the decimal equivalent of an 8-bit hex number stored in memory.

THEORY:

In this program, the hex number is converted to its equivalent decimal number. The algorithm followed is very simple. The hex number to be converted is brought to the accumulator and is divided by 100 D to find the number of hundreds in it. DIV instruction of 8051 is used in this program. The remainder is now divided by 10 D to count the number of tens in it. Finally, the remainder obtained from the above division gives the number of units in the given hex number. The result is stored in memory in the unpacked form.

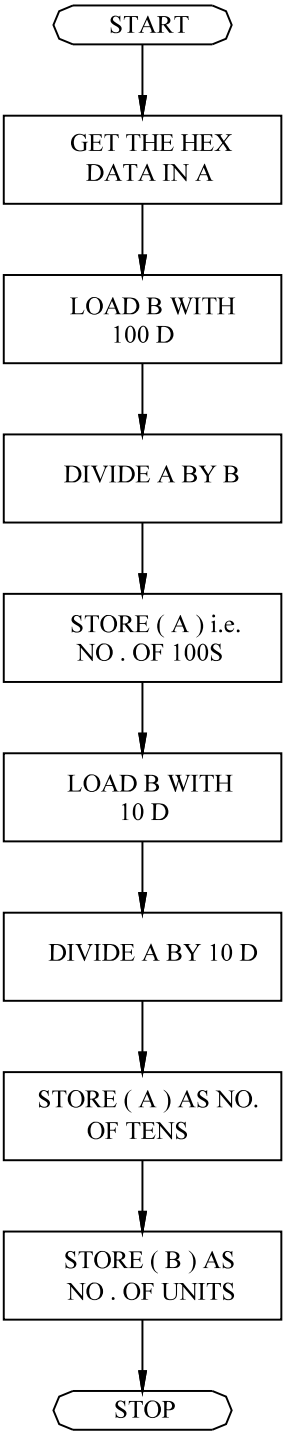
EXAMPLE:

Let us work with FF.

Data : [4500] = FF
Result : [4501] = 02
[4502] = 05
[4503] = 05

FLOW CHART:

HEX TO DECIMAL CONVERSION



PROGRAM:

```
        MOV     DPTR,#4500
        MOVX    A,@DPTR     ;Get the data
        MOV     B,#64
        DIV     AB          ;Find no. of 100s
        MOV     DPTR,#4501
        MOVX    @DPTR,A
        MOV     A,B
        MOV     B,#0A
        DIV     AB          ;Find no. of 10s
        INC     DPTR
        MOVX    @DPTR,A
        INC     DPTR
        MOV     A,B
        MOVX    @DPTR,A
HLT:    SJMP    HLT
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	90	MOV DPTR,#4500
4101	45	
4102	00	
4103	E0	MOVX A,@DPTR
4104	75	MOV B,#64
4105	F0	
4106	64	
4107	84	DIV AB
4108	90	MOV DPTR,#4501
4109	45	
410A	01	
410B	F0	MOVX @DPTR,A
410C	E5	MOV A,B
410D	F0	
410E	75	MOV B,#0A

410F	F0	
4110	0A	
4111	84	DIV AB
4112	A3	INC DPTR
4113	F0	MOVX @DPTR,A
4114	A3	INC DPTR
4115	E5	MOV A,B
4116	F0	
4117	F0	MOVX @DPTR,A
4118	80	HLT: SJMP HLT
4119	FE	

PROCEDURE:

i. Enter the opcodes from the specified address.

ii. Enter the data.

iii. Execute the program and check for results.

iv. Change data and see if exact results are stored.

DISCUSSION:

Code conversions are very important to extend the operations done with the trainer to the external world. In the above program, the result has been stored in unpacked BCD form. Try to write a program which will store the result in packed BCD format. Use of a previous example in this section may be resorted to.

EXERCISE:

i. Convert a 16-bit hex number at location 4500 and 4501 to its equivalent decimal number and store the result from 4502 onwards.

Sample Problem:

Data : [4500] = FF

[4501] = FF

Result :[4502] = 06

[4503] = 05

[4504] = 05

[4505] = 03

[4506] = 06

2.6.4 PROGRAM 16 - DECIMAL TO HEX CONVERSION:

OBJECTIVE:

To convert BCD digits in memory to the equivalent hex number.

THEORY:

Considering that out of the two unpacked BCD digits at 4200 and 4201, the digit at 4200 is the MSD, the logic is to multiply this by 0A (10D) and then add the LSD at 4201 to the product.

EXAMPLE:

The digits are at 4200 and 4201 and let the result be stored at 4202.

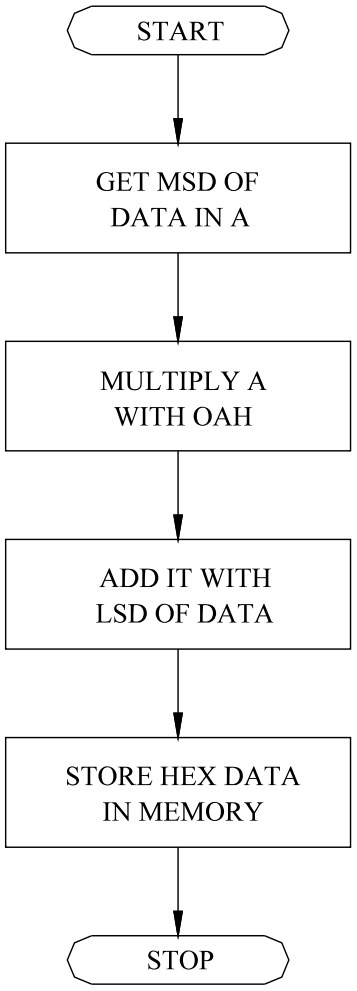
Data : [4200] = 03

[4201] = 06

Result : [4202] = 24

FLOW CHART:

DECIMAL TO HEX CONVERSION



PROGRAM:

```
        MOV     DPTR,#4200
        MOVX    A,@DPTR
        MOV     B,#0AH
        MUL     AB
        MOV     B,A
        INC     DPTR
        MOVX    A,@DPTR
        ADD     A,B
        INC     DPTR
        MOVX    @DPTR,A
HLT:    SJMP    HLT
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	90	MOV DPTR,#4200
4101	42	
4102	00	
4103	E0	MOVX A,@DPTR
4104	75	MOV B,#0A
4105	F0	
4106	0A	
4107	A4	MUL AB
4108	F5	MOV B,A
4109	F0	
410A	A3	INC DPTR
410B	E0	MOVX A,@DPTR
410C	25	ADD A,B
410D	F0	
410E	A3	INC DPTR
410F	F0	MOVX @DPTR,A
4110	80	HLT: SJMP HLT
4111	FE	

PROCEDURE:

i. Key in the opcodes from 4100.

ii. Enter the BCD data at 4200 and 4201.

iii. Execute the program.

iv. Check for result at 4202.

v. Change data and verify for proper results.

DISCUSSION:

BCD entries are converted to Hex in order to save on storage and calculations. BCD numbers require additional memory space and more complex calculations. However, this conversion may offset some of the advantages of binary arithmetic. The vice versa of this conversion is also very important if it is to be used with a peripheral device.

EXERCISES:

i. Convert the BCD digits 02 and 06 into a packed BCD number. Convert the hex number thus obtained into decimal and store the decimal number at 4250.

Result : [4250] = 38.

ii. Convert the BCD number 255 into the hex equivalent and store result in memory.

Result : [4250] = FF.

2.7 ARRAY OPERATIONS:

As the name implies, the examples given here deal with arrays of data. An array is a collection of data stored in consecutive memory locations, to be accessed by the program for manipulation of array. Finding the biggest number, sorting techniques, look-up table block operations all come under this array operation.

2.7.1 PROGRAM 17 - LARGEST ELEMENT IN AN ARRAY:

OBJECTIVE:

To find the biggest number in an array of 8-bit unsigned numbers of predetermined length.

THEORY:

To find the Biggest number in any given array, the contents of the array must be compared with an arbitrary Biggest number. In this experiment, since all numbers are said to be unsigned 8-bit numbers, let Internal memory location (say 40H) has the biggest number i.e. zero. Now compare the first number with Internal memory location. If it is greater, move it to Internal memory location. Further comparison is with this biggest number and this comparison is done till the end of the array. Now the biggest number in Internal memory location is stored in memory as the result.

EXAMPLE:

The length of the array is specified in the Register R5. The array itself starts at 4200. The largest number of the array is stored at location 420A.

Data : [4200] = 05

[4201] = 67

[4202] = 76

[4203] = 89

[4204] = 98

[4205] = 49

[4206] = 45

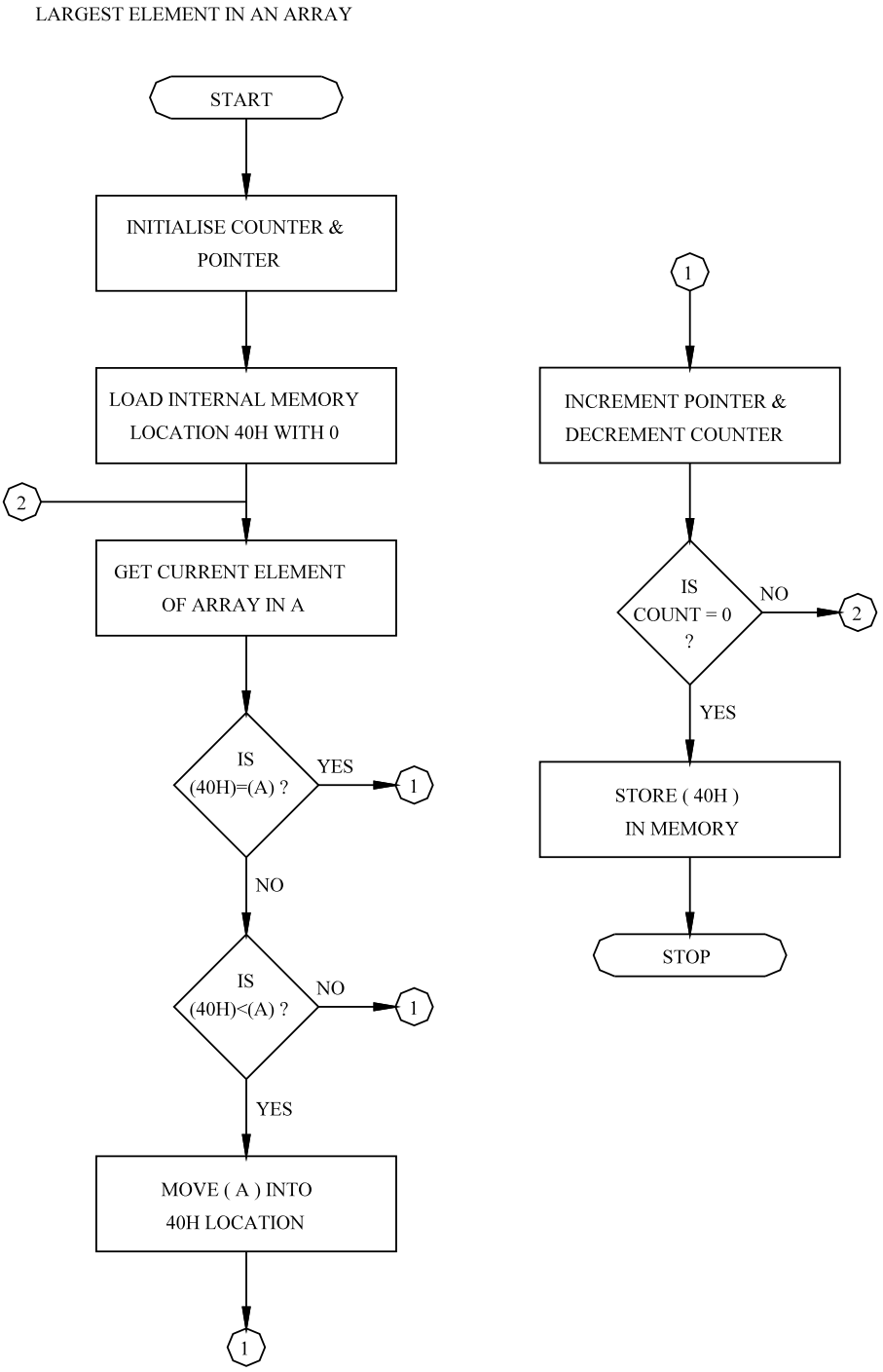
[4207] = 9F

[4208] = 57

[4209] = 7A

Result :[420A] = 9F

FLOW CHART:


LARGEST ELEMENT IN AN ARRAY
START
INITIALISE COUNTER & POINTER
LOAD INTERNAL MEMORY LOCATION 40H WITH 0
2
GET CURRENT ELEMENT OF ARRAY IN A
IS (40H)=(A) ?
YES
1
NO
IS (40H)<(A) ?
NO
1
YES
MOVE (A) INTO 40H LOCATION
1
1
INCREMENT POINTER & DECREMENT COUNTER
IS COUNT = 0 ?
NO
2
YES
STORE (40H) IN MEMORY
STOP

PROGRAM:

```
        MOV     DPTR,#4200
        MOV     40H,#00
        MOV     R5,#0AH
LOOP2:  MOVX    A,@DPTR
        CJNE    A,40H,LOOP1
LOOP3:  INC     DPTR
        DJNZ    R5,LOOP2
        MOV     A,40H
        MOVX    @DPTR,A
HLT:    SJMP    HLT
LOOP1:  JC      LOOP3
        MOV     40H,A
        SJMP    LOOP3
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	90	MOV DPTR,#4200
4101	42	
4102	00	
4103	75	MOV 40H,#00
4104	40	
4105	00	
4106	7D	MOV R5,#0AH
4107	0A	
4108	E0	LOOP2: MOVX A,@DPTR
4109	B5	CJNE A,40H,LOOP1
410A	40	
410B	08	
410C	A3	LOOP3: INC DPTR
410D	DD	DJNZ R5, LOOP2
410E	F9	
410F	E5	MOV A,40H

4110	40	
4111	F0	MOVX @DPTR,A
4112	80	HLT: SJMP HLT
4113	FE	
4114	40	LOOP1: JC LOOP3
4115	F6	
4116	F5	MOV 40H,A
4117	40	
4118	80	SJMP LOOP3
4119	F2	

PROCEDURE:

i. Enter the opcodes starting from 4100.

ii. Key in the data at 4200.

iii Execute the program and verify for result at 420A.

iv. Change the array length and data and repeat.

DISCUSSION:

In the above program, the Internal memory location 40H is initialised to zero for the first comparison. This is because the numbers are unsigned and hence after the first comparison, the first number will come to Internal memory location. This number is then used for further comparison. A new algorithm of doing the same program can also be thought of. The algorithm must be changed if the numbers are signed.

EXERCISES:

i. Find the smallest of two numbers 39 and 65 and store result at 4250.

Result : [4250] = 39

ii. Find the smallest number in an array of four elements residing from 4250 and store result at 4260.

Sample problem:

Data : [4250] = 04 - Count
[4251] = 56
[4252] = 73
[4253] = 13
[4254] = E5
Result : [4260] = 13

iii. Find the biggest and smallest number of an array in a single program; the length of the array is at memory location 4250 and the array itself starts at 4251. Store the biggest number at 4260 and smallest number at 4261.

Sample problem:

Data : [4250] = 05 - Length of the array.
[4251] = 85 - Array starts.
[4252] = 19
[4253] = 70
[4254] = 36
[4255] = 59
Result: [4260] = 85 - Biggest number.
[4261] = 19 - Smallest number.

iv. In an array of five signed numbers, find the smallest number. The numbers themselves are at memory from 4250 and the result is to be stored at 4260.

Sample problem:

Data : -77, +60, -9C, -2D, +40
Result : -9C

2.7.2 PROGRAM 18 - ASCENDING ORDER OF AN ARRAY:

OBJECTIVE:

To arrange an array of 8-bit unsigned numbers of known length in ascending order.

THEORY:

The sorting technique used here is relatively simple. First consider the first two numbers of the array. See if this pair of numbers is out of order. Similarly examine successively each pair of numbers in the array. If any pair is out of order, interchange the numbers in the pair. Perform this step for count times, count being one less than array length.

EXAMPLE:

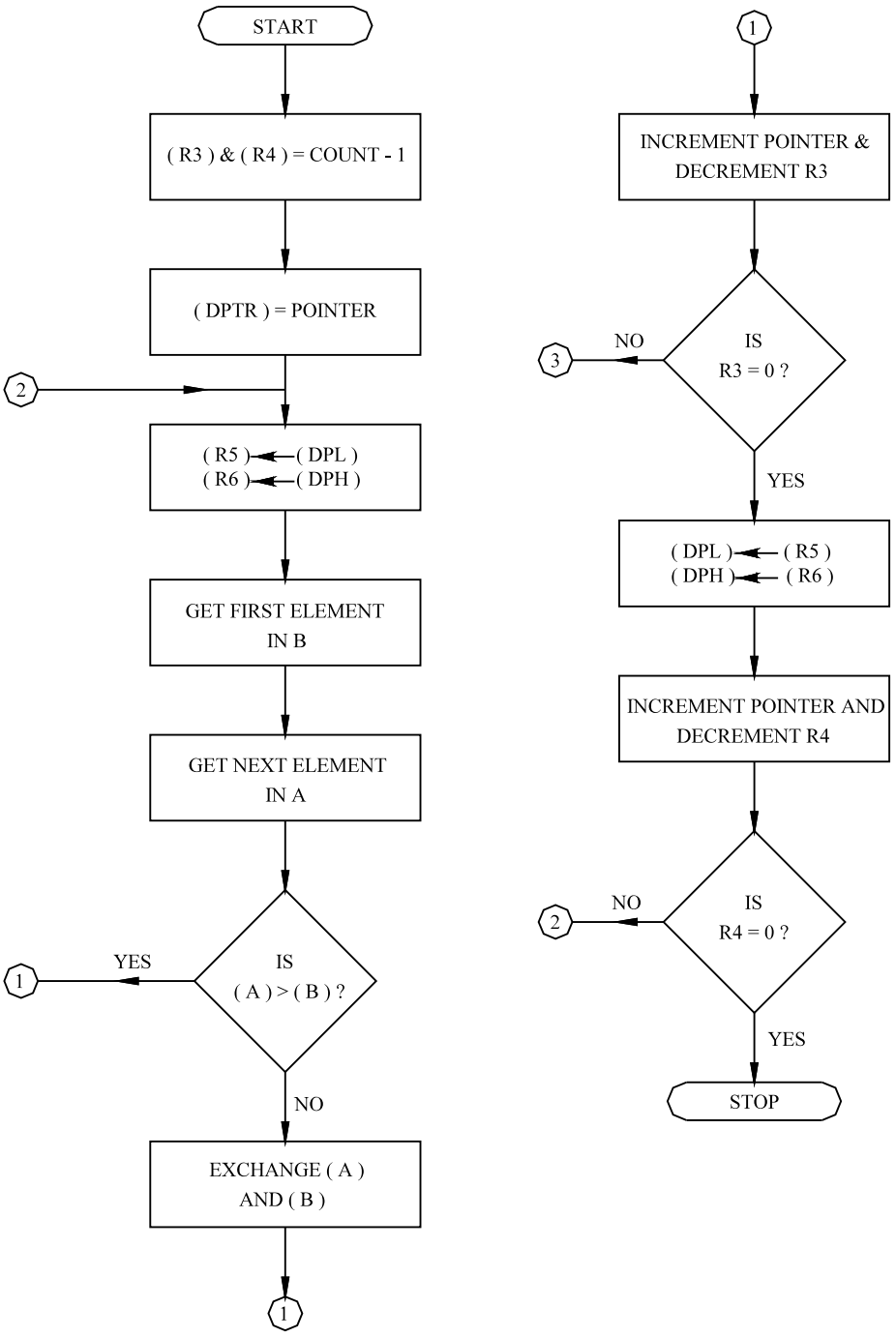
The length of the array is in register R3. The array starts from 4501.

Data : [4500] = 2A
[4501] = 1C
[4502] = F9
[4503] = 88
[4504] = 76

Result : [4500] = 1C
[4501] = 2A
[4502] = 76
[4503] = 88
[4504] = F9

FLOW CHART:

ASCENDING ORDER OF AN ARRAY



PROGRAM:

```
        MOV     R3,#4       ;Count-1
        MOV     R4,#4
        MOV     DPTR,#4500
REPT1:  MOV     R5,DPL
        MOV     R6,DPH
        MOVX    A,@DPTR
        MOV     B,A
REPT:   INC     DPTR
        MOVX    A,@DPTR
        MOV     R0,A
        CLR     C
        SUBB    A,B
        JNC     CHKNXT
EXCH:   PUSH    DPL
        PUSH    DPH
        MOV     DPL,R5
        MOV     DPH,R6
        MOV     A,R0
        MOVX    @DPTR,A
        POP     DPH
        POP     DPL
        MOV     A,B
        MOVX    @DPTR,A
        MOV     B,R0
CHKNXT: DJNZ    R3,REPT
        DEC     R4
        MOV     A,R4
        MOV     R3,A
        INC     R   4
        MOV     DPL,R5
        MOV     DPH,R6
        INC     DPTR
        DJNZ    R4,REPT1
HLT:    SJMP    HLT
```

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	7B	MOV R3,#4
4101	04	
4102	7C	MOV R4,#4
4103	04	
4104	90	MOV DPTR,#4500
4105	45	
4106	00	
4107	AD	REPT1: MOV R5,DPL
4108	82	
4109	AE	MOV R6,DPH
410A	83	
410B	E0	MOVX A,@DPTR
410C	F5	MOV B,A
410D	F0	
410E	A3	REPT: INC DPTR
410F	E0	MOVX A,@DPTR
4110	F8	MOV R0,A
4111	C3	CLR C
4112	95	SUBB A,B
4113	F0	
4114	50	JNC CHKNXT
4115	13	
4116	C0	EXCH: PUSH DPL
4117	82	
4118	C0	PUSH DPH
4119	83	
411A	8D	MOV DPL,R5

411B	82	
411C	8E	MOV DPH,R6
411D	83	
411E	E8	MOV A,R0
411F	F0	MOVX @DPTR,A
4120	D0	POP DPH
4121	83	
4122	D0	POP DPL
4123	82	
4124	E5	MOV A,B
4125	F0	
4126	F0	MOVX @DPTR,A
4127	88	MOV B,R0
4128	F0	
4129	DB	CHKNXT: DJNZ R3,REPT
412A	E3	
412B	1C	DEC R4
412C	EC	MOV A,R4
412D	FB	MOV R3,A
412E	OC	INC R 4
412F	8D	MOV DPL,R5
4130	82	
4131	8E	MOV DPH,R6
4132	83	
4133	A3	INC DPTR
4134	DC	DJNZ R4,REPT1
4135	D1	
4136	80	HLT: SJMP HLT
4137	FE	

PROCEDURE:

i. Key in the opcodes from 4100.

ii. Enter data at 4500.

iii. Execute the program and check for results from 4500.

DISCUSSION:

In the above program for ascending order, what will happen if two numbers compared are similar? What will happen if JC is used instead of JNC in the program.

The algorithm used above is very simple and is called **Bubble sort.** Various other methods like Quick sort, Insertion sort exist for which also programs may be written.

EXERCISES:

i. Perform a 16-bit sort of an array of unsigned numbers in descending order. The length of the array is at 4550 and the array starts from 4551. In this array, MSB is stored first.

Sample problem:

Data : [4550] = 03
[4551] = 2A
[4552] = B5
[4553] = 60
[4554] = 3F
[4555] = D1
[4556] = C6

Result : [4551] = D1
[4552] = C6
[4553] = B5
[4554] = 60
[4555] = 3F
[4556] = 2A

ii Given five arrays of 8-bit numbers, find the smallest number in each array. Do a descending order sort of the resultant numbers.

Data	:	First array	:	[4550]	= 3A
				[4551]	= 23
				[4552]	= AC
				[4553]	= 79
				[4554]	= F3
		Second array	:	[4555]	= 3B
				[4556]	= 4C
				[4557]	= 5E
				[4558]	= 89
				[4559]	= 3F
		Third array	:	[455A]	= 10
				[455B]	= 2E
				[455C]	= 9F
				[455D]	= 05
				[455E]	= 5D
		Fourth array	:	[455F]	= 37
				[4560]	= 12
				[4561]	= 49
				[4562]	= 72
				[4563]	= 89
		Fifth array	:	[4564]	= 17
				[4565]	= 88
				[4566]	= FD
				[4567]	= 28
				[4568]	= 97

Smallest number from each array:

		Array 1	= 23
		Array 2	= 3B
		Array 3	= 05
		Array 4	= 12
		Array 5	= 17
	Result :	[4600]	= 3B
		[4601]	= 23
		[4602]	= 17
		[4603]	= 12
		[4604]	= 05

2.8 STACK AND SUBROUTINES:

The **STACK** is a group of memory locations in the On-chip Internal Data RAM (here in 8051) that is used for temporary storage of data during program execution. The Stack Pointer Register is 8 bits wide.

A **SUBROUTINE** is a group of instructions that perform a subtask (e.g. time delay or arithmetic operation) of repeated occurrence. The subroutine is written as a separate unit, apart from the main program and the microprocessor transfers program execution from the main program to the subroutine whenever it is called to perform the task. After completion of the subroutine task, the execution returns to the main program. Before implementing a subroutine technique, the stack must be defined. The stack is used to store the address of the instruction in the main program that follows the subroutine call.

2.8.1 STACK OPERATIONS:

Stack operations move data between register and the top of the stack. It is incremented before data is stored during PUSH and CALL executions. While the stack may reside anywhere in the chip RAM, the Stack Pointer is initialized to 07 after a reset. This causes the stack to begin at location 08.

We can, and many times should and must, load the Stack pointer at another location. It cannot be located above 7F in the on-chip data RAM because the SFRs are located above 7F, and they will carry special information that could be destroyed by any stack operations carried out there.

To push registers onto the stack before branching to a subroutine, the following instructions may be employed.

```
PUSH    A
PUSH    B
PUSH    PSW
PUSH    DPL
PUSH    DPH
PUSH    IP
PUSH    IE
```

Upon returning from the routine, the register contents can be retrieved using the POP instruction.

```
POP     IE
POP     IP
POP     DPH
POP     DPL
POP     PSW
POP     B
POP     A
```

The addressing mode employed in both operations is immediate.

Let us examine the stack operation with a typical example.

```
MOV     SP,#10
MOV     A,#88
MOV     B,#67
MOV     DPL,#43
PUSH    A
PUSH    B
PUSH    DPL
```

The stack pointer contents and the contents of the stack itself are as depicted below after the execution of each instruction.

Instruction	Register Contents			Stack Contents
	A	B	DPL	
MOV SP,#10	XX	XX	XX	SP initialised to 10 of internal RAM
MOV A,#88	88	XX	XX	A has 88
MOV B,#67	88	67	XX	B has 67
MOV DPL,#43	88	67	43	DPL has 43
PUSH A	88	67	43	SP incremented by 1 and (A) is stored here
PUSH B	88	67	43	SP incremented by 1 and (B) is stored here
PUSH DPL	88	67	43	SP Incremented by 1 and (DPL) is stored here.

STACK CONTENTS AFTER INSTRUCTION PUSH A

10	XX
11	88

STACK CONTENTS AFTER INSTRUCTION PUSH B

10	XX
11	88
12	67

STACK CONTENTS AFTER INSTRUCTION PUSH DPL

10	XX
11	88
12	67
13	43

XX = USER DEFINED DATA

2.8.2 SUBROUTINES:

If the structure of a program requires repetitive execution of a certain set of instructions, these instructions may be combined as a routine and execution of this routine may be done as and when required by employing two types of CALL instructions namely LCALL and ACALL. Return from the subroutine to the main program is by the use of the RET instruction. ACALL is Absolute CALL and LCALL is Long CALL.

ABSOLUTE CALL:

The following is the sequence of the ACALL instruction.

i The contents of PC is incremented twice.

ii Stack Pointer is incremented by one.

iii. low-order PC is pushed onto the stack.

iv. Stack Pointer is incremented by one.

v. The high-order PC is pushed onto the stack.

vi. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, opcode bits 7-5, and the second byte of the instruction.

vii. Now, the routine starting from (PC) is executed.

viii. RET returns control to the instruction following ACALL.

This Absolute branch can access any instruction within the same 2K block of Program memory as the first byte of the instruction following ACALL.

The long branch can access any instruction in the total 64K memory area since the relative jump is specified by 16 bits.

LONG CALL:

The following is the sequence of the LCALL instruction.

i. The contents of PC is incremented by three.

ii. Stack Pointer is incremented by one.

iii. The low-order PC is pushed onto the stack.

iv. Stack Pointer is incremented by one.

v. The high-order PC is pushed onto the stack.

vi The second byte of the instruction is loaded to PCH and the third byte of the instruction is loaded to PCL.

vii Now, the instruction at (PC) is executed.

viii. RET returns control to the instruction following LCALL.

NOTE:

While using subroutines;
i. Save all registers employed in the subroutine on the stack.
ii. Avoid using the stack in the subroutine since it contains the return address.

2.9 DELAY LOOPS:

OBJECTIVE:

To write a program that generates a delay of 0.5 ms with the 8051 at the Clock rate of 12 MHz.

THEORY:

Timing intervals or delays can be generated by either of the following three modes.

i. Hardware delays employing monostable multivibrators which produce a pulse output as decided by the trigger.

ii. Programmable timers which are the combination of hardware and software.

iii. Software loops employing register(s) as a counter.

Let us now examine the third method more closely; what needs to be done is summarised as below.

Load a register with a count value.
Decrement the register contents.
If the register does not contain 0, then repeat previous step.

The delay interval depends upon the execution time required by each instruction.

EXAMPLE:

In this example, a 0.5 ms delay is produced employing the R0 register. Calculation of the count value is also detailed below.

PROGRAM:

```
        MOV     R0,#0FB
REPT:   DJNZ    R0,REPT
```

DELAY CALCULATION:

The MOV R0,#COUNT instruction takes execution time of 1 Microsec. and DJNZ instruction takes 2 Microsecs.

The count to be loaded into register R0 is as calculated below:

1 + 2 x count = Execution time.

Required Delay = 500 micro seconds.

500 = 1 + 2 x count

$$\text{Count} = \frac{500 - 1}{2}$$

= 249.5 D.

In hex the count = FB in hex.

The value FB has to be loaded in register R0 to obtain a delay of 0.5 millisecond.

OBJECT CODES:

MEMORY ADDRESS	OBJECT CODES	MNEMONICS
4100	78	MOV R0,#FB
4101	FB	
4102	D8	REPT: DJNZ R0,REPT
4103	FE	

PROCEDURE:

i. Employ this delay subroutine with another program so that the delay is meaningful and visible.

ii. Change counter value and repeat to see if the delay varies.

DISCUSSION:

To further increase the delay, nested loops which decrement two registers in cascade can be employed.

EXERCISE:

Implement a real time Digital clock using software delay loops.

CHAPTER 3

PERIPHERAL INTERFACING & HARDWARE EXAMPLES

OBJECTIVES

i. To illustrate the interfacing of 8255 with 8031 / 8051.

ii. To illustrate the interfacing of 8279 with keyboard and display interface using 8031 / 8051.

iii. To illustrate the interfacing of printer using 8031 / 8051.

iv. To illustrate the interfacing of LCD keyboard and On-Chip features of 8031 / 8051.

3.1 INTRODUCTION:

This Chapter emphasizes more on the **INTERFACE PROGRAMMING ASPECTS** of the various peripherals available on Micro-51 EB .

As in the earlier Chapter, you will have to try and exercise the given **EXAMPLES AND EXERCISES** to strengthen your technical muscle, in understanding the operational capabilities of the peripheral devices used. Chapters 2 and 3 should no doubt set a milestone in shaping you into a proficient system programmer.

Ample details pertaining to the programming aspect of the peripherals has been provided in the succeeding sections. But for further details that concern the internal architecture, timing diagrams and hardware organisation of the chips, the user is requested to consult the Manufacturer Data Sheets of Intel or National Semiconductor as the case may be.

NOTE:

1. The signals specified with a "*" to the right indicate active low signals.

2. In the example programs, as the programs are common for Micro-51 EB except the peripheral addresses, the peripheral addresses are denoted as 'XXXX' in the following programs. The user has to substitute the correct address depending on the kit in which he is working by referring the table given.

3.2 PARALLEL INTERFACE:

3.2.1 COMPONENT:

The Intel 8255, Programmable Peripheral Interface is the one used here to provide the Parallel interface. Two such numbers are available providing a total of 48 I/O lines for user interface.

The 8255 is a widely used, programmable, parallel I/O device. It can be programmed to transfer data under various conditions, from simple I/O to interrupt I/O. It is flexible, versatile and economical (when multiple I/O ports are required), but somewhat complex. It is an important general purpose I/O device that can be used with almost any microprocessor.

Its **unique features** are,

24 programmable I/O pins
Improved timing characteristics
Direct bit set, reset capability
Reduces system package count

3.2.2 COMPONENT DESCRIPTION:

The 24 I/O lines of each 8255 can be programmed in two groups of 12 and used in three major modes of operation. It has a CONTROL REGISTER to which the modes of the three ports are written into. It cannot be read.

a) BASIC SYSTEM INTERFACE:

The control pins with which the CPU communicates with 8255 are the RESET, CS*, RD*, WR*, A0, A1, D0 - D7. Its basic operation is as given in the following table.

RD*	-	Active Low Read
WR*	-	Active Low Write
CS*	-	Active Low Chip Select
A0,A1	-	Select Control Register / Port
D0-D7	-	Bidirectional Data Bus

The basic operation of the 8255 using the above signals is as below:

A1	A0	RD*	WR*	CS*	FUNCTION
					INPUT OPERATION (READ)
0	0	0	1	0	PORT A ? DATA BUS
0	1	0	1	0	PORT B ? DATA BUS
1	0	0	1	0	PORT C ? DATA BUS
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	DATA BUS ? PORT A
0	1	1	0	0	DATA BUS ? PORT B
1	0	1	0	0	DATA BUS ? PORT B
1	1	1	0	0	DATA BUS ? CONTROL
					DISABLE FUNCTION
X	X	X	X	1	DATA BUS ? 3 - STATE
1	1	0	1	0	ILLEGAL CONDITION
X	X	X	1	0	DATA BUS ? 3 - STATE

3.2.3 PROGRAMMING DESCRIPTION:

The 8255 offers three modes of operation, that can be selected by writing control words appropriately. The three modes are,

MODE 0 - Basic Input / Output
MODE 1 - Strobed Input / Output
MODE 2 - Bi-Directional Bus

The 24 I/O lines of the 8255 are organised as three ports A, B and C each having eight I/O lines. The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions.

PROGRAMMING THE CONTROL REGISTER:

Figure F3.8 illustrates the operation of 8255 in all the three modes. From the figure, it is understood that in Mode 0, all the ports can be configured either as input or output port. Hence, this mode can be used for almost all applications.

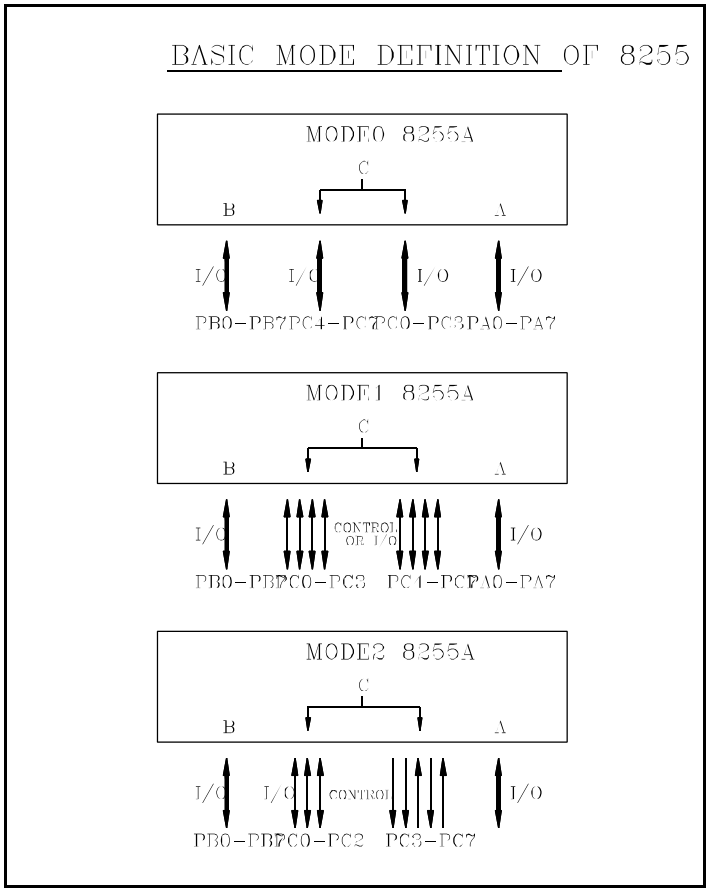


Figure F3.8

Figure F3.9 shows the mode definition format for 8255. Using this, the mode control word is configured according to the requirement and can be written into the Control register. For instance to have,

APORT as Input Port,

BPORT as Output Port,

CPORT higher nibble as Input Port,

CPORT lower nibble as Output Port.

the control word will be **9C**.

SINGLE BIT SET / RESET FEATURE:

Any of the eight bits of Port C can be set or reset using a single OUTput instruction. This feature reduces software requirements in Control-based applications. The Bit Set / Reset Format is given in Figure F3.10. When Port C is being used as status / control for Port A or Port B, these bits can be set or reset using the Bit Set / Reset operation just as if they were data output ports.

PROGRAMMING SEQUENCE:

From the above discussion, it should be clear that the sequence involved in configuring the 8255 consists of,

i) Writing the mode definition format byte into the Control register.

ii) Writing/Reading the data from the ports according to their configuration.

3.2.4 CIRCUIT IMPLEMENTATION:

The Figure F3.11a, F3.11b and F3.11c give a clear idea of the **interface between the 8255 and 8051** in Micropower-i, Micro-51 EB respectively.

As seen from the block diagram, the D0-D7 lines are connected to the system data lines. The RESET, RD* and WR* are connected to RESET, IOR* and IOW* lines of the circuit respectively. The CS* is got from 8255SELECT* which goes low for I/O addresses A00C to A00F (E00C to E00F) and (FF0C to FF0F) for 8255-1 and A010 to A013 (E010 to E013) and (FF10 to FF13) for 8255-2 (E014 to E017 for 8255-3).

MODE DEFINITION FORMAT

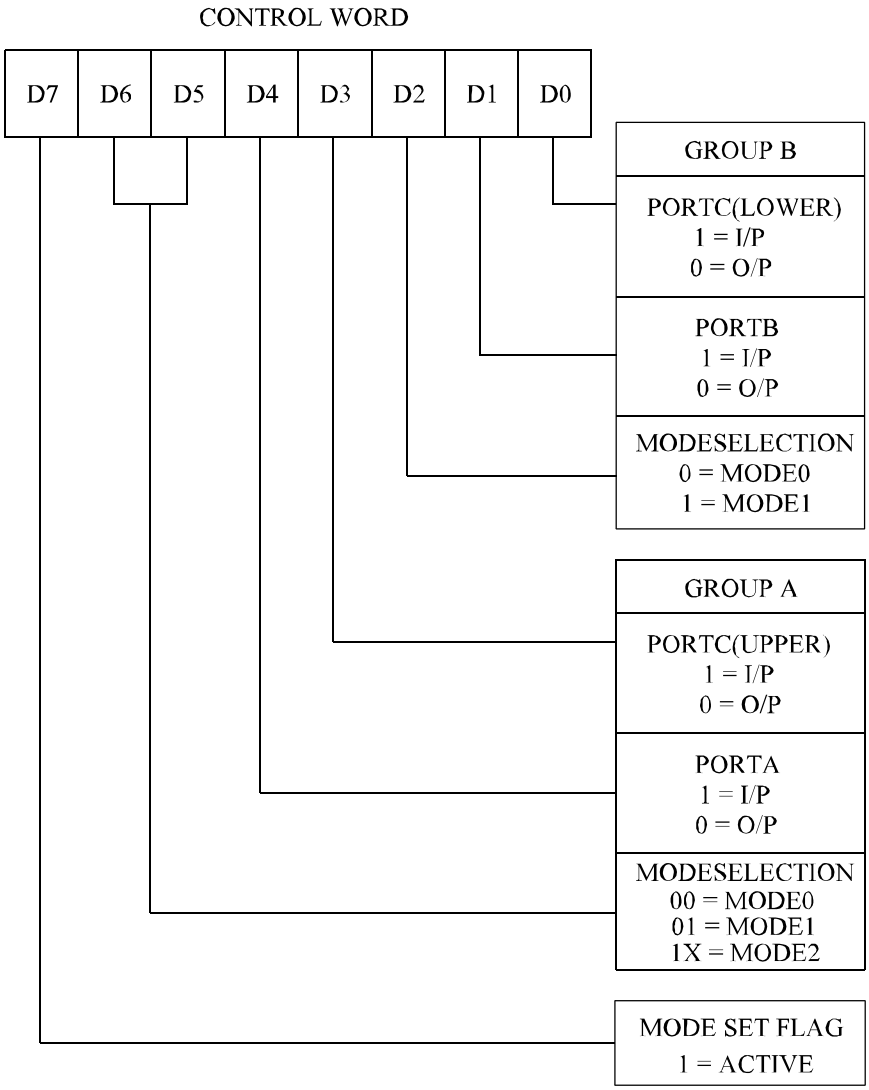


Figure F3.9

BIT SET / RESET FORMAT

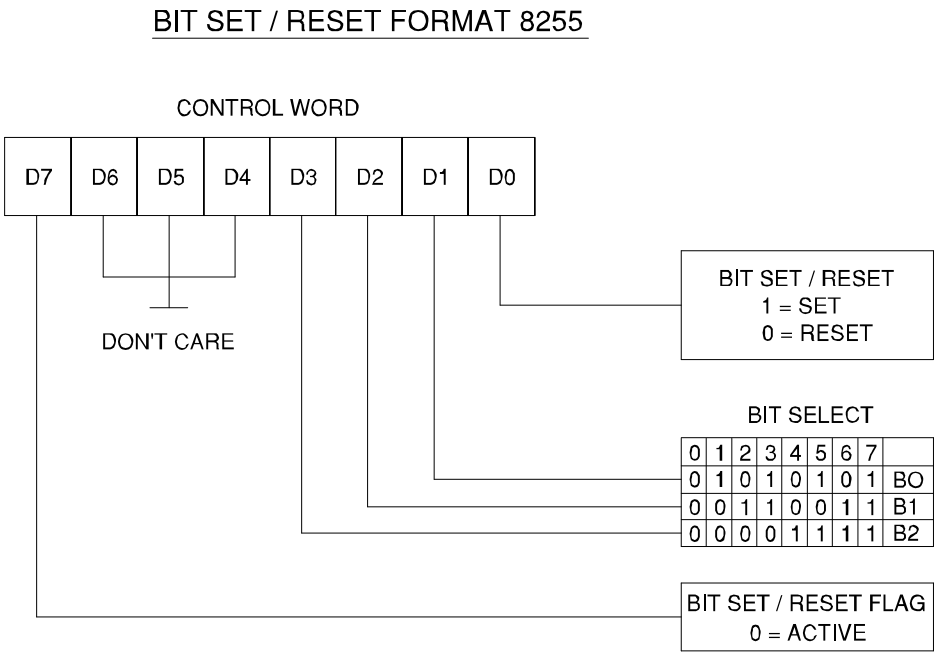


Figure F3.10

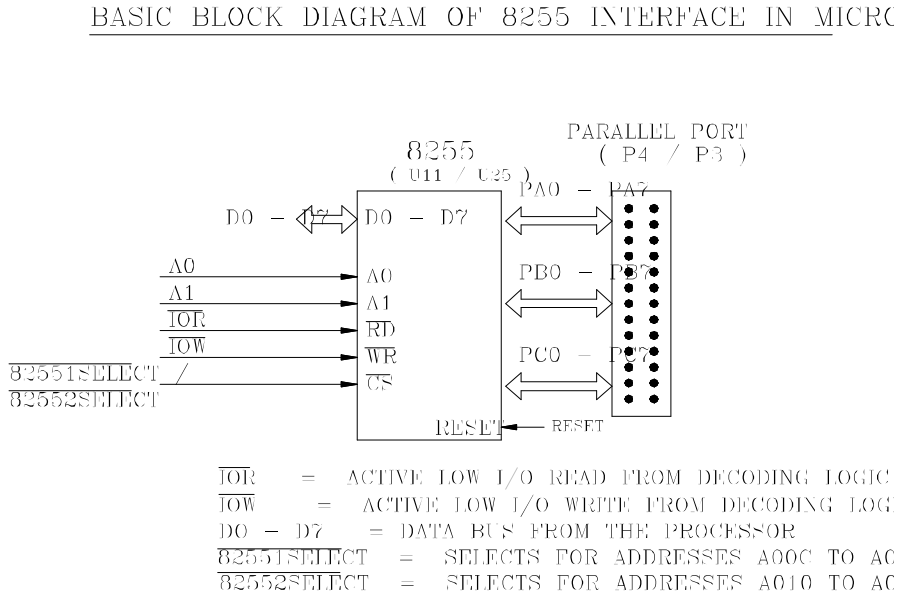


Figure 3.11 A

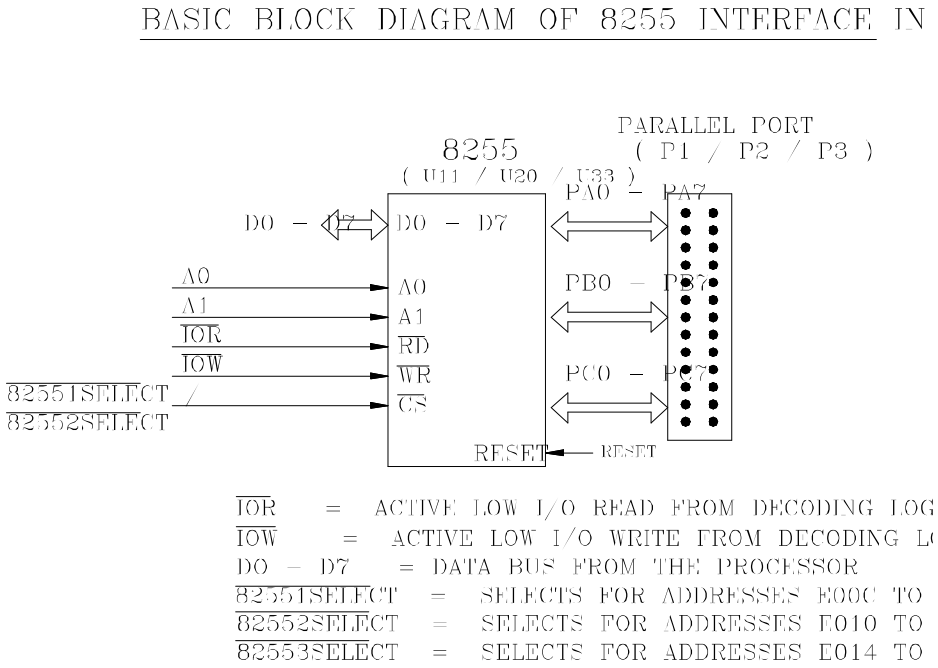


Figure 3.11B

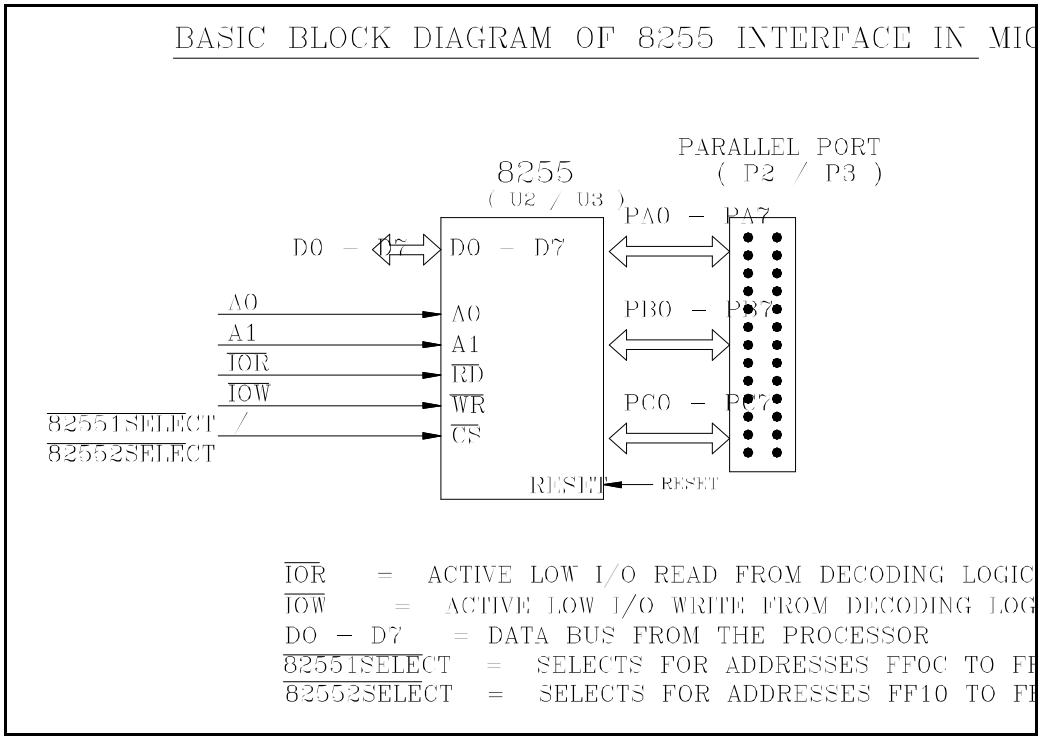


Figure 3.11C

The following are the I/O addresses for 8255:

Function	Address in Micropower - i	Address in Micro-51	Address in Micro - 51EB
Control Register	A00F	E0FF	FF0F
Port A	A00C	E00C	FF0C
Port B	A00D	E00D	FF0D
Port C	A00E	E00E	FF0E
Control Register	A013	E013	FF13
Port A	A010	E010	FF10
Port B	A011	E011	FF11
Port C	A012	E012	FF12
Control Register	-	E017	-
Port A	-	E014	-
Port B	-	E015	-
Port C	-	E016	-

3.2.5 EXAMPLE 1 - SQUARE WAVE GENERATION:

OBJECTIVE:

To generate a Square wave at all 24 I/O lines of 8255-1. The frequency of the Square wave depends solely on the time delay routine used.

THEORY:

To generate a Square wave using 8255, first initialise all the ports of the 8255 as output ports by writing the corresponding control word. Then, make the 24 I/O lines high and low with a certain delay and you will get a square wave out of 8255. If you don't give any delay, then you cannot see any square wave because of the speed at which the 8255 output lines change states from high to low and low to high. Increase the delay to reduce the frequency and reduce the delay to increase the frequency.

PROGRAM:

```
4100  90 XX XX          MOV     DPTR,#CNT_1
4103  74 80             MOV     A,#80
4105  F0                MOVX    @DPTR,A
4106  74 FF       REPT: MOV     A,#FF
4108  12 41 11           LCALL   OUT
410B  E4                CLR     A
410C  12 41 11           LCALL   OUT
410F  80 F5              SJMP    REPT
4111  90 XX XX     OUT:  MOV     DPTR,#APORT_1
4114  F0                MOVX    @DPTR,A
4115  A3                INC     DPTR
4116  F0                MOVX    @DPTR,A
4117  A3                INC     DPTR
4118  F0                MOVX    @DPTR,A
                  DELAY:
4119  78 FF             MOV     R0,#FF
411B  D8 FE       DLY:  DJNZ    R0,DLY
411D  22                RET
```

VERIFICATION:

After entering the program given above and executing it, verify for the square wave at the Port pins of the 8255-1. Calculate the frequency for the delay used. Change delay and verify whether the frequency changes accordingly or not.

3.2.6 EXERCISES:

i) Configure 8255-1 as output and 8255-2 as input ports. Connect the Parallel port connectors of 8255-1 and 8255-2 using a 26-core flat cable. Output data to 8255-1 and check them by inputting through 8255-2.

ii) Using our PLC add-on board, write a program that will generate a sequential logic display on PLC through 8255-2.

1. Output 01 to Port A, give a delay of 2 secs.
2. Output 02 to Port A, give a delay of 3 secs.
3. Output 04 to Port A, give a delay of 5 secs.
4. Output 08 to port A, give a delay of 4 secs.
5. Repeat again from step 1.

3.3 KEYBOARD / DISPLAY INTERFACE:

3.3.1 COMPONENT:

Intel's **8279 - Keyboard and Display Controller (KDC)** is responsible for the Keyboard / Display interface.

Its main features are,

i) Simultaneous Keyboard Display operation.

ii) 3 input modes such as Scanned Keyboard Mode, Scanned Sensor Mode, Strobed Input Entry Mode.

iii) 2 output modes such as 8 or 16 character multiplexed displays, right or left entry display formats.

iv) Clock Prescaler.

v) Programmable Scan Timing.

vi) 2-Key lockout or N-key Roll-over with contact debounce.

vii) Auto Increment facility for easy programming.

3.3.2 COMPONENT DESCRIPTION:

The Intel 8279 is responsible for the debouncing of the keys, the coding of the keypad matrix and the refreshing of the display elements in the Micropower-i, Micro-51 and Micro-51EB . The keyboard portion can provide a scanned interface to a 64-contact key matrix.

The keyboard portion will also interface to an array of sensors or a strobed interface keyboard. Key depressions can be 2-key lockout or N-key roll-over. Keyboard entries are debounced and strobed in an 8-character FIFO. Key entries set the interrupt output line.

The display portion provides a scanned display interface for LED, incandescent and other popular display technologies. Both numeric and alphanumeric segment displays may be used as well as simple indicators. The 8279 has 16 x 8 display RAM which can be loaded or interrogated by the CPU. Both right entry, calculator and left entry typewriter display format are possible. Both read and write of the display RAM can be done with or without auto-increment of the display RAM address.

a) BASIC SYSTEM INTERFACE:

The 8279 is designed specifically to connect to any 8-bit microprocessor bus directly. This allows the CPU to do some other work other than scanning the keyboard and refreshing the display. The CPU can program the operating modes for 8279.

The control signals with which the 8279 communicates with the CPU are the Data Bus D0-D7, RST, CS*, A0, RD*, WR*, and IRQ.

D0 - D7 - Bidirectional data bus
RST - Reset Input
CS* - Active Low Chip Select
A0 - Buffer Address Select
RD* - Active Low Read Input
WR* - Active Low Write Input
IRQ - Interrupt Request Output

The functional definition is as below:

A0	RD*	WR*	FUNCTION
0	0	1	Read Display RAM or keyboard FIFO
0	1	0	Write Display RAM
1	0	1	Read Status
1	1	0	Write Command

The interface pins for Keyboard and Display are,

SL0 - SL3 - Scan lines. Used to Scan the key switch or sensor matrix or display digits. They can be encoded or decoded.

RL0 - RL7 - Return line inputs. They are connected to scan lines through the keys or sensor switches. They also serve as an 8-bit input port in the strobed input mode.

SHIFT - This input is a status stored along with the key position.

CNTL/STB - This is a control input for keyboard modes and is stored like a status. This is the strobe line that enters the data into the FIFO in the Strobed Input Mode.

BD* - Blank Display. It blanks the display during digit switching.

A0 - A3) ,

B0 - B3) - - These 2 ports are the outputs of the 16 x 4 display registers. The data on these outputs is synchronised to scan lines (SL0 - SL3) for multiplexed displays. These two 4 bit ports can also be considered as a single 8-bit port. B0 corresponds to D0 of Data bus and A3 to D7.

3.3.3 PROGRAMMING DESCRIPTION:

PROGRAMMING SEQUENCE (8279):

The **KDC** requires proper loading of the following command words for satisfactory operation:

i) Keyboard / Display Mode Setup Word.

ii) Program Clock Select Word.

iii) FIFO / Sensor RAM Read Word.

iv) Display RAM Read Word.

v) Display RAM Write Word.

vi) Blank / Write Inhibit Word.

vii) Clear Display Word.

viii) End Interrupt Setup Mode Word.

ix) FIFO Status Initialisation Word.

Hence the sequence to be followed for displaying or inputting a key, using the KDC is as follows:

i) Select / Display mode, with the proper number of digits and entry mode.

ii) Select the keyboard mode whether encoded, decoded etc.

iii) Clear the digits of the display.

iv) Select row address for display and auto-increment or non-auto-increment mode.

v) Write the data to be displayed in the correct segment definition into the data buffer or input the key code after the keyboard read operation in the data buffer.

Step (i) to (iv) are to be performed with the control section and step (v) is the actual data output or input.

The following discussion elaborates upon the above points.

8279 COMMANDS:

The following commands program the operating modes of Intel 8279. The commands sent on the Data Bus with CS* low and A0 high, are loaded to the 8279 on the rising edge of WR*.

i) KEYBOARD / DISPLAY MODE SETUP

0	**0**	**0**	**D**	**D**	**K**	**K**	**K**

DD - DISPLAY MODE

00 - 8 8-bit Character display - Left Entry

01 - 16 8-bit Character display - Left Entry

10 - 8 8-bit Character display - Right Entry

11 - 16 8-bit Character display - Right Entry

KKK - KEYBOARD MODE

000 - Encoded Scan Keyboard - 2 key Lockout

001 - Decoded Scan Keyboard - 2 key Lockout

010 - Encoded Scan Keyboard - N key rollover

011 - Decoded Scan Keyboard - N key rollover

100 - Encoded Scan Sensor Matrix

101 - Decoded Scan Sensor Matrix

110 - Strobed input, Encoded Display scan

111 - Strobed input, Decoded Display scan.

Intel 8279 supports four types of keyboard modes. They are,

a) Scan Keyboard - 2 key Lockout,
b) Scan Keyboard - N Key Roll over,
c) Scan Sensor Matrix ,
d) Strobed Input Display Scan.

In the Scanned Keyboard Mode, when a key is depressed, the debounce logic is set. Other depressed keys are looked for during the next two scans. If none are encountered, it is a single key depression and the key position is entered into the FIFO along with the status of CNTL and SHIFT lines. If the FIFO was empty, IRQ will be set to signal the CPU that there is an entry in the FIFO. If the FIFO was full, the key will not be entered and the error flag will be set.

If another closed switch is encountered, no entry to the FIFO can occur. If all other keys are released before this one, then it will be entered to the FIFO. If this key is released before any other, it will be entirely ignored. A key is entered to the FIFO only once per depression, no matter how many keys were pressed along with it or in what order they are released. This logic is defined as 2 Key Lockout.

In case of N Key rollover, any number of keys can be depressed. If a simultaneous depression occurs, the keys are recognized and entered to the FIFO according to the order the keyboard scan found them.

In all the above modes, we have two types.

i) Decoded keyboard mode (or)
ii) Encoded keyboard mode

In the decoded keyboard mode, the scan lines SL0 through SL3 are internally decoded and can be directly connected to keyboard matrix. Hence, in the decoded keyboard mode, only a maximum of four displays can be used as the scan lines available are only four.

In the encoded keyboard mode, the scan lines represents a binary count and has to be externally decoded using a three to eight (74LS138) or four to sixteen (74LS154) line decoders. In the encoded keyboard mode, a maximum of sixteen displays can be connected to 8279 by decoding the four scan lines SL0 - SL3.

ii) PROGRAM CLOCK

0	**0**	**1**	**P**	**P**	**P**	**P**	**P**

PPPPP - Determines the value of the prescale which divides the external clock to yield the 100 KHz, to give the specified scan and debounce time.

iii) READ FIFO / SENSOR RAM

0	**1**	**0**	**AI**	**X**	**A**	**A**	**A**

X - Don't care.

AI - Auto Increment Flag - Irrelevant if Scanned Keyboard Mode. For Sensor Matrix Mode, if AI=1, then each successive read will be from subsequent row of the sensor RAM.

AAA - In scanned keyboard Mode - Irrelevant. In Sensor Matrix Mode, it selects one of the 8 rows of sensor RAM.

NOTE: In Scanned Keyboard Mode, the 8279 will automatically drive the data for subsequent read in the same sequence in which data first entered the FIFO.

iv) READ DISPLAY RAM

0	**1**	**1**	**AI**	**A**	**A**	**A**	**A**

AI - Auto Increment Flag. If AI=1, the row address selected will be incremented after each following read or write to the Display RAM.

AAAA - Selects one of the 16 rows of the Display RAM.

v) WRITE DISPLAY RAM

1	**0**	**0**	**AI**	**A**	**A**	**A**	**A**

This command is written with A0=1. All subsequent writes with A0=0 will be to the Display RAM.

vi) DISPLAY WRITE INHIBIT/BLANKING

1	**0**	**1**	**X**	**IW**	**IW**	**BL**	**BL**

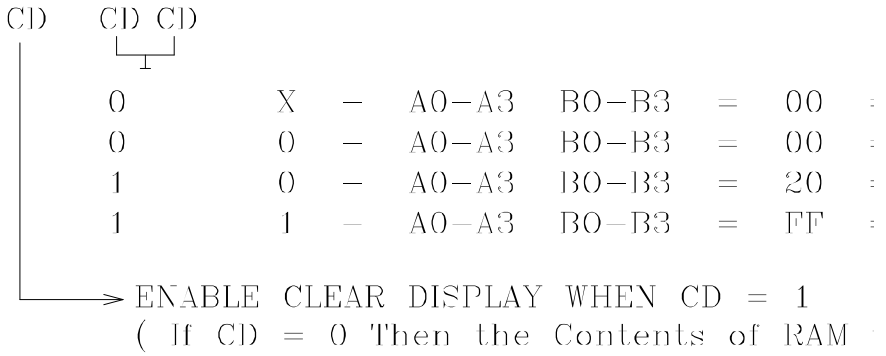
X - Don't Care.

IW - Inhibit writing. By setting IW=1, the corresponding port i.e. A(A0-A3) or B(B0-B3) becomes marked, so that entries to Display RAM do not affect this port.

BL - Blank flag. It is used to blank the display and one flag is available for each nibble

vii) CLEAR

1	1	0	CD	CD	CD	CF	CA



CF - If CF=1, FIFO status is cleared, interrupt output line is reset. Sensor RAM pointer is set to row 0.

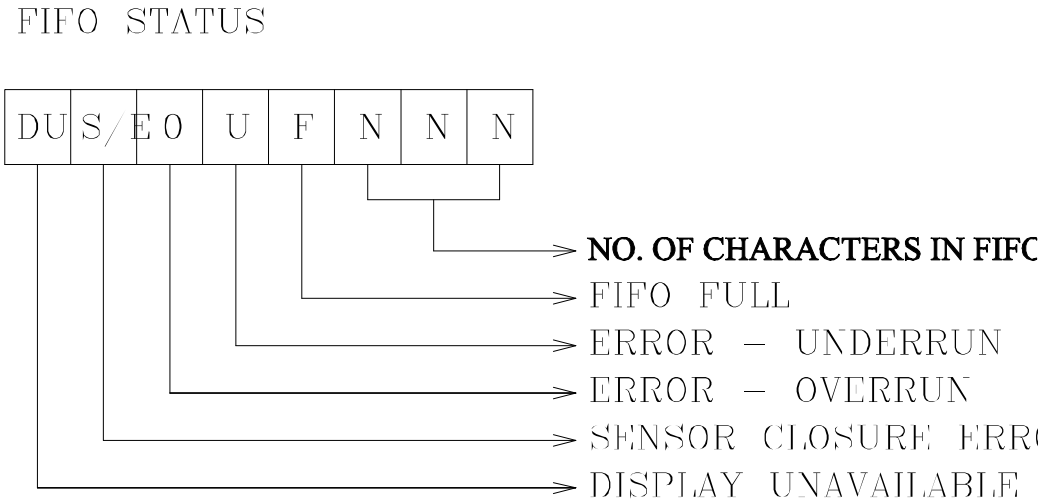
CA - Clear All bit has the combined effect of CD and CF. Resynchronises the internal timing chain. It uses CD clearing code on Display RAM and clears FIFO status.

viii) END INTERRUPT / ERROR MODE SETUP

1	1	1	E	X	X	X	X

E - If E=1, then the chip will operate in the special Error mode for N - key rollover mode.

ix) FIFO STATUS



The keyboard connected has been designed to be used in the Encoded scan keyboard - 2 key lockout mode. The display has been selected for eight 8-bit character display - left entry. A prescale factor of 25 is selected to provide 5 ms keyboard and display scan time and 10 ms debounce time. These initialisations are done during RESET operation.

3.3.4 CIRCUIT IMPLEMENTATION:

The Figure F3.15a, F3.15b and F3.15c give a clear idea of **the 8279 interface with 8051 in Micropower-i, Micro-51 EB.**

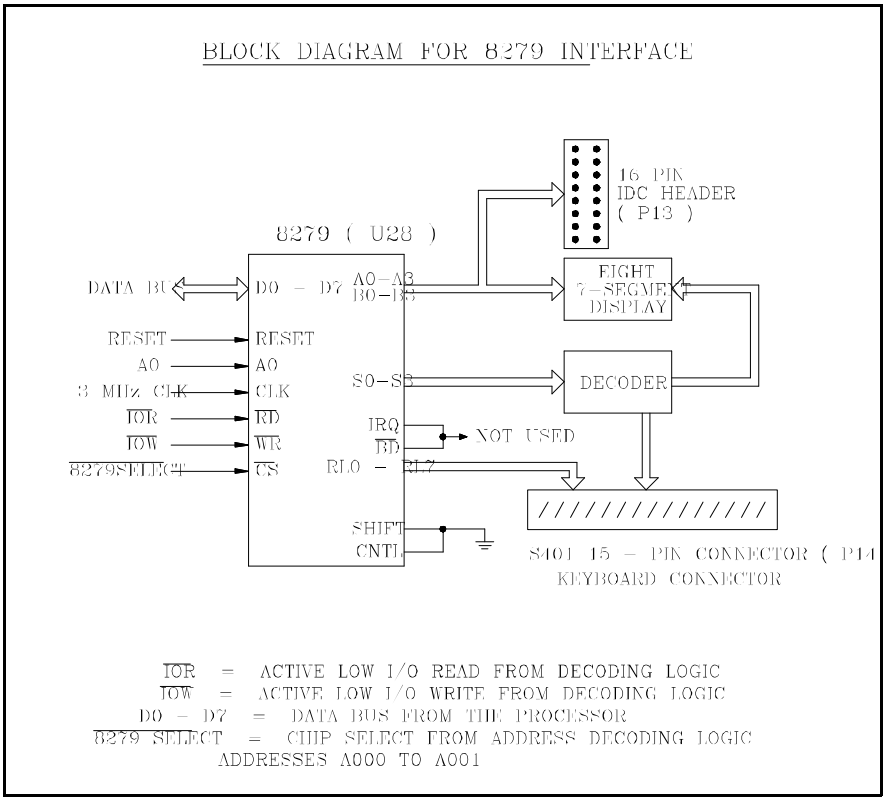
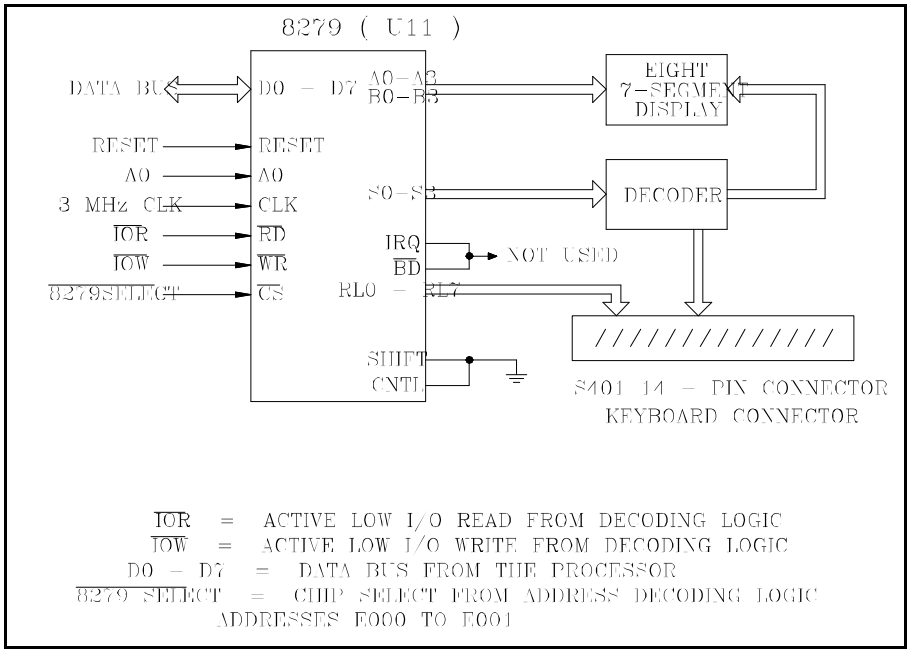


Figure 3.15a



Figrue 3.15b

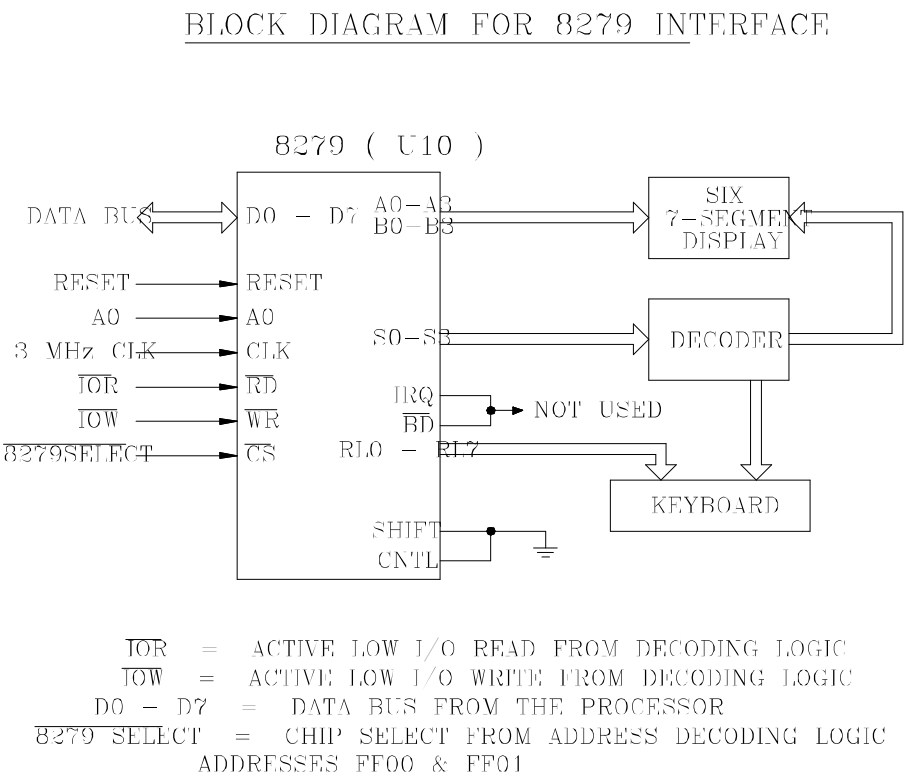


Figure 3.15c

As seen from the block diagram, the D0-D7 lines are connected to the system data bus. The RD*, WR*, RESET, CLK and A0 lines are connected to the IOR*, IOW*, RESET, CLK and A0 lines of the circuit respectively. The RL0-RL7 lines are used for the keyboard interface. The A0-A3 and B0-B3 lines are used for the display interface. The SL0-SL3 lines are decoded and used for both the keyboard and display interface. The CNTL and SHIFT lines are grounded. The IRQ signal is not used.

The I/O address for the 8279 are as follows:

Function	Address in Micropower - i	Address in Micro-51	Address in Micro-51EB
8279 Control / Status 8279 Data	A001 A000	E001 E000	FF01 FF00

3.3.5 EXAMPLE 2 - DISPLAY DATA "A" USING 8279:

OBJECTIVE:

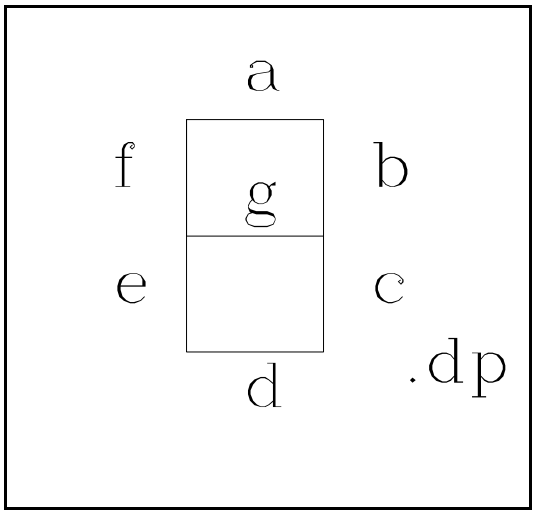
To initialise 8279 and to display the character A in the first digit of the display.

THEORY:

The initialisation of 8279 must be done before writing the data to be displayed. First select the Keyboard / Display mode which is 00 for eight 8-bit character display (6 bit character display for Micro-51 EB) - Left entry for display and Encoded scan keyboard - 2 key lockout for keyboard. Then, clear the display using the command (vii) which will actually clear the Display RAM. Then, write the control word to select the Row address and set Auto- increment flag. Now, write the data to be displayed, which is found by the data bus and display correspondence given below.

The segment definition and correspondence between data bus, segments enabled and 8279 output bits are as follows.

SEGMENT DEFINITION :



PROGRAM:

```
4100   90 XX XX        MOV     DPTR,#DSP_CNT
4103   E4              CLR     A
4104   F0              MOVX    @DPTR,A
4105   74 CC           MOV     A,#CC
4107   F0              MOVX    @DPTR,A          ;Clear Display RAM
4108   74 90           MOV     A,#90
410A   F0              MOVX    @DPTR,A          ;Select auto-row increment
410B   90 XX XX        MOV     DPTR,#DSP_DAT
410E   74 88           MOV     A,#88
4110   F0              MOVX    @DPTR,A
4111   80 FE     HLT:  SJMP    HLT
```

VERIFICATION:

Enter the above program from the address specified and execute it. The display should be "A" in the first digit, and the rest should be blank.

3.3.6 EXAMPLE 3 - READ A KEY:

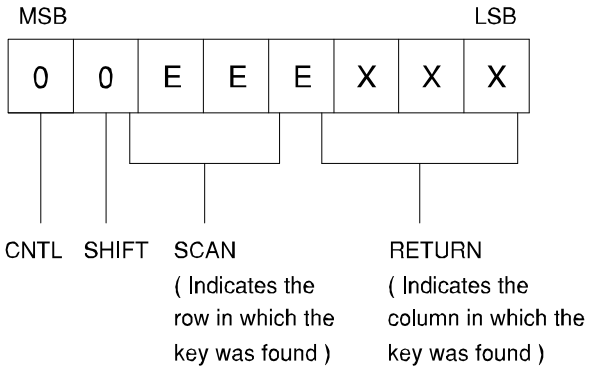
OBJECTIVE:

To read the code of a key pressed from the 8279.

THEORY:

The steps to be followed to read a key from 8279 are given below:

In the Scanned Keyboard Mode, the character entered into the FIFO corresponds to the position of the switch in the keyboard plus the status of CNTL and SHIFT lines. In the hardware, CNTL and SHIFT inputs of 8279 are grounded.



This code will be entered into the FIFO whenever a key is pressed. So,

i) Read the FIFO.

ii) Check if the least significant 3 bits is less than 0111 B. Because any key closure will increment the row indicated by the AAA bits.

iii) Read the data from FIFO RAM, which is the key code.

The following program waits for a key to be pressed in the keyboard, reads the same and stores it in the location 4300.

PROGRAM:

```
4100   90 XX XX            MOV     DPTR,#DSP_CNT
4103   E0          REPT:   MOVX    A,@DPTR
4104   54 07               ANL     A,#7
4106   60 FB               JZ      REPT
4108   74 40               MOV     A,#40
410A   F0                  MOVX    @DPTR,A
410B   15 82                DEC     DPL
410D   E0                  MOVX    A,@DPTR
410E   90 43 00            MOV     DPTR,#4300
4111   F0                  MOVX    @DPTR,A
4112   80 FE        HLT:    SJMP    HLT
```

3.3.7 EXERCISES:

i) Receive a key and display the key code received in the right most digit of the 7 segment Display.

ii) Flash the message " HELLO" using 8279.

iii) Display a rolling message "HAPPY bIrTHdAY to YOU" using 8279.

iv) Initialise 8279 without setting the AI flag and display the message "diSPLAY".

[**Hint:** The Display RAM should be initialised each time a data is to be displayed for, it will not increment display RAM after displaying a data.]

3.4 PRINTER INTERFACE:

3.4.1 COMPONENT:

The Printer interface logic has been implemented here using a 74LS273 as the 8-bit output latch, a 74LS175 as the 4-bit control latch and a 74LS244 as the 5-bit status buffer. The Printer Port in Micropower-i and Micro-51 EB are specifically designed for interfacing a Centronics compatible Parallel Printer, In Micro-51 EB , 8255-1 Port is used for interfacing a centronics compatible parallel printer, but it can also be used as a general input/output port for any application that matches its input/output capabilities. It has 12 TTL buffer output points, which are latched. The interface also has five steady state TTL input points.

3.4.2 COMPONENT DESCRIPTION:

When the Printer port (8255-1 Port in the case of Micro-51 EB) is attached to a Printer to take printouts, 'PRINT' command available in the keypad can be used. When the printer command is activated, data are loaded into the 8-bit latched output port and when the strobe line is activated, data is written to the buffer of the printer connected to the printer port (8255-1 port for Micro-51 EB).

BASIC SYSTEM INTERFACE:

The 8-bit data/latch, the 4-bit control latch and the 5-bit status buffer communicate with the CPU with the help of the bidirectional data bus and the chip select signals. The correspondence between the data bus and the 4-bit control latch (PCNT) is as tabulated below:

4 - BIT CONTROL LATCH (PCNT)				
DATA BUS	D0	D1	D2	D3
PRINTER SIGNAL	INIT *	SLCT IN *	STROBE *	AUTO FEED

The correspondence between the data bus and the -bit status buffer (PSTAT) is as below:

5- BIT STATUS BUFFER					
DATA BUS	D0	D1	D2	D3	D4
PRINTER SIGNAL	SELECT	P-END	ERROR *	ACK*	BUSY

SIGNAL TIMING:

Each character transmitted to the Printer must be clocked by a STROBE* signal. When a STROBE* is received, the Printer sets the BUSY signal to inform the host that the printer is no longer able to receive further data. When the character has been processed, an ACK* pulse is sent and BUSY is dropped to inform the host that the Printer can receive further data.

When the Printer is powered-ON, a HIGH level BUSY signal is output by the Printer until the Printer is put in the READY state to accept data. At the end of the initialisation, an ACK* pulse is sent to the host before dropping the BUSY signal. The Printer timing diagram is given in Figure F3.16 which clearly indicates the timing of STROBE, data latch etc.

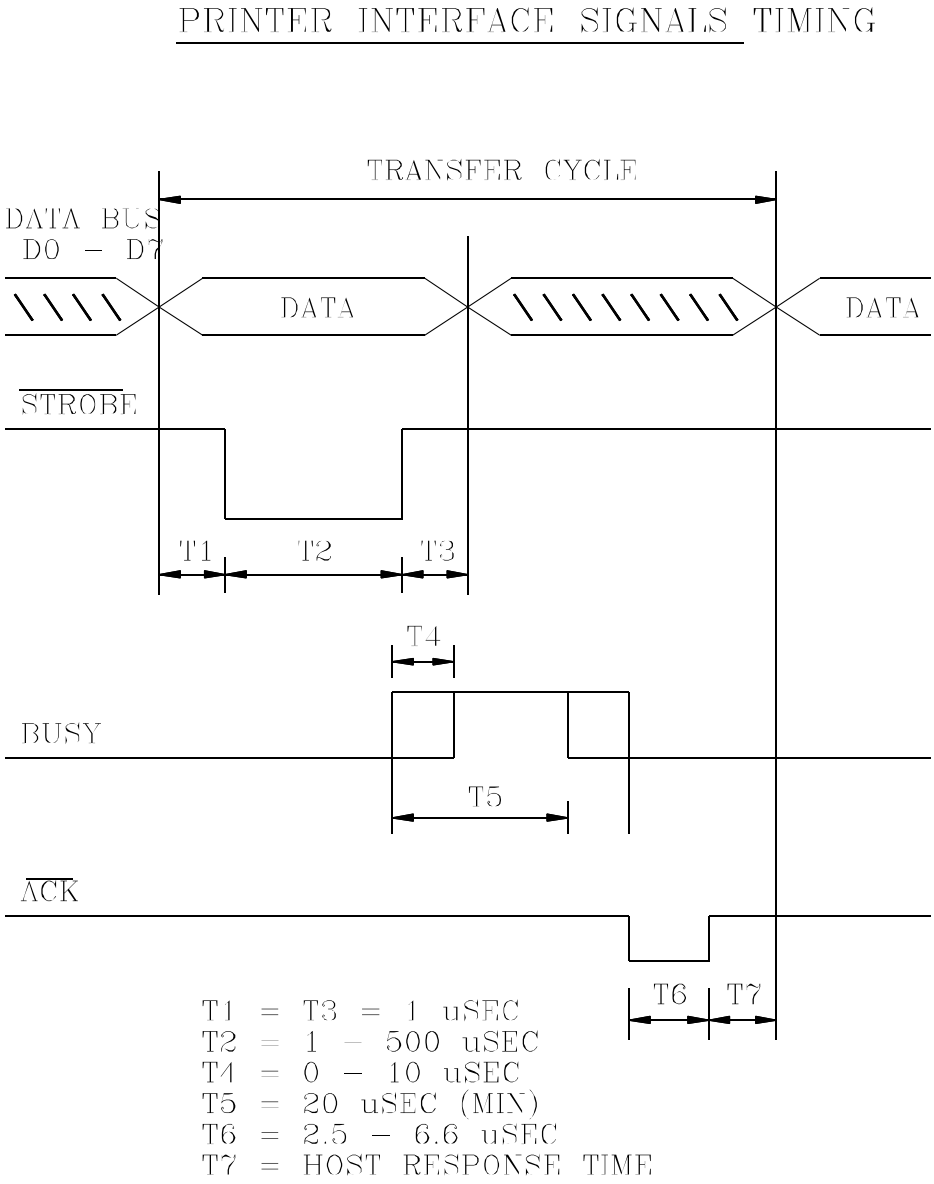


Figure F3.16

3.4.3 PROGRAMMING DESCRIPTION:

Simplifying the process of programming or making the Printer print the data supplied by our system, we shall limit ourselves to the discussion of BUSY and ERROR signals only.

PROGRAMMING SEQUENCE:

i) Check if the Printer is busy using the data read from status buffer.

ii) Write data using the STROBE* control signal for latching the data.

iii) Send a carriage return character to ensure that the Printer prints the character sent.

In the following program, only the BUSY and ERROR* signals are considered to check whether the Printer is busy or ready for receiving more data.

First, the BUSY signal is checked and if the Printer is not busy, then 8-bit data is latched onto the Printer clocked by a STROBE* signal. For the Printer to print that character, a carriage return must be sent. So after sending the Carriage Return it will be seen that the character is printed. The connections between the Printer port of the kit and the Printer is given in Figure F3.17.

CONNECTOR SPECIFICATIONS

(For Micro-51 EB and Micropower-i)

SIGNAL NAME

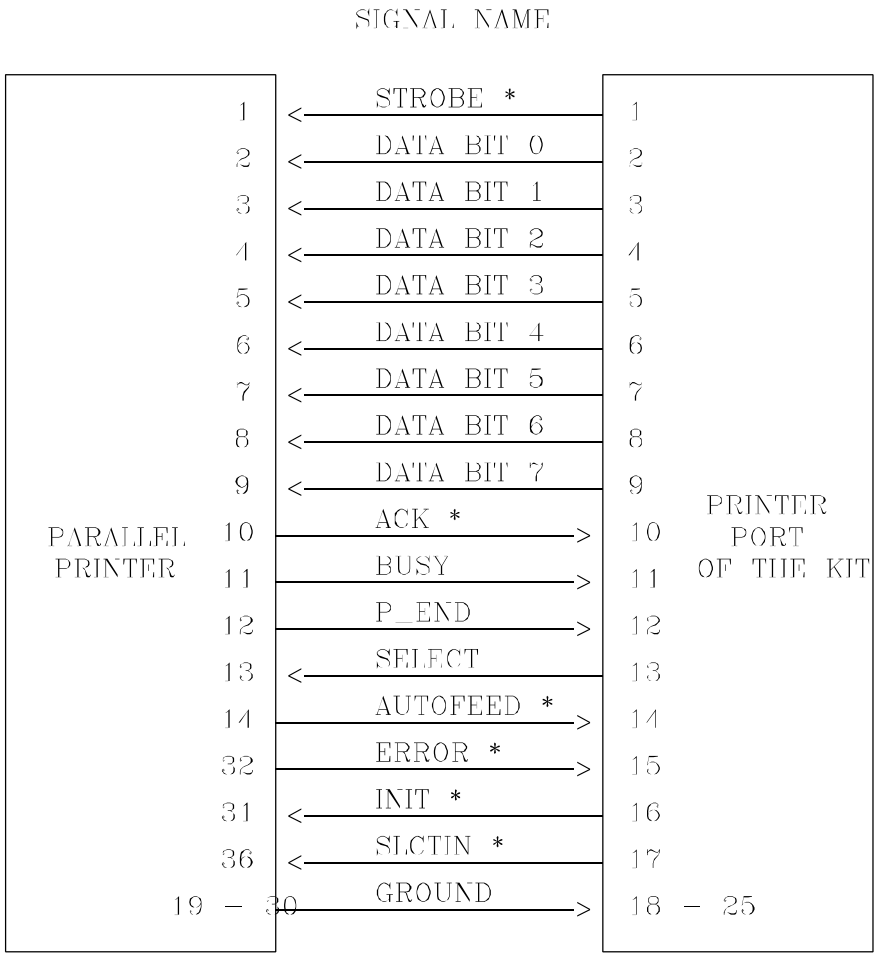


Figure F3.17

3.4.4 CIRCUIT IMPLEMENTATION:

The Figure F3.18 given below gives a clear idea of **the Printer interface**.

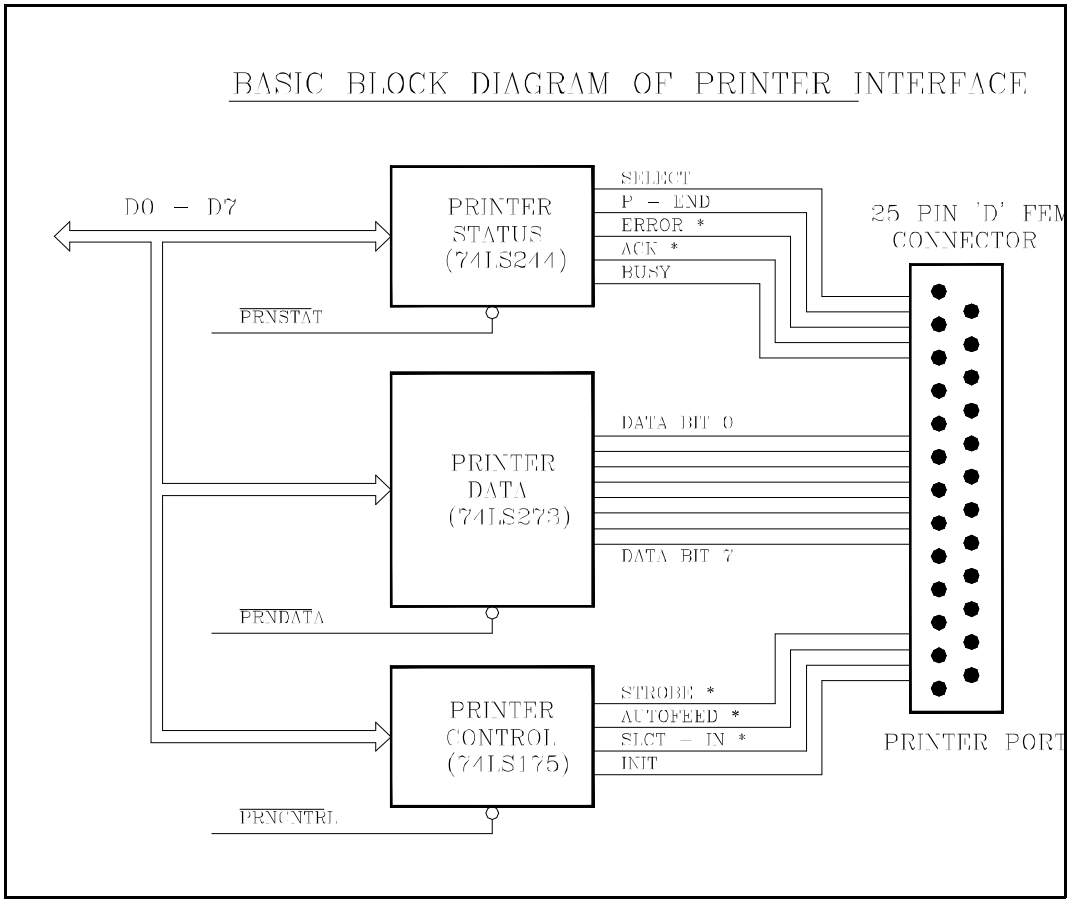


Figure F3.18

As seen from the block diagram, the D0-D7 data lines are used by the two latches and one buffer for the control, data and status lines respectively. The outputs of the two latches and the inputs to the buffer are terminated at a 25-pin D Female connector.

The I/O addresses for the Printer are as follows:

FUNCTION	Address in Mpi	Address in M-51	Address in M-51 EB
Printer Control	A030	E020	FF0E
Printer Data	A034	E028	FF0C
Printer Status	A02C	E024	FF0D

3.4.5 EXAMPLE 4 - PRINT A SINGLE CHARACTER:

OBJECTIVE:

To write a sample program that will print a single character "A" onto the Printer. Checking for the Printer Busy and Error signals should also be done.

THEORY:

First of all, the Printer is initialised to clear the Printer buffer. Then, proper Control word is written to make Strobe high and Select low. Then, the Busy and Error signals are checked by reading its Status Port to know whether the Printer is ready to process further data or not. After the Printer is ready to receive data, the ASCII code of character "A", which is 41 hex, is output to the data port of the Printer. Also, followed by this character is the Carriage Return (0D hex), because the Centronics compatible Printers will print the characters received only after receiving a Carriage Return. If there is any problem in the Printer side or in the Printer port of the kit (8255-1 port for Micro-51 EB), 'E' will be displayed to denote Error.

In the PRNDATA routine of the following program, the Printer status is read and the Printer is checked for the Paper error signals. It will be in a continuous loop if any error is detected. If no errors are encountered, then the data will be outputted while making the strobe low and high after 1 microsecond so that the data will be latched. Now the data can be seen printed.

PROGRAM: (For Micro-51 EB only)

```
4100   90  FF  0F              MOV     DPTR,#FF0F
4103   74 82                       MOV     A,#82
4105   F0                      MOVX    @DPTR,A
4106   90      FF 0E               MOV     DPTR,#PRN_CNT
4109   74      0C                  MOV     A,#0C
410B   F0                      MOVX    @DPTR,A
410C   12  41 4C               LCALL   DELAY
410F   74  0D                  MOV     A,#0D
4111   F0                      MOVX    @DPTR,A
4112   74      41                  MOV     A,#41
4114   12      41      1E              LCALL   PRNDATA
4117   74      0D                  MOV     A,#0D
4119       12      41      1E              LCALL   PRNDATA
411C   80 FE      HLT:        SJMP    HLT
411E   F5 F0      PRNDATA:    MOV     B,A
4120   90 FF 0D   LOOP1:      MOV     DPTR,#PRN_STA
4123   E0                      MOVX    A,@DPTR
4124   54 10                       ANL     A,#10
4126   B4 00 F7                CJNE    A,#0,LOOP1
 4129  E0                      MOVX    A,@DPTR
```

```
412A   54 01                ANL     A,#1
412C   B4 01 15             CJNE    A,#1,ERR
 412F  E5 F0                MOV     A,B
4131   90 FF 0C             MOV     DPTR,#PRN_DAT
4134   F0                   MOVX    @DPTR,A
 4135  12 41 4C             LCALL   DELAY
4138   90 FF 0E             MOV     DPTR,#PRN_CNT
413B   E4                   CLR     A
413C   F0                   MOVX    @DPTR,A
413D   12 41 4C             LCALL   DELAY
4140   74 05                MOV     A,#5
4142    F0                  MOVX    @DPTR,A
4143    22                  RET
                   ERR:
4144    90 FF 00            MOV     DPTR,#DSP_DAT
4147    74 68               MOV     A,#68
4149    F0                  MOVX    @DPTR,A
414A    80 FE       HLT1:   SJMP    HLT1
                   DELAY:
414C    79 FF               MOV     R1,#FF
414E    D9 FE       DLY:    DJNZ    R1,DLY
4150    22                  RET
```

3.4.6 EXERCISE:

i) Print the message **"Hello! I am now aware of printer interface".**

3.5 EPROM PROGRAMMER: **(for Micro-51 EB only)**

3.5.1 COMPONENT:

The **Onboard EPROM programmer** is implemented in Micro-51 EB using the buffers 74LS07 to switch the switching transistors to obtain the appropriate voltages. The buffers are driven by an octal latch 74LS273. The address and data lines for the EPROM are provided through an 8255 operating in output and input mode and the Programmer Socket is a 28 pin ZIF (Zero Insertion Force) Socket.

3.5.2 COMPONENT DESCRIPTION:

To use the Onboard programmer, the user is required to press the Function Key F1 and EXEC key that are available in the keypad, in succession. The user can then select any of the EPROMs 2716, 2732, 2732A, 2764, 2764A, 27128, 27128A, 27256 and 27256A. The programming voltages, the pin at which the voltage appears, the capacity of each EPROM and hence the number of address lines are as tabulated below.

EPROM	Programming Voltage	Pin No. in ZIF	Capacity	No. of Address Lines
2716	25 V	23	2K	11
2732	25 V	22	4K	12
2732A	21.5 V	22	4K	12
2764	21.5 V	1	8K	13
2764A	12.5 V	1	8K	13
27128	21.5 V	1	16K	14
27128A	12.5 V	1	16K	14
27256	21.5 V	1	32K	15
27256A	12.5 V	1	32K	15

PROGRAMMING MODES AND PROGRAM VERIFY:

Programming an EPROM can be accomplished in any of three ways namely the Standard Programming Algorithm, the Quick Pulse Programming Algorithm or the Intelligent Programming Algorithm. The EPROM is programmed using the algorithm which it supports. Either Standard or Quick Pulse is used.

STANDARD PROGRAMMING ALGORITHM:

The flowchart below shows the method followed in programming an EPROM using Standard Programming Algorithm.

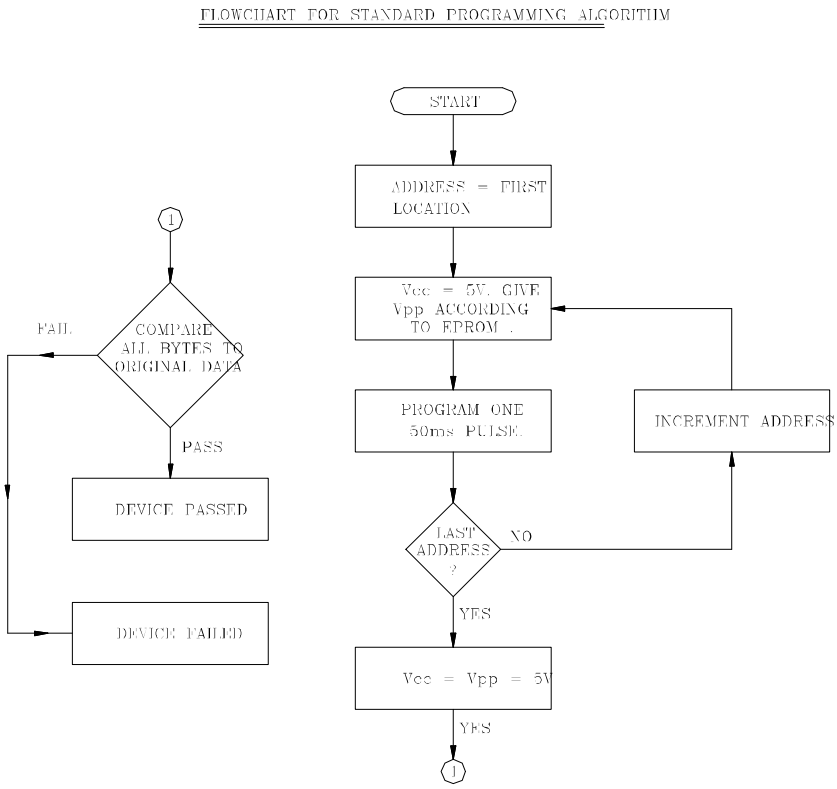


Figure F3.25

The device is in the programming mode when the OE*/Vpp or Vpp pin is at the specified programming voltage. The data to be programmed is given 8 bits in parallel to the data bus of EPROM through Port B of 8255-3. Logic high of the TTL level corresponds to 2.0V and logic low level corresponds to 0.8V.

When the address and data are stable, a 20 microseconds active low, TTL program pulse is applied to the CE* input. A program pulse is applied at each address location to be programmed, the pulse having a maximum width of 55 microseconds, either sequentially, individually or in random.

A Verify (Read) is performed on the programmed bits to determine that they are correctly programmed, with OE*/Vpp or Vpp and CE* at their TTL low level, after the falling edge of CE*.

QUICK PULSE ALGORITHM:

This algorithm substantially reduces the throughput time in the programming environment. It uses initial pulses of 100 microseconds followed by a byte verification to determine whether the address byte has been successfully programmed. Upto 25-100 microseconds pulses per byte are provided before a failure is recognized.

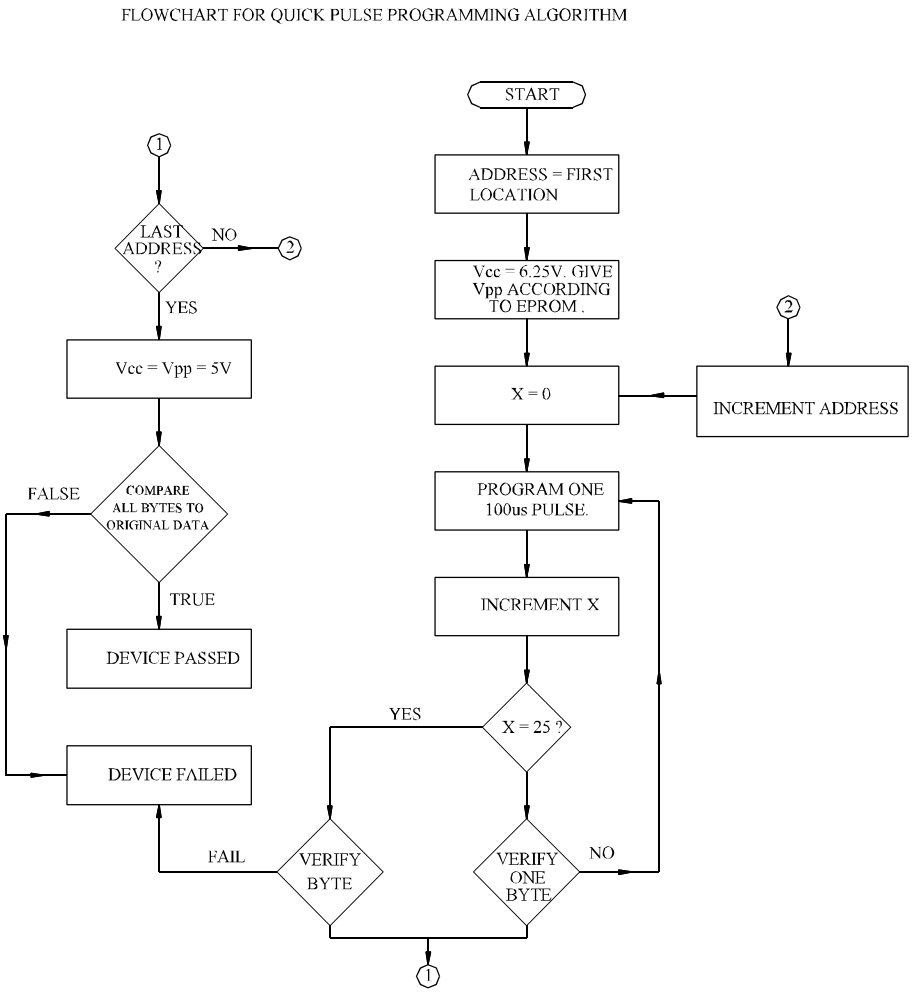


Figure F3.26

The flow chart for Quick-Pulse Programming Algorithm is shown in Figure F3.26 For this algorithm, the entire sequence of programming pulse and byte verifications are performed at V_{cc} = 6.25V and V_{pp} at 12.75V. When programming of the EPROM is completed, all bytes should be compared to the original data with $V_{cc} = V_{pp} = 5.0V$

VOLTAGE GENERATION:

Hence, the circuit must be capable of generating appropriate voltages and these voltages are derived as described below. From the +30V supply of the Linear Power Supply provided with the kit, through Zener networks, 12.5V, 22V and 27V are derived. From the +12V supply, a 5.6V and 6.3V are obtained.

The Programming voltages are derived from the Emitters of Switching transistors. The base of a Switching transistor is connected to the Output of a buffer, 74LS07. The buffer Outputs are pulled high and the buffer inputs are driven by the outputs of an octal latch.

So, if the emitter of the transistor whose collector gets 12.5V should switch to 12.5V, then, the buffer driving it should get a TTL high input. This is done by making the appropriate data line high.

ADDRESS AND DATA LINES:

The address and data lines are the Port outputs of the 8255 (U33). The Port A and Port C are configured in the output mode. The lower order address lines are derived from Port A and the higher order address lines are given through Port C. The data lines are from Port B of the 8255 (U33). When the data is given to the EPROM as in programming, then Port B acts as an Output Port. When the function selected is either reading or blank checking, then Port B is configured in input mode.

3.5.3 CIRCUIT IMPLEMENTATION:

The following Figure F3.27 is the simple block diagram of the **EPROM programming interface in Micro-51 EB.**

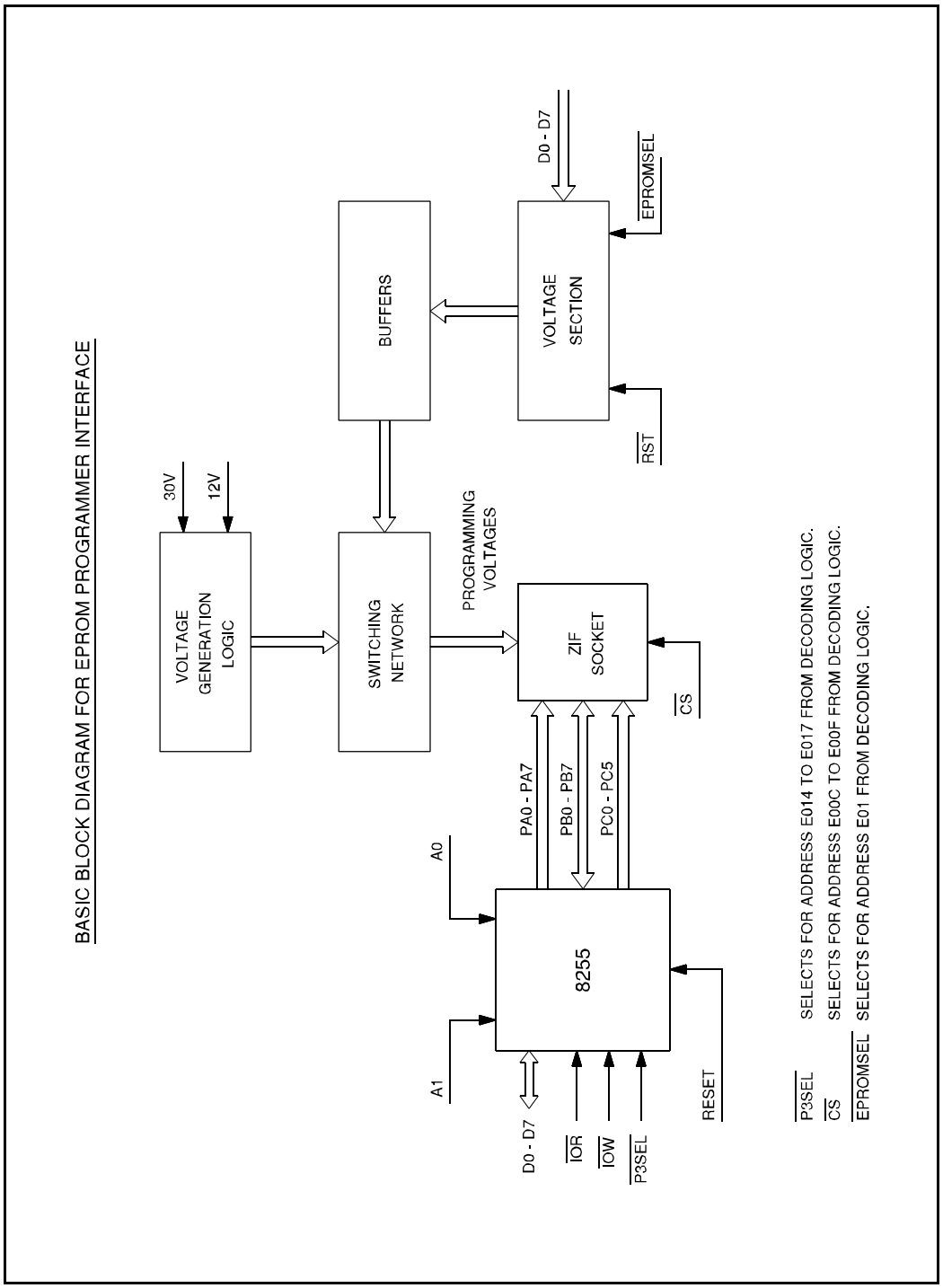

BASIC BLOCK DIAGRAM FOR EPROM PROGRAMMER INTERFACE
D0 - D7
EPROMSEL
BUFFERS
VOLTAGE SECTION
RST
30V
12V
VOLTAGE GENERATION LOGIC
SWITCHING NETWORK
PROGRAMMING VOLTAGES
ZIF SOCKET
CS
PA0 - PA7
PB0 - PB7
PC0 - PC5
A0
8255
A1
D0 - D7
IOR
IOW
P3SEL
RESET
P3SEL SELECTS FOR ADDRESS E014 TO E017 FROM DECODING LOGIC.
CS SELECTS FOR ADDRESS E00C TO E00F FROM DECODING LOGIC.
EPROMSEL SELECTS FOR ADDRESS E01 FROM DECODING LOGIC.

Figure F3.27

USING THE PROGRAMMER:

Pressing F1 and EXEC keys in succession, the user enters into the Programming Mode.

As explained in the Technical Reference Manual, the EPROM can be selected by pressing any of the hex keys from 0 to 8. Then, the functions like **Blank Check, Read or Write** can be performed.

After invoking the EPROM Programmer, the user can select any of the EPROMs from 2716 to 27256A. This selection can be done by pressing the corresponding hex key from 0 to 8 as explained in the Technical Reference Manual.

Then, the function to be performed can be selected by pressing keys from 0 to 2 for blank checking, reading or writing onto the EPROM, respectively.

The following example shows you how to Blank Check, Program and Verify the programmed contents for a 2716 EPROM.

3.5.4 EXAMPLE 8: PROGRAMMING AN EPROM (2716)

OBJECTIVE:

To Blank check, Program and Verify the contents in a 2716 EPROM.

THEORY:

Reset the System. Initially press F1 when prompted for entry and then press EXEC. Then press "0" to select 2716. Now insert the EPROM into the socket taking care to use the lower24 pins of the ZIF.

Then press **"0"** to select **"blank check"** mode of operation. Now, the trainer will ask you for the **Start and End address** of the block of EPROM to be blank checked. Upon giving these addresses as 0000 and 07FF, the trainer reads all the locations and checks whether they all contain FF. If all the locations are blank, the trainer displays **"Y"** and comes back to the Programming prompt "EPProG". Else, **"n"** will be displayed before displaying the prompt "EPProG".

Now let us copy the contents of locations 0000 to 07FF of the Monitor EPROM to the 2716 in the ZIF.

After the programming prompt "EPProG", press **"0"** for 2716 selection and **"2"** for selecting the programming function. Then, the user has to enter the **Start and End address** of the block to be programmed and the **Destination address** from where to read the data to be programmed into the EPROM. Giving all the three addresses and pressing **NEXT** causes the programmer to write into the blank locations. A data byte is written by switching to "off" state or "on"state, the flip-flop or memory cell in the EPROM. The selection of the addresses is done, programmed sequentially and the prompt is displayed, if the programming is done successfully. Now, we should verify if the EPROM has the correct data. Hence let us read the contents of the EPROM into RAM to check it with the Monitor contents.

This is achieved by performing a **"read"** operation by selecting **"1"** after "EPProG" and "0". The **Start and End addresses** are given as 0000 and 07FF and the **Destination address** is specified as 4100, which is the RAM address. After reading is completed, the Programmer Prompt comes to "EPProG". Now Reset the kit and compare the contents of source with that of the data read from EPROM.

Use the CMP command available on the keypad to compare the contents from 0000 to 07FF with 4100. Now, the trainer will come back to the Command Prompt if the EPROM has been programmed successfully.

3.5.4 EXERCISES:

i) Get the following EPROMs.
2732, 2732A, 2764, 2764A, 27128, 27128A.
Blank check all the EPROMs.

ii) a) Write a program to implement a digital clock. Configure addresses from 6000.
b) Copy your program in a 2764 EPROM and insert it into the memory socket U9.
c) Check if your digital clock program is working.

3.6 LIQUID CRYSTAL ALPHANUMERIC DISPLAY INTERFACE :

(Applicable for Micro-51 EB only)

3.6.1 COMPONENT:

In Micro-51 EB, Liquid Crystal Alphanumeric Display module arranged as 2 lines of 16 characters is used as the output device.

3.6.2 COMPONENT DESCRIPTION:

The display module includes a CMOS VLSI Microprocessor Controller which supplies character memory RAM, character generator RAM and ROM, and all refresh, control and timing signals.

3.6.3 BASIC SYSTEM INTERFACE:

Interfacing the LCD module with a microprocessor bus requires minimum hardware. We have to provide only +5VDC and parallel TTL level ASCII in order to display data. The 4 or 8 bit bus interfaces easily with most popular microprocessors with a minimum of additional hardware and allows such control functions as cursor home, cursor addressing, display shift and programmable characters to be easily implemented.

The control signals with which the LCD module communicates with the CPU are the data bus D0-D7, RS, R/W and E.

The following table shows the LCD interface pin connections.

INTERFACE PIN CONNECTIONS

Pin No.	Symbol	Function
1	VCS	Power Supply - ground
2	VDD	Power Supply +5VDC ±0.25V
3	V0	LCD Driving voltage (Viewing angle adjustment)
4	RS	Seth high to READ and WRITE alphanumeric data and character generate data. Set low to READ status and address information and to WRITE control words.
5	R/W	READ / WRITE set low to WRITE to the module. Set high to READ from the display.
6	E	Enable pulse. Data is clocked into the DAYSTAR module on the trailing edge of this pulse. Valid Busy Flag data appears on Data Bait 7, a maximum of 320ns after the leading edge of this pulse and is valid for 20ns after the trailing edge.
7 8 9 10	DB0 DB1 DB2 DB3	low order 4 lines of bi-directional tri-state data bus. Used for data transfer between the MPU and the display. These four are not used during 4-bit operation.
11 12 13 14	DB4 DB5 DB6 DB7	Higher order 4 lines of bi-directional tri-state data bus. Used for data transfer between the MPU and the display. D7 can be used as a BUSY flag.

3.6.4 DETAILED DESCRIPTION OF COMMANDS:

(i) ***Clear Display***

	RS	R/W	DB7						DB0
Code	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**

Writes ASCII "space" (20H) into all of the DD RAM. The cursor returns to Address 0(A_{DD} = "00") and the display, if it has been shifted, returns to the original position.

(ii) Return Home

	RS	R/W	DB7						DB0
Code	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	*****

* (Dont Care)

Returns the cursor to Address 0 (A_{DD} = "00") and display, if it has been shifted, to the original position. The DD RAM contents remain unchanged.

(iii) Entry Mode Set

	RS	R/W	DB7						DB0
Code	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**I / D**	**S**

* (Dont Care)

I/D : This mode automatically increments (I/D=1) or decrements (I/D=0) THE DD RAM address by one every time a character is written to or read from the DD RAM. (The cursor moves to the right when incremented.) The same applies to writing and reading of CG RAM.

S : Automatically shifts the entire display to either the right or the left after each character is written, if S=1; either to the left (when I/D=1), or to the right (when I/D=0). Therefore, the cursor looks as if it stood still and only the display moved. The display is not shifted when reading from the DD RAM, or if S=0.

(iv) Display ON/OFF Control (Display Blanking)

	RS	R/W	DB7						DB0
Code	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**D**	**C**	**B**

* (Dont Care)

D : The display is turned ON when D=1 and OFF when D=0. When the display is turned OFF by D=0,the display data remains in the DD RAM and it can be displayed immediately by setting D=1.

C : The cursor is displayed as an under bar when C=1 and not displayed when C=0. Even if the cursor is not displayed, the function of I/D,etc. does not change during display data write. The cursor is displayed using 5 dots in either the 8th line when the 5x7 dot character font is selected or in the 11th line when 5x10 dot character font is selected .

B : The character at the cursor position blinks when B=1. The blink is done by switching between all block dots and the displayed character at 0.4 second intervals. The cursor underbar and the blink can be set concurrently.

(v) Cursor or Display shift

RS R/W DB7 ----------DB0

Code	**0**	**0**	**0**	**0**	**0**	**1**	**S / C**	**R / L**	*****	*****

* (Dont Care)

Shifts the cursor position or display to the right and the left without writing or reading the display data. This function is used for correction or search of display.

S/C	R/L	
0	0	Shifts the cursor position to the left. (AC is decremented by one.)
0	1	Shifts the cursor position to the right. (AC is incremented by one.)
1	0	Shifts the entire display to the left. The cursor follows the display shift.
1	1	Shifts the entire display to the right. The cursor follows the display shift.

(vi) Function set

RS R/W DB7 ----------DB0

Code	**0**	**0**	**0**	**0**	**1**	**DL**	**N**	**F**	*****	*****

* (Dont Care)

DL : Sets interface data length. Data is sent or received in 8-bit length (DB7 through DB0) when DL=1 and 4-bit length (DB7 through DB4) when DL=0. When 4-bit length is selected, data must be written or read twice for each 8-bit byte of data.

N : Sets number of display lines.
F : Sets character font.

N	F	No of Display Lines	Character Font	Duty Factor	Remarks
0	0	1	5 × 7 dots	1 / 8	
0	1	1	5 × 10 dots	1 / 11	
1	*	2	5 × 7 dots	1 / 16	Cannot display 2 lines with 5 × 10 dot character font

(vii) Set CG RAM Address

	RS	R/W	DB7						DB0
Code	**0**	**0**	**0**	**1**	**A**	**A**	**A**	**A**	**A**	**A**

* (Dont Care)

Sets the CG RAM address as a binary number "AAAAAA" in the address counter. Data pertaining to the CG RAM is written to or read from the host system after this command is executed.

(viii) Set DD RAM Address

	RS	R/W	DB7						DB0
Code	**0**	**0**	**1**	**A**	**A**	**A**	**A**	**A**	**A**	**A**

* (Dont Care)

Sets the DD RAM address as a binary number "AAAAAAA" in the address counter. Data pertaining to the DD RAM is written or read from the host system after this command is executed.

When N=0 (1-line display), the value of "AAAAAAA" is "00" through "4F" (Hex). When N=1 (2-line display), the value of "AAAAAAA" is "00" through "27" (Hex) for the first line, and "40"through "67" (Hex) for the second line.

(ix) Read Busy Flag and Address

	RS	R/W	DB7						DB0
Code	**0**	**1**	**BF**	**A**	**A**	**A**	**A**	**A**	**A**	**A**

* (Dont Care)

Reads the Busy Flag (BF), which indicates that the system is internally operating on an instruction received previously. When BF=1, it indicates that internal operation is going on and the next instruction is not accepted until BF returns to "0". Check the BF status before the next write operation.

At the same time, this value of the address counter is expressed in a binary number "AAAAAAA". The address counter is used for both CG and DD RAM address, and its value is determined by the previous instruction. Address contents are the same as Items (7) and (8).

(x) Write Data to CG or DD RAM

	RS	R/W	DB7 ----------DB0							
Code	**1**	**0**	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**D**
			High Order Bits					Low Order Bits		

Writes binary 8 bit data "DDDDDDDD" to the CG or the DD RAM. Whether the CG or the DD RAM is to be written is determined by the previous command. (CG RAM address setting or DD RAM address setting). After writing, the address is automatically incremented or decremented by one according to entry mode. Display shift also follows the entry mode.

(xi) Read Data from CG or DD RAM

	RS	R/W	DB7----------DB0							
Code	**1**	**1**	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**D**
			High Order Bits					Low Order Bits		

Reads binary 8-bit data "DDDDDDDD" from the CG or the DD RAM. Whether the CG RAM or the DD RAM is to be read is determined by the previous instruction. PRIOR TO INPUTTING **READ** INSTRUCTIONS, EITHER THE **CG RAM ADDRESS SET** INSTRUCTION OR THE **DD RAM ADDRESS SET** INSTRUCTION MUST BE EXECUTED.

After reading data, the address is automatically incremented or decremented by one according to the entry mode. However, display shift is not performed regardless of entry mode setting.

CHARACTER GENERATOR ROM(CG ROM)*:*

The character generator ROM generates character patterns of 5x7 dots or 5x10 dots from 8-bit character codes. It can generate 160 5x7 dot character patterns and 32 5x10 dot character patterns. APPENDIX E shows the correspondence between character codes and character patterns.

USER PROGRAMMABLE CHARACTER GENERATOR RAM (CG RAM):

The character generator RAM is the RAM to which the user can write programmable character patterns. If the display uses 5x7 dots, 8 character patterns can be written; if it uses 5x10 dots, 4 may be programmed. Write the character codes listed in the left edges of APPENDIX E to display character patterns stored in CG RAM.

APPENDIX E shows relation between CG RAM addresses and data and display patterns. As shown in APPENDIX E, area that is not used for display can be used as general data RAM.

BUSY FLAG:

When data or commands are written to the Display Module it requires a certain period to place the data in the correct location or to execute the desired command. The time required to complete this process and prepare for a new character or command is the appropriate Execution Time. The Character Load Execution Time determines the maximum speed in which characters can be input (the Character Loading Rate). No data should be sent to the module until it has processed a previously entered character. To obtain the maximum character loading rate, the Busy Flag (Data Bit 7) should be read to determine when the module is ready to accept the next character.

Character loading can also be accomplished while ignoring the status of the Busy Flag at a slight loss in loading rate. In this method, characters may be entered at a rate determined by the user to be less than the Display Module's Character Loading Rate.

PROGRAMMING SEQUENCE:

From the above discussion it should be clear that the sequence involved in displaying characters in the LCD module requires proper loading of the following commands.

i) Function Set

ii) Clear Display

iii) Entry Mode Set

iv) Cursor or Display Shift

v) Set DD RAM Address

vi) Write Data to DD RAM

3.6.5 CIRCUIT IMPLEMENTATION:

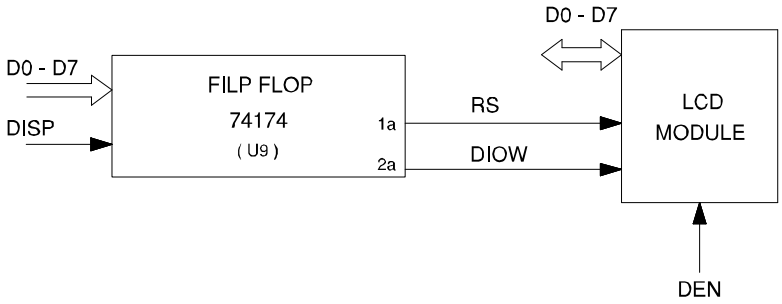


Figure F3.28

DEN is the Enable pulse to LCD which selects for address FF04H. Read/Write (DIOW) and register select (RS) control lines of LCD are provided by the Flipflop (U_9) whose address is FF08H.

3.6.6 EXAMPLE 6 - DISPLAY A STRING IN LCD:

OBJECTIVE:

To initialize LCD and to display the string " VI MICROSYSTEMS " in the first row of the display.

THEORY:

The LCD module must be sent data to set the desired functions, before sending the characters for displaying. Before reading or writing data command from /to the module the BUSY flag must be read to determine when the module is ready to accept the next character/command.

PROGRAM:

```
5000                    ORG     5000H
FF 04       DEN         EQU     0FF04H
00 04       DENL        EQU     04H
FF 08       LATCH       EQU     0FF08H
00 08       LATCHL      EQU     08H
00 FF       IOHIGH      EQU     0FFH
5000            START:
5000  12 50 39          CALL    BSYCHK
5003  90 FF 04          MOV     DPTR,#DEN
5006  74 38              MOV     A,#38H
5008  F0                MOVX    @DPTR,A     ;Function set
5009  12 50 39          CALL    BSYCHK
500C  74 01              MOV     A,#01
```

```
500E  F0              MOVX   @DPTR,A    ;Clear display
500F  12 50 39        CALL   BSYCHK
5012  74 06              MOV     A,#06
5014  F0              MOVX   @DPTR,A    ;Entry mode set
5015  12 50 39        CALL   BSYCHK
5018  74 0F           MOV    A,#0FH
501A  F0              MOVX   @DPTR,A    ;Display control
501B  12 50 39        CALL   BSYCHK
501E  74 80              MOV     A,#80H
5020  F0              MOVX   @DPTR,A    ;Set DDRAM address
5021  90 50 52        MOV    DPTR,#LOOKUP
5024  7A 0F           MOV    R2,#0FH
5026            MORE:
5026  12 50 39        CALL   BSYCHK
5029  74 01              MOV     A,#01
502B  75 A0 FF        MOV    P2,#IOHIGH
502E  F2              MOVX   @R0,A      ;RS =1,R/W= 0
502F  E0              MOVX   A,@DPTR
5030  75 A0 FF        MOV    P2,#IOHIGH
5033  F3              MOVX   @R1,A      ;Write data to DDRAM
5034  A3              INC    DPTR
5035  DA EF           DJNZ   R2,MORE
5037  80 FE      HLT: SJMP   HLT
5039            BSYCHK:
5039  79 04              MOV     R1,#DENL
503B  78 08              MOV     R0,#LATCHL
503D  75 A0 FF        MOV    P2,#IOHIGH
5040  74 02              MOV     A,#02          ;RS = 0,R/W=1
5042  F2              MOVX   @R0,A
5043            BSY:
5043  75 A0 FF        MOV    P2,#IOHIGH
5046  E3              MOVX   A,@R1 ;Check busy flag
5047  54 80              ANL     A,#80H
5049  70 F8           JNZ    BSY
504B  75 A0 FF        MOV    P2,#IOHIGH
504E  74 00              MOV     A,#00          ;RS = 0,R/W=0
5050  F2              MOVX   @R0,A
5051  22              RET
5052  56 49 20 4D  LOOKUP:    DB     'VI MICROSYSTEMS '
5056  49 43 52 4F
505A  53 59 53 54
505E  45 4D 53 20
5062                    END
```

VERIFICATION:

Enter the program and the lookup table. Execute the program and see that the message " VI MICROSYSTEMS " is displayed.

3.6.7 EXERCISES:

i) Display a rolling message "You and I are friends" using display shift option.
ii) Blink the message "Hello".
iii) Display the symbol " ".(Hint: Use character generator RAM).

3.7 IBM PC-AT KEYBOARD INTERFACE:

The IBM PC-AT Keyboard Interface is centered around a 5-pin DIN connector used for connecting the 84-keys IBM PC Keyboard or 101-keys IBM PC AT/XT keyboard. The keyboard is operated in AT mode.

3.7.1 KEYBOARD CONTROLLER:

The keyboard controller is a single-chip microcomputer (Intel 8042) that is programmed to support the IBM Personal Computer AT Keyboard serial interface. The keyboard controller receives serial data from the keyboard, checks the parity of the data, translates scan codes, and presents the data to the system as a byte of data in its output buffer. The status register contains bits that indicate if an error was detected while receiving the data. Data may be sent to the keyboard by writing to the keyboard controller's input buffer. The byte of data will be sent to the keyboard serially with an odd parity bit automatically inserted. The keyboard is required to acknowledge all data transmissions. No transmission should be sent to the keyboard until acknowledgment is received for the previous byte sent.

Receiving Data from the Keyboard

The keyboard sends data in a serial format using an 11-bit frame. The first bit is a start bit, and is followed by eight data bits, an odd parity bit, and a stop bit. Data sent is synchronized by a clock supplied by the keyboard. At the end of transmission, the keyboard controller disables the interface until the system accepts the byte. If the byte of data is received with a parity error, a Resend command is automatically sent to the keyboard. If the keyboard controller is unable to receive the data correctly, a hex FF is placed in its output buffer, and the parity error. The keyboard controller will also time a byte of data from the keyboard. If a keyboard transmission does not end within two milliseconds, a hex FF is placed in the keyboard controller's output buffer, and the receive time-out bit in the status register is set. No retries will be attempted on a receive time-out error.

Scan Code Translation

Scan codes, which are received from the keyboard, are converted by the keyboard controller before they are put into the controller's output buffer. The keys, keyboard scan code and system scan code are given in Appendix.

Sending Data to the Keyboard

Data is sent to the keyboard in the same serial format used to receive data from the keyboard. A parity bit is automatically inserted by the keyboard controller. If the keyboard does not start clocking the data out of the keyboard controller within 15 milliseconds or complete that clocking within 2 milliseconds, a hex FE is placed in the keyboard controller's output buffer, and the transmit time-out and parity error bits are set in the status register. The keyboard controller is programmed to set a time limit for the keyboard to respond. If 25 milliseconds are exceeded, the keyboard controller places a hex FE in its output buffer and sets the transmit and receive time-out error bits in the status register. No retries will be made by the keyboard controller for any transmission error.

Keyboard Controller System Interface

The keyboard controller communicates with the Microprocessor through a status register, an output buffer, and an input buffer. The following figure is a block diagram of the keyboard interface.

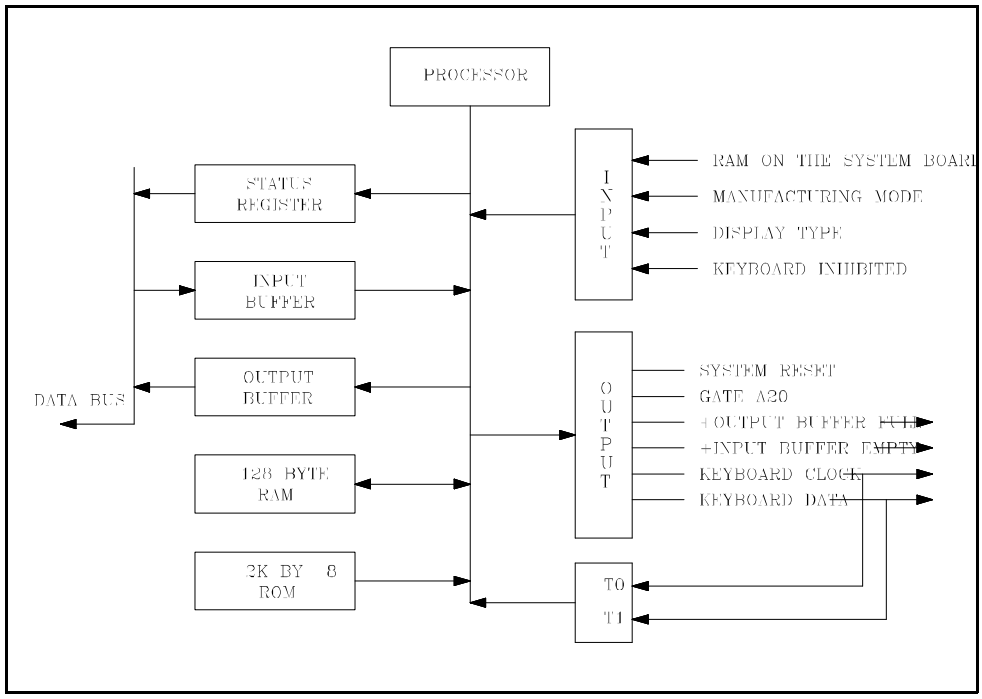


Figure 3-28A

Status Register

The status register is an 8-bit read-only register at address hex FF1C. It has information about the state of the keyboard controller (8042) and interface. It may be read at any time.

Status-Register Bit Definition

Bit 0 Output Buffer Full - A 0 indicates that the keyboard controller's output buffer has no data. A 1 indicates that the controller has placed data into its output buffer but the system has not yet read the data. When the CPU reads the output buffer (address hex FF18), this bit will return to a 0.

Bit 1 Input Buffer Full - A 0 indicates that the keyboard controller's input buffer (address hex FF18 or FF1C) is empty. A 1 indicates that data has been written into the buffer but the controller has not read the data. When the controller reads the input buffer, this bit will return to 0.

Bit 2 System Flag - This bit may be set to 0 or 1 by writing to the system's flag bit in the keyboard controller's command byte. It is set to 0 after a power on rest.

Bit 3 Command/Data - The keyboard controller's input buffer may be addressed as either address hex FF18 or FF1C. Address hex FF18 is defined as the data port, and address hex FF1C is defined as the command port. Writing to address hex FF1C sets this bit to 1; writing to address hex FF18 sets this bit to 0. The controller uses this bit to determine if the byte in its input buffer should be interpreted as a command byte or a data byte.

Bit 4 Inhibit Switch - This bit is updated whenever data is placed in the keyboard controller's output buffer. It reflects the state of the keyboard-inhibit switch. A 0 indicates the keyboard is inhibited.

Bit 5 Transmit Time-Out - A 1 indicate that a transmission started by the keyboard controller was not properly completed. If the transmit byte was not clocked out within the specified time limit, this will be the only error. If the transmit byte was clocked out but a response was not received within the programmed time limit, the transmit time-out and receive time-out error bits are set On. If the transmit byte was clocked out but the response was received with a parity error, the transmit time-out and parity error bits are set On.

Bit 6 Receive Time-Out - A 1 indicates that a transmission was started by the keyboard but did not finish within the programmed receive time-out delay.

Bit 7 Parity Error - A 0 indicates the last byte of data received from the keyboard had odd parity. A 1 indicates the last byte had even parity. The keyboard should send with odd parity.

Output Buffer

The output buffer is an 8-bit read-only register at address hex FF18. The keyboard controller uses the output buffer to send scan codes received from the keyboard, and data bytes requested by commands to the CPU. The output buffer should be read only when the output buffer full bit in the status register is 1.

Input Buffer

The input buffer is an 8-bit write-only register at address FF18 or FF1C. Writing to address hex FF18 sets a flag, that indicates a data write; writing to address hex FF1C sets a flag, indicating a command write. Data written to address hex FF18 is sent to the keyboard, unless the keyboard controller is expecting a data byte following a controller command. Data should be written to the controller's input buffer only if the input buffer full bit in the status register is equal to 0. The following are valid keyboard controller commands.

Commands (Address hex FF1C)

20 Read Keyboard Controller's Command Byte

The controller sends its current command byte to its output buffer.

60 Write Keyboard Controller's Command Byte

The next byte of data written to address hex FF18 is placed in the controller's command byte. Bit definitions of the command byte are as follows:

Bit 7 Reserved - Should be written to a 0.

Bit 6 IBM Personal Computer Compatibility Mode - Writing a 1 to this bit causes the controller to convert the scan codes received from the keyboard to those used by the IBM Personal Computer. This includes converting a two-byte break sequence to the one-byte IBM Personal Computer format.

Bit 5 IBM Personal Computer Mode - Writing a 1 to this bit programs the keyboard to support the IBM Personal Computer keyboard interface. In this mode the controller does not check parity or convert scan codes.

Bit 4 Disable Keyboard - Writing a 1 to this bit disables the keyboard interface by driving the `clock' line low. Data is not sent or received.

Bit 3 Inhibit Override - Writing a 1 to this bit disables the keyboard inhibit function.

Bit 2 System Flag - The value written to this bit is placed in the system flag bit of the controller's status register.

Bit 1 Reserved - Should be written to a 0.

Bit 0 Enable Output - Buffer - Full Interrupt - Writing a 1 to this bit causes the controller to generate an interrupt when it places data into its output buffer.

AA Self-Test - This commands the controller to perform internal diagnostic tests. A hex 55 is placed in the output buffer if no errors are detected.

AB Interface Test - This commands the controller to test the keyboard clock and data lines. The test result is placed in the output buffer as follows:

00 No error detected.

01 The `keyboard clock' line is stuck low.

02 The `keyboard clock' line is stuck high.

03 The `keyboard data' line is stuck low.

04 The `keyboard data' line is stuck high.

AC Diagnostic Dump - Sends 16 bytes of the controller's RAM, the current state of the input port, the current state of the output port, and the controller's program status word to the system. All items are sent in scan-code format.

AD Disable Keyboard Feature - This command sets bit 4 of the controller's command byte. This disables the keyboard interface by driving the clock line low. Data will not be sent or received.

AE Enable Keyboard Interface - This command clears bit 4 of the command byte, which releases the keyboard interface.

C0 Read Input Port - This commands the controller to read its input port and place the data in its output buffer. This command should be used only if the output buffer is empty.

D0 Read Output Port - This command causes the controller to read its output port and place the data in its output buffer. This command should be issued only if the output buffer is empty.

D1 Write Output Port - The next byte of data written to address hex FF18 is placed in the controller's output port.

E0 Read Test Inputs - This command causes the controller to read its T0 and T1 inputs. This data is placed in the output buffer. Data bit 0 represents T0, and data bit 1 represents T1.

F0-FF Pulse Output Port - Bits 0 through 3 of the controller's output port may be pulsed low for approximately 6 microseconds. Bits 0 through 3 of this command indicate which bits are to be pulsed. A 0 indicates that the bit should be pulsed, and a 1 indicates the bit should not be modified.

I/O Ports

The keyboard controller has two 8-bit I/O ports and two test inputs. One of the ports is assigned for input and the other for output. The controller uses the test inputs to read the state of the keyboard's `clock' line and the keyboard's `data' line.

KEYBOARD INTERFACE:

The keyboard uses a bidirectional serial interface to carry signals between the keyboard and system unit.

Sequencing Key Code Scanning

The keyboard is able to detect all keys that are pressed, and their scan codes will be sent to the interface in correct sequence, regardless of the number of keys held down. Keystrokes are stored only when the keyboard is not serviced by the system.

Keyboard Buffer

The keyboard has a 16-character first-in-first-out (FIFO) buffer where data is stored until the interface is ready to receive it.

A buffer-overrun condition will occur if more than sixteen codes are placed in the buffer before the first keyed data is sent. The seventeenth code will be replaced with the overrun code, hex 00. The 17th position is reserved for overrun codes). If more keys are pressed before the system allows a keyboard output, the data will be lost. When the keyboard is allowed to send data, the characters in the buffer will be sent as in normal operation, and new data entered will be detected and sent.

Keys

All keys are classified as make/break, which means when a key is pressed, the keyboard sends a make code for that key to the keyboard controller. When the key is released, its break code is sent (the break code for a key is its make code preceded by hex F0).

All keys are typematic. When a key is pressed and held down, the keyboard continues to send the make code for that key until the key is released. The rate at which the make code is sent is known as the typematic rate. When two or more keys are held down, only the last key pressed repeats at the typematic rate. Typematic operation stops when the last key pressed is released even if other keys are still held down. When a key is pressed and held down while the interface is inhibited, only the first make code is stored in the buffer. This prevents buffer overflow as a result of typematic action.

3.7.2 FUNCTIONS PERFORMED AT POWER-ON TIME

Power-On Reset

The keyboard logic generates a POR when power is applied to keyboard. The POR lasts a minimum of 300 milliseconds and maximum of 9 seconds.

Basic Assurance Test

Immediately following the POR, the keyboard executes a basic assurance test (BAT). This test consists of a checksum of all read-only memory (ROM), and a stuck-bit and addressing test of all random-access memory (RAM) in the keyboard's microprocessor. The mode indicators - three light emitting diodes (LEDs) on the upper right-hand corner of the keyboard - are turned on then off, and must be observed to ensure they are operational.

Execution of the BAT will take from 600 to 900 milliseconds. (This is in addition to the time required for the POR).

The BAT can also be started by a Reset command.

After the BAT, and when the interface is enabled ("clock" and "data" lines are set high), the keyboard sends a completion code to the interface - either hex AA for satisfactory completion or hex FC (or any other code) for a failure. If the system issues a Resend command, the keyboard sends the BAT completion code again. Otherwise, the keyboard sets the keys to typematic and make/break.

Commands from the system

The commands described below may be sent to the keyboard at any time. The keyboard will respond within 20 milliseconds.

Note: The following commands are those sent by the system. They have a different meaning when issued

by the keyboard.

Reset (Hex FF)

The system issues a Reset command to start a program reset and a keyboard internal self-test. The keyboard acknowledges the command with an "acknowledge" signal (ACK) and ensures the system accepts the "ACK" before executing the command. The system signals acceptance of the "ACK" by raising the clock and data for a minimum of 500 microseconds. The keyboard is disabled from the time it receives the Reset command until the "ACK" is accepted or until another command overrides the previous one. Following acceptance of the "ACK", the keyboard begins the reset operation, which is similar to a power-on reset. The keyboard clears the output buffer and sets up default values for typematic and delay rates.

Resend (Hex FE)

The system can send this command when it detects an error in any transmission from the keyboard. It can be sent only after keyboard transmission and before the system enables the interface to allow the next keyboard output. Upon receipt of Resend, the keyboard sends the previous output again unless the previous output was Resend. In this case, the keyboard will resend the last byte before the Resend command.

No-operation (NOP) (Hex FD through F7)

These commands are reserved and are effectively no-operation or NOP. The system does not use these codes. If sent, the keyboard will acknowledge the command and continue in its prior scanning state. No other operation will occur.

Set Default (Hex F6)

The Set Default command resets all conditions to the power-on default state. The keyboard responds with "ACK", clears its output buffer, set default conditions, and continues scanning (only if the keyboard was previously enabled).

Default Disable (Hex F5)

This command is similar to Set Default, except the keyboard stops scanning and awaits further instructions.

Enable (Hex F4)

Upon receipt of this command, the keyboard responds with "ACK", clears its output buffer, and starts scanning.

Set Typematic Rate/Delay (Hex F3)

The system issues this command, followed by a parameter, to change the typematic rate and delay. The typematic rate and delay parameters are determined by the value of the byte following the command. Bits 6 and 5 serve as the delay parameter and bits 4,3,2,1, and 0 (the least-significant bit) are the rate parameter. Bit 7, the most-significant bit, is always 0. The delay is equal to 1 plus the binary value of bits 6 and 5 multiplied by 250 milliseconds ±20%. The period (interval from one typematic output to the next) is determined by the following equation.
Period = (8 + A) X (2^B) X 0.00417 seconds, where A = binary value of bits, 2,1, and 0 and B = binary value of bits 4 and 3.

The typematic rate (make code per second) is 1/period. The period is determined by the first equation above. The following table results.

Bit Rate	Bit Rate
00000 30.0	10000 7.5
00001 26.7	10001 6.7
00010 24.0	10010 6.0
00011 21.8	10011 5.5
00100 20.0	10100 5.0
00101 18.5	10101 4.6
00110 17.1	10110 4.3
00111 16.0	10111 4.0
01000 15.0	11000 3.7
01001 13.3	11001 3.3
01010 12.0	11010 3.0
01011 10.9	11011 2.7
01100 10.0	11100 2.5
01101 9.2	11101 2.3
01110 8.6	11110 2.1
01111 8.0	11111 2.0

Typematic Rate

The keyboard responds to the Set Typematic Rate Delay command with an "ACK", stops scanning, and waits for the rate parameter. The keyboard responds to the rate parameter with another "ACK", sets the rate and delay, and continues scanning (if the keyboard was previously enabled). If a command is received instead of the rate parameter, the set-typematic-rate function ends with no change to the existing rate, and the new command is processed. However, the keyboard will not resume scanning unless instructed to do so by an Enable command.

The default rate for the system keyboard is as follows:
The typematic rate = 10 characters per second ±20% and the delay = 500 ms ±20%.

No-operation (NOP) (Hex F2 through EF)

These commands are reserved and are effectively no-operation (NOP). The system does not use these codes. If sent, the keyboard acknowledges the command and continues in its prior scanning state. No other operation will occur.

Echo (Hex EE)

Echo is a diagnostic aide. When the keyboard receives this command, it issues a hex EE response and continues scanning if the keyboard was previously enabled.

Set/Reset Mode Indicators (Hex ED)

Three mode indicators on the keyboard are accessible to the system. The keyboard activates or deactivates these indicators when it receives a valid command from the system. They can be activated or deactivated in any combination.

It is up to the using system to remember the previous state of an indicator. This is in case its setting does not change when a command sequence is issued to change the state of another indicator. The system remembers the previous state of an indicator so that its setting does not change when a command sequence is issued to change the state of another indicator.

The command has the following format:

Command	Options

A Set/Reset Mode Indicators command consists of two bytes. The first is the command byte and has the following bits setup:

11101101 - hex ED

The second byte is an option byte. It has a list of the indicators to be acted upon. The format of the option byte is as follows:

Bit 7 Reserved
Bit 6 Reserved
Bit 5 Reserved
Bit 4 Reserved
Bit 3 Reserved
Bit 2 Caps Lock indicator
Bit 1 Numeric Lock indicator
Bit 0 Scroll Lock indicator

Note:

Bit 7 is the most-significant bit; bit 0 is the least-significant. The keyboard will respond to the Set/Reset Mode Indicators command with an "ACK", discontinue scanning, and wait for the option byte. The keyboard will respond to the option byte with an ACK, set the indicators, and continue scanning if the keyboard was previously enabled. If another command is received in place of the option byte, execution of the function of the Set/Reset Mode Indicators command is stopped with no change to the indicator states, and the new command is processed. Then scanning is resumed.

Command Codes to the System

The command codes described here are those sent by the keyboard. The codes have a different meaning when issued by the system.

Resend (Hex FE)

The keyboard issues a Resend command following receipt of an invalid input, or any input with incorrect parity. If the system sends nothing to the keyboard, no response is required.

ACK (Hex FA)

The keyboard issues an "ACK" response to any valid input other than an Echo or Resend command. If the keyboard is interrupted while sending "ACK", it will discard "ACK" and accept and respond to the new command.

Overrun (Hex 00)

An overrun character is placed in position 17 of the keyboard buffer, overlaying the last code if the buffer becomes full. The code is sent to the system as an overrun when it reaches the top of the buffer.

Diagnostic Failure (Hex FD)

The keyboard periodically tests the sense amplifier and sends a diagnostic failure code if it detects any problems. If a failure occurs during BAT, the keyboard stops scanning and waits for a system command or power-down to restart. If a failure is reported after scanning is enabled, scanning continues.

Break Code Prefix (Hex F0)

This code is sent as the first byte of a 2-byte sequence to indicate the release of a key.

BAT completion Code (Hex AA)

Following satisfactory completion of the BAT, the keyboard sends hex AA. Hex FC (or any other code) means the keyboard microprocessor check failed.

ECHO Response (Hex EE)

This is sent in response to an Echo command from the system.

Clock and Data Signals

The keyboard and system communicate over the "clock" and "data" lines. The source of each of these lines is an open-collector device on the keyboard that allows either the keyboard or the system to force a line to a negative level. When no communication is occurring, both the "clock" and "data" lines are at a positive level.

Data transmissions to and from the keyboard consist of 11-bit data streams that are sent serially over the serial data line. The following figure shows the structure of the data stream.

Bit	Function
1st Bit	0 Start Bit
2nd Bit	Data Bit 0 (Least Significant)
3rd Bit	Data Bit 1
4thBit	Data Bit 2
5hBit	Data Bit 3
6hBit	Data Bit 4
7hBit	Data Bit 5
8hBit	Data Bit 6
9Bit	Data Bit 7 (Most Significant)
10Bit	Parity Bit (Odd Parity)
11th Bit	Stop Bit.

Transmission Data Stream

The parity bit is either 1 or 0, and the eight data bits, plus the parity bit, always have an odd number. When the system sends data to the keyboard, it forces the "data" line to a negative level and allows the "clock" line to go to a positive level.

When the keyboard sends data to, or receives data from the system, it generates the "clock" signal to time the data. The system can prevent the keyboard from sending data by forcing the "clock" line to a negative level; the "data" line may go high or low during this time.

During the BAT, the keyboard allows the "clock" and "data" lines to go to a positive level.

Keyboard Data Output

When the keyboard is ready to send data, it first checks for a keyboard-inhibit or system request-to-send status on the "clock" and "data" lines. If the "clock" line is low (inhibit status), data is stored in the keyboard buffer. If the "clock" line is high and "data" is low (request-to-send), data is stored in the keyboard buffer, and the keyboard receives system data.

If "clock" and "data" are both high, the keyboard sends the 0 start bit, 8 data bits, the parity bit and the stop bit. Data will be valid before the falling edge and beyond the rising edge of "clock". During transmission, the keyboard checks the "clock" line for a positive level at least every 60 milliseconds. If the system lowers the "clock" line from a positive level after the keyboard starts sending data, a condition known as line contention occurs, and the keyboard stops sending data. If line contention occurs before the rising edge of the tenth clock (parity bit), the keyboard buffer returns the "data" and "clock" lines to a positive level. If contention does not occur by the tenth clock, the keyboard completes the transmission.

Following a transmission, the system can inhibit the keyboard until the system processes the input or until it requests that a response be sent.

Keyboard Data Input

When the system is ready to send data to the keyboard, it first checks if the keyboard is sending data. If the keyboard is sending but has not reached the tenth clock, the system can override the keyboard output by forcing the "clock" line to a negative level. If the keyboard transmission is beyond the tenth clock, the system must receive the transmission. If the keyboard is not sending, or if the system elects to override the keyboard"s output, the system forces the "clock" line to a negative level for more than 60 microseconds while preparing to send. When the system is ready to send the start bit ("data" line will be low), it allows the "clock" line to go to a positive level.

The keyboard checks the state of the "clock" line at intervals of no less than 60 milliseconds. If a request-to-send is detected, the keyboard counts 11 bits. After the tenth bit, the keyboard forces the "data" line low and counts one more (the stop bit). This action signals the system that the keyboard has received its data. Upon receipt of this signal, the system returns to a ready state, in which it can accept keyboard output, or goes to the inhibited state until it is ready.

Each system command or data transmission to the keyboard required a response from the keyboard before the system can send its next output. The keyboard will respond within 20 milliseconds unless the system prevents keyboard output. If the keyboard response is invalid or has a parity error, the system sends the command or data again. A Resend command should not be sent in this case.

The configuration of 5-pin DIN Connector provided in the IBM PC keyboard is given in Figure F3.29. KVcc and Gnd are usually connected to system Vcc and Gnd. The RST pin is not connected to any of the system signals.

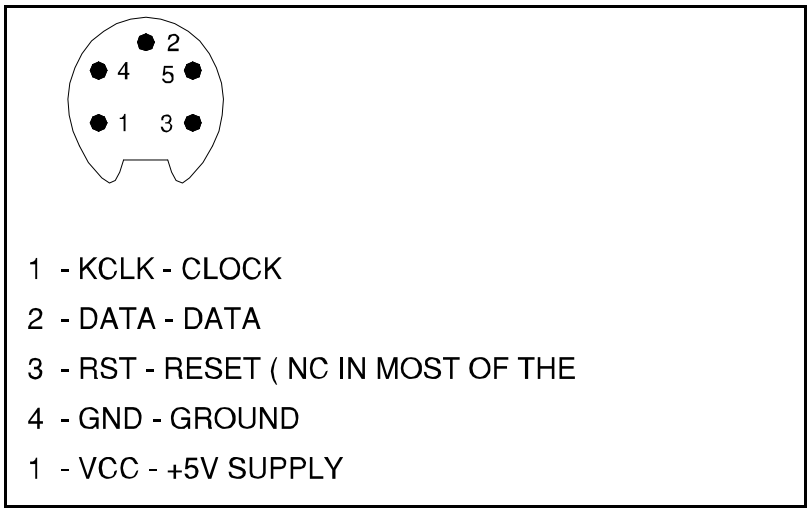


Fig F3.29 IBM PC Keyboard Connector Configuration

The following table lists the addresses of keyboard controller's buffers.

IC No	Function	Read / Write	Address in hex
U_{12}	Status Register	Read Only	FF1C
U_{12}	Output Buffer	Read Only	FF18
U_{12}	Input Buffer (Data)	Write Only	FF18
U_{12}	Input Buffer (Command)	Write Only	FF1C

3.7.3 PROGRAMMING DESCRIPTION:

When a key is pressed in the IBM PC keyboard, if enabled, it sends the scan code of the key pressed serially in synchronization with the clock signal. The keyboard controller receives the data and store it in its output buffer and set the output buffer full bit in the status register. The scan code can be read from the controller's output buffer after checking the output buffer full bit in the status register.

3.7.4 PROGRAMMING SEQUENCE:

To read a key from the keyboard, do the following

i) Check whether the output buffer is full, by reading the status register bit "0".
ii) If the output buffer full bit is "high" then read the scan code from the output register.

3.7.5 EXAMPLE - 7 READ A KEY FROM THE KEYBOARD:

OBJECTIVE: To read the scan code of the key pressed in the keyboard and store it at location 5100H.

PROGRAM:

```
5000                        ORG     5000H
FF 1C   KEYSTAT             EQU     0FF1CH
FF 18   OUTBUF              EQU     0FF18H
5000            START:
5000    12 50 16            CALL    KEYRDY
5003    90 FF 18            MOV     DPTR,#OUTBUF
5006    E0                  MOVX    A,@DPTR
5007    F5 F0               MOV     B,A
5009    54 80               ANL     A,#80H
500B    B4 00 F2            CJNE    A,#00,START     ;SKIP BREAK CODE
500E    E5 F0               MOV     A,B
5010    90 51 00            MOV     DPTR,#5100H
5013    F0                  MOVX    @DPTR,A
5014    01 14       STOP:   JMP     STOP
5016            KEYRDY:
5016    90 FF 1C            MOV     DPTR,#KEYSTAT
5019            WAIT:
5019    E0                  MOVX    A,@DPTR
501A    54 01               ANL     A,#01
501C    B4 01 FA            CJNE    A,#01,WAIT
501F    22                  RET
5020                        END
```

VERIFICATION:

After entering the program given above and executing it, press any key in the keyboard. Verify that the corresponding scan code of the key which you have pressed, is stored at location 5050 using substitute command. Refer Appendix F for the scan code.

3.7.6 EXERCISES:

i) Write a program to input keys from the keyboard and display the corresponding characters in the LCD

3.8 DISCUSSION ON-CHIP PORTS OF 8051:

3.8.1 COMPONENT:

As already explained in **"Chapter 1"**, the 8051 has One full Duplex Serial Port, Two 16-bit Timer/Counters and 32-bit Bidirectional and individually addressable I/O lines, onchip. The user can access these On-chip ports by using an optionally available 40-pin FRC IDC header with their own expansion cards. (This connector is available only in 8051 Piggy back board with Micropower-i)

3.8.2 COMPONENT DESCRIPTION:

8051 has four 8-bit Parallel ports. All four ports are bidirectional. Each consists of a latch, an output driver and an input buffer. All port pins are terminated at a 40-pin IDC header optionally available on the Piggy back board (in Micropower-i only).

3.8.3 PROGRAMMING THE PARALLEL I/O PORTS:

Though there are 32 I/O lines available in 8051, only 8 I/O lines (Port 1) are exclusively available for the user. This is because, the output drivers of Ports 0 and 2 and input buffers of Port 0 are used in accesses to external memory. Also, all the Port 3 pins are multi functional. They also serve various special functions, as already explained in Chapter-1 of this manual.

3.8.4 WRITING TO PARALLEL PORT:

To write any data to Port 1, Port 1 register (P1) should be loaded with that data to be output. As all the 8 I/O lines are bit addressable, they can be programmed either low or high independently.

3.8.5 EXAMPLE 8 - WRITING DATA TO PARALLEL PORT:

OBJECTIVE:
To write a sample program to output an 8-bit data, to Port 1 (P1).

THEORY:

8051 has four parallel ports. But, only Port 1 is user-free as Port 2 outputs the high byte of the external memory address and Port 0 outputs the low byte of the external memory address, time-multiplexed with the byte being written or read and Port 3 pins have alternate functions.

PROGRAM:

```
4100   74 45          MOV    A,#45
4102   F5 90          MOV    P1,A
4104   80 FE    HLT:  SJMP   HLT
```

VERIFICATION:

After entering the above program, execute it. Using a Multimeter or an Oscilloscope, check whether correct data is available at Port 1 (P1.0 - P1.7) of 8051. (i.e. Pin no. 1 - LSB of P1 & 8 - MSB of P1).

3.8.6 READING THE PARALLEL PORT:

The Parallel port of 8051 can also be used as an input port. To read a pin of any port of 8051, that pin should be made high.

3.8.7 EXAMPLE 9 - READING DATA FROM PARALLEL PORT:

OBJECTIVE:

To write a sample program to read data from the Parallel Port P1 of 8051.

THEORY: To read a pin of a port on the 8051 requires that a logic 1 first be written to that pin. Hence FF should be written to Port 1 to read all the 8-bits.

PROGRAM:

```
4100   75 90 FF          MOV     P1,#FF
4103   E5 90              MOV     A,P1
4105   90 45 00              MOV      DPTR,#4500
4108   F0                 MOVX    @DPTR,A
4109   80 FE       HLT:   SJMP    HLT
```

VERIFICATION:

To check this program, first of all, the Port 1 should be given an input which can be done by connecting the 26-pin IDC header of any of the 8255 with the on-board 40-pin IDC header through a 26 to 40 flat core cable in the case of Micropower-i only. Now, output any desired data through the port of 8255 that is connected to Port 1 of 8051, using the OUTput command available in the keypad. Enter the above program and execute it. Check whether the location 4500 has the data written to the 8255.

3.9 DESCRIPTION OF ON-CHIP TIMER/COUNTERS:

3.9.1 COMPONENT:

The 8051 has two 16-bit Timer/counter registers: Timer 0 and Timer 1. They can be configured to operate either as Timer or Counter.

3.9.2 COMPONENT DESCRIPTION:

In the Timer function, the register is incremented every machine cycle. Thus, one can think of it as counting machine cycles. Since a machine cycle consists of 12 oscillator periods, the count rate is 1/12 of the oscillator frequency.

In the Counter function, the register is incremented in response to a 1-to-0 transition at its corresponding external input pin, T0 and T1. In this function, the external input is sampled during S5P2 of every machine cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during S3P1 of the following the one in which the transition was detected. In addition to the "Timer" or "Counter" selection, Timer 0 and Timer 1 have four operating modes from which to select.

M1	M0	Operating Mode
0	0	8 - Bit Timer / Counter "Thx" with "Tlx" as 5 - bit prescalar
0	1	16-bit Timer / Counter "Thx" and Tlx" are Cascaded : there is no prescalar.
1	0	8-bit Auto reload timer / Counter Thx holds a value which is to be reloaded into Tlx each time it overflows.
1	1	(Timer 0) TL0is an 8-bit Timer / Counter controlled by the standard timer 0 control bits. TH0 is an 8-bit timer only controlled by timer 1 control bits.
1	1	(Timer 1) Timer / Counter 1 stopped.

NOTE:

Refer to **"CHAPTER 1"** for the complete details of the On-chip features of 8051.

3.9.3 PROGRAMMING DESCRIPTION:

For Programming description, refer to Chapter 4 on **"APPLICATION EXAMPLES".**

3.10 DESCRIPTION OF ON-CHIP SERIAL PORT:

3.10.1 COMPONENT:

The Serial Port is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered i.e. it can commence the reception of a second byte before a previously received byte has been read from the receive register. (However, if the first byte still hasn't been read by the time reception of the second byte is completed, one of the bytes will be lost).

3.10.2 COMPONENT DESCRIPTION:

The Serial Port receive and transmit registers can both be accessed at Special Function Register SBUF. Writing to SBUF loads the transmit register and reading SBUF accesses a physically separate receive register. The Serial Port can operate in 4 modes as already explained in **"CHAPTER 1"**.

3.10.3 GENERATING BAUD RATES USING TIMERS:

SERIAL PORT IN MODE 0:

Mode 0 has a fixed baud rate which is 1/12 of the Oscillator frequency (12MHz). To run the serial port in this mode, none of the Timer/Counters need to be set up. Only the SCON register needs to be defined.

$$\text{Baud Rate} = \frac{\text{Osc. Freq.}}{12}$$

MODE 1:

Mode 1 has a variable baud rate. The baud rate can be generated by Timer 1.

USING TIMER/COUNTER 1 TO GENERATE BAUD RATES:

For this purpose, Timer 1 is used in Mode 2 (Auto-Reload).

$$\text{Baud Rate} = \frac{\text{K x Oscillator frequency}}{32 \times 12 \times [256 - (\text{TH1})]}$$

If SMOD = 0, then K = 1
If SMOD = 1, then K =2 (SMOD is in PCON register)

Most of the time, the user knows the baud rate and needs to know the reload value for TH1. Therefore, the equation to calculate TH1 can be written as:

$$\text{TH1} = 256 - \frac{\text{K x Osc. Freq.}}{384 \times \text{baud rate}}$$

TH1 must be an integer value. Rounding off TH1 to the nearest integer may not produce the desired baud rate. In this case, the user may have to choose another crystal frequency.

Since the PCON register is not bit addressable, one way to set the bit is logical ORing the PCON register. (i.e. ORL PCON,#80H). The address of PCON is 87.

MODE 2:

The baud rate is fixed in this mode and is 1/32 or 1/64 of the oscillator frequency depending on the value of the SMOD bit in the PCON register.

In this mode, none of the Timers are used and the clock comes from the internal phase 2 clock.

SMOD = 1, Baud Rate = 1/32 of the Osc. Freq.

SMOD = 0, Baud Rate = 1/64 of the Osc. Freq.

CHAPTER - 4

APPLICATION EXAMPLES

OBJECTIVES:

i. To enable you to understand multiple peripheral interfacing.

ii. To write a program using more than a single I/O device, to familiarise with the concept of system design.

4.1 INTRODUCTION:

The following are some sample experiments that uses more than one peripheral to achieve a specific task. With these experiments, you can learn a lot about interfacing two different peripherals which will help to enhance your knowledge in the field of programming based on the hardware.

4.2 EXAMPLE 1 - DIFFERENTIATE RAM AND EPROM:

In **Micropower-i** with 8051 Piggy Back Board, the addresses from **E000 to FFFF** are **EPROM locations** (without auto-switching), while 4100 to 5FFF are User RAM locations. In **Micro-51, 0000 to 1FFF are EPROM locations** and **4100 to 5FFF are user RAM locations.** In Micro-51 EB, 0000 to 3FFF are EPROM locations and 4100 to 5FFF are user RAM locations. Now, let us see, what is the difference between EPROM and RAM?

An **EPROM** is an **Erasable Programmable Read Only Memory.** It cannot be written into, using SUB, MOVe, or FILL commands of the kit. To write into the EPROM, a separate EPROM programmer must be used.

A **RAM** is a **Random Access Memory** which can be either written into or read from. But the data written into can be retained only if the power is retained. If the power is turned-off, then the data in the RAM will be lost. But the EPROM can retain the data even when the power is turned-off.

Let us examine these two memory types.

i) Using FILL command, fill locations from 4200 to 420F with the data byte 45.

ii) Examine the following memory locations using SUB command and write down the contents next to each address. Use the first column of "DATA" here.

ADDRESS	DATA	DATA	ADDRESS	DATA	DATA
E000	__	__	4200	__	__
E001	__	__	4201	__	__
E002	__	__	4202	__	__
E003	__	__	4203	__	__
E004	__	__	4204	__	__
E005	__	__	4205	__	__
E006	__	__	4206	__	__
E007	__	__	4207	__	__
E008	__	__	4208	__	__
E009	__	__	4209	__	__
E00A	__	__	420A	__	__
E00B	__	__	420B	__	__
E00C	__	__	420C	__	__
E00D	__	__	420D	__	__
E00E	__	__	420E	__	__
E00F	__	__	420F	__	__

iii) If you are working in Micro-51 or in Micro-51 EB, then use locations 0000 to 000F, instead of E000 to E00F.

iv) Now,turn the power to the trainer off. Wait for around 10 to 15 seconds and then turn-on the trainer now.

v) Examine the memory locations as stated in Step (ii) and write the data read in the second column provided. By comparing these two sets of data, it will be clear that the data at 4200 to 420F has changed but the data at E000 to E00F (or at 0000 to 000F) remains the same.

vi) Therefore, the addresses from 4200 is RAM while E000 (0000 in the case of Micro-51 and Micro-51 EB) is EPROM. The contents of RAM are lost as the power is turned-off. When power is given again, a random data appears.

vii) Enter a new data at E000 (0000 for Micro-51 and Micro-51 EB). What happens? "Err." is displayed, which indicates that E000 (0000) cannot be written into.

EXERCISES:

i) Write a program that distinguishes between a RAM address or an EPROM address of Micropower-i. The address is specified by the user. Store data at a location, as the result to distinguish whether the specified address is RAM or EPROM. (00 to indicate RAM and FF to indicate EPROM)

4.3 EXAMPLE 4 - INTERFACING 8255 AND 8279:

OBJECTIVE:

To configure 8255-1 having all ports as output and 8255-2 to have ports as input depending upon the user input. Then, read the key and output it to 8255-1. Then, input it using 8255-2 at the desired port and display the data.

THEORY:

i) First of all, connect a 26-pin flat core cable between the two 26-pin IDC headers of 8255-1 and 8255-2.

ii) Read a key from the keyboard.

iii) If key = 0, then 8255-2 Port A alone is input.

iv) If key = 1, then 8255-2 Port B alone is input.

v) If key = 2, then 8255-2 Port C alone is input.

vi) 8255-1 has all its ports as output.

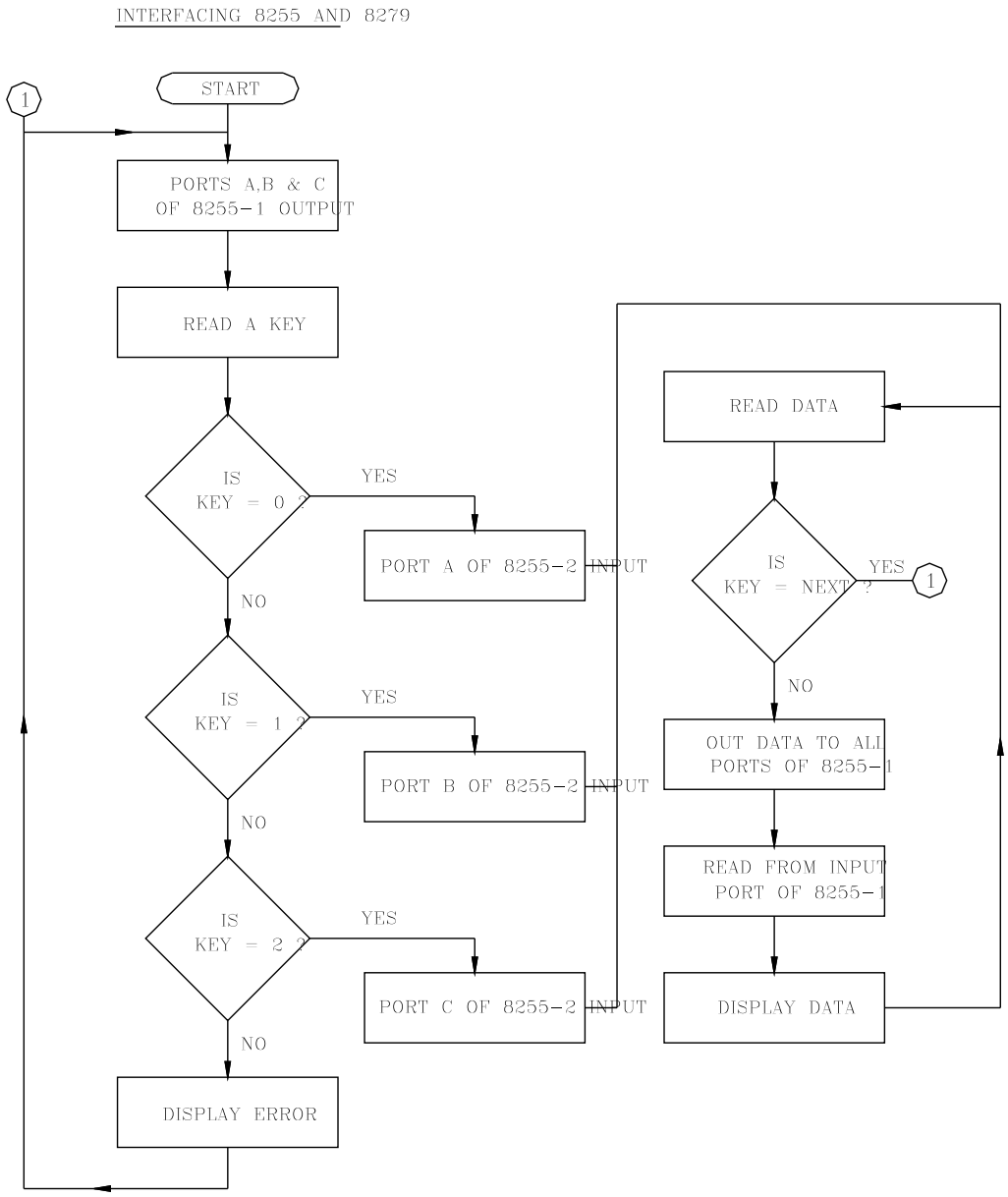
vii) Now, again read a key from 0 to F.

viii) Output it to 8255-1.

ix) Read this data from 8255-2 through the port which was selected as input port.

x) Display the data read from 8255-2 with the message of the Port from which data was read.

FLOW CHART:


INTERFACING 8255 AND 8279
START
1
PORTS A,B & C OF 8255-1 OUTPUT
READ A KEY
IS KEY = 0 ?
YES
PORT A OF 8255-2 INPUT
NO
IS KEY = 1 ?
YES
PORT B OF 8255-2 INPUT
NO
IS KEY = 2 ?
YES
PORT C OF 8255-2 INPUT
NO
DISPLAY ERROR
READ DATA
IS KEY = NEXT ?
YES
1
NO
OUT DATA TO ALL PORTS OF 8255-1
READ FROM INPUT PORT OF 8255-1
DISPLAY DATA

PROGRAM:

```
4100  90 XX XX        MOV     DPTR,#CNT_1
4103  74 80           MOV     A,#80H
4105  F0              MOVX    @DPTR,A
                  START1:
4106  12 41 5F        LCALL   DISPORT
4109  12 41 DD        LCALL   DELAY
                                        ;Get key to decide which port
                                        ;of 8255-2 is input port.
410C  12 41 ED        LCALL   RDKBD
410F  90 XX XX        MOV     DPTR,#CNT_2
4112  B4 00 13        CJNE    A,#0,NXT
                                        ;Port A input
                  P2AIN:
4115  74 90           MOV     A,#90H
4117  F0              MOVX    @DPTR,A
4118  12 41 5F        LCALL   DISPORT
411B  12 41 E6        LCALL   STA_INT
411E  90 XX XX        MOV     DPTR,#DSP_DAT
4121  74 88           MOV     A,#88H
4123  F0              MOVX    @DPTR,A
4124  78 00           MOV     R0,#0
4126  80 63           SJMP    SAME
                  NXT:
4128  B4 01 13        CJNE    A,#1,NXT1
                                        ;
                                        ;Port B input
                                        ;
                  P2BIN:
412B  74 82           MOV     A,#82H
412D  F0              MOVX    @DPTR,A
412E  12 41 5F        LCALL   DISPORT
4131  12 41 E6        LCALL   STA_INT
4134  74 38           MOV     A,#38H
4136  90 XX XX        MOV     DPTR,#DSP_DAT
4139  F0              MOVX    @DPTR,A
413A  78 01           MOV     R0,#1
413C  80 4D           SJMP    SAME
                  NXT1:
413E  B4 02 13        CJNE    A,#2,ERR
                                        ;Port C input
                  P2CIN:
4141  74 89           MOV     A,#89H
4143  F0              MOVX    @DPTR,A
```

```
4144  12 41 5F        LCALL   DISPORT
4147  12 41 E6        LCALL   STA_INT
414A  74 6C           MOV     A,#6CH
414C  90 XX XX        MOV     DPTR,#DSP_DAT
414F  F0              MOVX    @DPTR,A
4150  78 02           MOV     R0,#2
4152  80 37           SJMP    SAME
                                        ;This routine gives Error
                                        ;message for invalid input keys.
              ERR:
4154  90 42 04              MOV     DPTR,#ERRMSG
4157  12 41 62              LCALL   DISPORT1
415A  12 41 83              LCALL   DLY3
415D  80 A7                 SJMP    START1
                                        ;Display Routine
              DISPORT:
415F  90 41 FC              MOV     DPTR,#PORTMSG
              DISPORT1:
4162  C0 82                 PUSH    DPL
4164  C0 83                 PUSH    DPH
4166  90 XX XX              MOV     DPTR,#DSP_CNT
4169  74 96                 MOV     A,#96H
416B  F0                    MOVX      @DPTR,A
416C  D0 83                 POP     DPH
416E  D0 82                 POP     DPL
4170  79 08                 MOV     R1,#8
              DISPAG:
4172  E0                    MOVX    A,@DPTR
4173  C0 82                 PUSH    DPL
4175  C0 83                 PUSH    DPH
4177  90 XX XX              MOV     DPTR,#DSP_DAT
417A  F0                    MOVX    @DPTR,A
417B  D0 83                 POP     DPH
417D  D0 82                 POP     DPL
417F  A3                    INC     DPTR
4180  D9 F0                 DJNZ    R1,DISPAG
4182  22                    RET
                                        ;Delay routine
              DLY3:
4183  7C 07                 MOV     R4,#7
4185  12 41 DD  DLY2:       LCALL   DELAY
4188  DC FB                 DJNZ    R4,DLY2
418A  22                    RET
              SAME:
418B  12 41 83              LCALL   DLY3
```

```
418E  90 42 0C          MOV     DPTR,#BYTEMSG
4191  12 41 62          LCALL   DISPORT1
                                        ;Get Data from the Keyboard
4194  12 41 ED          LCALL   RDKBD
4197  12 41 D2          LCALL   _NXT
419A  C4                SWAP    A
419B  FA                MOV     R2,A
419C  12 41 ED          LCALL   RDKBD
419F  12 41 D2          LCALL   _NXT
41A2  2A                ADD     A,R2
                                        ;Output the data to all ports of 8255-1.
41A3  90 XX XX          MOV     DPTR,#APORT_1
41A6  F0                MOVX    @DPTR,A
41A7  A3                INC     DPTR
41A8  F0                MOVX    @DPTR,A
41A9  A3                INC     DPTR
41AA  F0                MOVX    @DPTR,A
                                        ;Check which port has beenselected as
                                        input port and input data from that
                                        port.
41AB  E4                CLR     A
41AC  B8 00 16          CJNE    R0,#0,NXT3
                                        ;
41AF  90 XX XX          MOV     DPTR,#APORT_2
                SAME1:
41B2  E0                MOVX    A,@DPTR
                                        ;Display the data read from the input port.
41B3  F5 82             MOV     DPL,A
41B5  79 02             MOV     R1,#2
41B7  74 02             MOV     A,#2
41B9  12 00 20          LCALL   20H
                                        ;Read keyboard & check for
                                        ;'NEXT' key
41BC  12 41 ED          LCALL   RDKBD
41BF  B4 14 92          CJNE    A,#14H,ERR
41C2  02 41 06          LJMP    START1
                                        ;Keycode 10 for next key in
                                        ;Micro-51 and Micro-51 EB
                NXT3:
41C5  B8 01 05          CJNE    R0,#1,NXT4
41C8  90 XX XX          MOV     DPTR,#BPORT_2
41CB  80 E5             SJMP    SAME1
                NXT4:
41CD  90 XX XX          MOV     DPTR,#CPORT_2
41D0  80 E0             SJMP    SAME1
```

```
                    _NXT:
41D2  C3            CLR     C
41D3  94 10         SUBB    A,#10H
41D5  50 03         JNC     ERR1
41D7  24 10         ADD     A,#10H
41D9  22            RET
                    ERR1:
41DA  02 41 54      LJMP    ERR    ;Delay Routine

                    DELAY:
41DD  7F FF         MOV     R7,#0FFH
                    DLY1:
41DF  7E FF         MOV     R6,#0FFH
41E1  DE FE         DJNZ    R6,DLY
41E3  DF FA         DJNZ    R7,DLY1
41E5  22            RET
                    STA_INT:
41E6  90 XX XX      MOV     DPTR,#DSP_CNT
41E9  74 94         MOV     A,#94H
41EB  F0            MOVX    @DPTR,A
41EC  22            RET
                                    ;
                                    ;Routine to Read the
                                    ;Key pressed.
                                    ;
                    RDKBD:
41ED  90 XX XX      MOV     DPTR,#DSP_CNT
                    CHK:
41F0  E0            MOVX    A,@DPTR
41F1  54 07         ANL     A,#7
41F3  60 FB         JZ      CHK
41F5  74 40         MOV     A,#40H
41F7  F0            MOVX    @DPTR,A
                                    ;
41F8  15 82         DEC     DPL
41FA  E0            MOVX    A,@DPTR
41FB  22            RET
                                    ;
                    PORTMSG:
41FC  C8 0C F9 78           0C8H,0CH,0F9H,78H
4200  9F B9 FF FF           9FH,0B9H,0FFH,0FFH
                                    ;
                    ERRMSG:
4204  FF FF 68 F9           0FFH,0FFH,68H,0F9H
4208  F9 F7 FF FF           0F9H,0F7H,0FFH,0FFH
                        ;
                BYTEMSG:
420C  9F B9 38 1A           9FH,0B9H,38H,1AH
4210  78 68 FF FF           78H,68H,0FFH,0FFH
```

VERIFICATION:

Enter the program and execute it. Select the Input Port for 8255-2 by selecting either 0 or 1 or 2 for Ports A or B or C. Now, enter data and see that the data is displayed. To go back to the start, press "NEXT" key. Error message 'E' will be displayed when an invalid key is pressed.

4.4 ACCESSING ON-CHIP TIMER-COUNTERS:

4.4.1 EXAMPLE 6 - DIGITAL CLOCK USING TIMER 1 IN MODE 1:

OBJECTIVE:

To write a sample program to implement a Digital clock using the On-chip Timer/Counter 1 in Mode 1.

THEORY:

To implement the digital clock, for every 1 sec delay, interrupt should be generated. On chip timer 1 has been utilised to generate 1 sec Delay. TH1 and TL1 are loaded with 37H and FFH respectively and a loop of 12H times (No. of interrupts to be generated) is continued so as to get 1 sec Delay i.e. we arrive at counts 449982D before overflow is reached and an interrupt is generated. This procedure is utilised in our program to generate the desired delay. First of all, to use Timer 1 in Mode 1, following initialisations should be done.

Enable Timer 1 Interrupt.
Select Timer / counter 1 as Timer.
Select the Mode of operation.
Set Timer 1 on Load no. of interrupts to be counted for 1 sec Delay.

PROGRAM:

```
4100  75 A8 88          MOV     IE,#88H
4103  75 89 10          MOV     TMOD,#10H
4106  75 8D 37          MOV     TH1,#37H
4109  75 8B FF          MOV     TL1,#FFH
410C  75 50 12          MOV     50H,#12H
410F  75 88 40          MOV     TCON,#40H
                                        ;SECS, MINS, HRS LOADED
                                        ;FROM 4700H, ARE MOVED TO
                                        ;INTERNAL MEM. LOCATION 51H,
                                        ;52H AND 53H RESPECTIVELY
4112  90 47 00          MOV     DPTR,#DATA
4115  7A 03             MOV     R2,#03H
4117  78 51             MOV     R0,#51H
4119  E0       Loop1:   MOVX    A,@DPTR
```

```
411A  F6              MOV     @R0,A
411B  A3              INC     DPTR
411C  08              INC     R0
411D  DA FA           DJNZ    R2,LOOP1
411F  12 41 3C  Loop2: LCALL  LOOP5     ;waits for 1 sec Delay.
4122  10 00 FA  Loop3: JBC    00,LOOP2
4125  80 FB           SJMP    LOOP3
                                        ;INTERRUPT SERVICE ROUTINE.
```

This routine decrements the Int. RAM 50H, and checks for 00. If not zero, again loads the TH1 & TL1. Then starts the timer and returns. If Zero, Sets the bit in address 0 and then loads TH1 & TL1 and starts the timer and returns.

```
4127  15 50           DEC     050H
4129  E5 50           MOV     A,050H
412B  70 05           JNZ     LOOP4
412D  D2 00           SETB    00
412F  75 50 12        MOV     50,#12H
4132  75 8D 37  Loop4: MOV     TH1,#37H
4135  75 8B FF        MOV     TL1,#FFH
4138  75 88 40        MOV     TCON,#40H
413B  32              RETI
```

This part of the program increments SECS, checks for 60. If not, goes to display. If yes, resets SECS and increments MINS, and checks for 60. If not, goes to display. If yes, Resets MINS and increments HRS, then checks for 13. If not, goes to display. If yes, sets HRS to 01 and goes to display.

```
413C  7A 02     Loop5: MOV     R2,#02H
413E  78 51            MOV     R0,#51H
4140  E6        Loop6: MOV     A,@R0
4141  24 01            ADD     A,#01
4143  D4                DA      A
4144  F6                MOV     @R0,A
4145  B4 60 0E          CJNE    A,#60H Display
4148  76 00            MOV     @R0,#00
414A  08                INC     R0
414B  DA F3             DJNZ    R2,LOOP6
414D  06                INC     @R0
414E  C3                CLR     C
414F  E6                MOV     A,@R0
4150  D4                DA      A
4151  B4 13 02          CJNE    A,#13H, Display
4154  76 01            MOV     @R0,#01
4156  C0 E0    Display: PUSH    A
```

```
4158  C0 08        PUSH   R0
415A  C0 0A        PUSH   R2
415C  74 02        MOV    A,#02
415E  79 01        MOV    R1,#01
4160  75 F0 00     MOV    B,#00
4163  78 53        MOV    R0,#53H
4165  86 83        MOV    DPH,@R0
4167  18          DEC    R0
4168  86 82        MOV    DPL,@R0
416A  12 00 20     LCALL  0020H
416D  78 51        MOV    R0,#51H
416F  74 02        MOV    A,#02
4171  79 02        MOV    R1,#02
4173  75 F0 04     MOV    B,#04
4176  86 82        MOV    DPL,@R0
4178  12 00 20     LCALL  0020H
417B  D0 0A        POP    R2
417D  D0 08        POP    R0
417F  D0 E0        POP    A
4181  22          RET
```

VERIFICATION:

Enter the above program from 4100. Before executing this Program, first enter the I.S.R (Interrupt Service Routine) jump address from 4018 (403B for Micro-51 EB) onwards as 02, 41 and 27 at locations 4018, 4019 and 401A (For Micro-51 EB 403B, 403C and 403D) locations respectively. Then, enter the starting time from 4700 in the 12 hours format in the order Seconds, Minutes and Hours respectively. Now, execute the program. You can see the Digital clock displayed on the 7-segment display in 12 hour format.

4.5 ACCESSING ON-CHIP SERIAL PORT:

4.5.1 EXAMPLE 7 - TRANSMIT FROM 8051 & RECEIVE USING 8251:

OBJECTIVE:

To transmit a byte using the On-chip Serial Port in Mode 2 and receiving the same using the on-board 8251.

THEORY:

As the Baud clock to the serial port should be 1/32 (375 KHz) or 1/64 (187.5 KHz) of the Oscillator frequency (12MHz) in Mode 2, baud clock to 8251 should also be the same. Channel 0 of 8253 is used to generate the baud clock to 8251. Here, serial transmission is done at the baud clock rate of 1/32 of the Oscillator frequency. Hence, Channel 0 of 8253 is initialised in

Mode 3 to generate 375 KHz clock.

(Therefore, SMOD i.e. Bit 7 of PCON (POWER CONTROL REISTER)) should be set to 1. This can be done by ORing PCON with 80).

PROGRAM:

```
4100  90 XX XX      MOV     DPTR,#TMR_CNT
4103  74 16         MOV     A,#16H
4105  F0            MOVX    @DPTR,A
                                        ;
4106  90 XX XX      MOV     DPTR,#TMR_CH0
4109  74 04         MOV     A,#4
410B  F0            MOVX    @DPTR,A
                                        ;
410C  90 XX XX      MOV     DPTR,#SER_CNT
410F  E4            CLR     A
4110  F0            MOVX    @DPTR,A
4111  F0            MOVX    @DPTR,A
4112  F0            MOVX    @DPTR,A
4113  74 40         MOV     A,#40H
4115  F0            MOVX    @DPTR,A
4116  74 4D         MOV     A,#4DH
4118  F0            MOVX    @DPTR,A
4119  74 37         MOV     A,#37H
411B  F0            MOVX    @DPTR,A
                                        ;
411C  75 98 80      MOV     SCON,#80H
411F  43 87 80      ORL     PCON,#80H
                                        ;
4122  75 99 54      MOV     SBUF,#54H
                                        ;
              CHK:
4125  30 99 FD      JNB     SCON.1,CHK
4128  90 XX XX      MOV     DPTR,#SER_CNT
              CHK1:
412B  E0            MOVX    A,@DPTR
412C  54 02         ANL     A,#2
412E  60 FB         JZ      CHK1
4130  15 82         DEC     DPL
4132  E0            MOVX    A,@DPTR
4133  90 42 50      MOV     DPTR,#STOR
4136  F0            MOVX    @DPTR,A
4137  80 FE  HLT:   SJMP    HLT
```

VERIFICATION:

First of all, connect the TXD (Transmit Data) i.e Pin no. 11 of 8051 to RXD (Receive Data) i.e Pin no. 3 of 8251. Short RTS* and CTS* signals of 8251. Enter the above Program from 4100. Now, execute the above program. Check at 4250 whether it has the data 54.

4.5.2 EXAMPLE 8 - TRANSMIT USING 8251 & RECEIVE USING 8051:

(For Micro-51 and Micropower-i only)

OBJECTIVE:

To write a sample program to receive a byte from 8251 and store it at location 4250.

THEORY:

Here, the data is transmitted serially from 8251 at the clock rate of 157 KHz. So, using Channel 0 of 8253, the baud clock to 8251 is generated. The serially transmitted data is received using the onchip serial port of 8051 through RXD.

PROGRAM:

```
4100  90 XX XX        MOV     DPTR,#TMR_CNT
4103  74 16           MOV     A,#16H
4105  F0              MOVX    @DPTR,A
                                                    ;
4106  90 XX XX        MOV     DPTR,#TMR_CH0
4109  74 08           MOV     A,#8
410B  F0              MOVX    @DPTR,A
                                                    ;
410C  90 XX XX        MOV     DPTR,#SER_CNT
410F  E4              CLR     A
4110  F0              MOVX    @DPTR,A
4111  F0              MOVX    @DPTR,A
4112  F0              MOVX    @DPTR,A
                                                    ;
4113  74 40           MOV     A,#40H
4115  F0              MOVX    @DPTR,A
4116  74 4D           MOV     A,#4DH
4118  F0              MOVX    @DPTR,A
4119  74 37           MOV     A,#37H
411B  F0              MOVX    @DPTR,A
                ;
411C  90 XX XX        MOV     DPTR,#SER_CNT
                CHK2:
```

```
411F  E0              MOVX   A,@DPTR
4120  54 04           ANL    A,#4
4122  60 FB           JZ     CHK2
4124  15 82           DEC    DPL
4126  74 46           MOV    A,#46H
4128  F0              MOVX   @DPTR,A
                                        ;
4129  75 98 90        MOV    SCON,#90H
412C  43 87 7F        ORL    PCON,#7FH
                CHK3:
412F  30 98 FD        JNB    SCON.0,CHK3
4132  C2 98           CLR    SCON.0
4134  E5 99           MOV    A,SBUF
4136  90 42 50        MOV    DPTR,#STOR
4139  F0              MOVX   @DPTR,A
413A  80 FE       HLT: SJMP   HLT
```

VERIFICATION:

First of all, connect the RXD i.e Pin no. 10 of 8051 to TXD i.e Pin no. 19 of 8251. Short RTS* and CTS* signals of 8251. Enter the above Program from 4100. Now, execute the above program. Check at 4250 whether it has the data 46.

(For Micro-51 EB)

OBJECTIVE:

To transmit and receive a byte using on-chip Serial Port in mode 1 and Store the received byte at location 4500.

PROGRAM:

```
4100  75 89 20        MOV    TMOD,#20H
4103  75 8D FD        MOV    TH1,#FDH
4106  75 8B 00        MOV    TL1,#00
4109  75 88 40        MOV    TCON,#40H
410C  75 98 58        MOV    SCON,#58H
410F  75 99 54        MOV    SBUF,#54H
                REPEAT:
4112  30 98 FD        JNB    SCON.0 REPEAT
4115  C2 98           CLR    SCON.0
4117  E5 99           MOV    A,SBUF
4119  90 45 00        MOV    DPTR,#4500
411C  F0              MOVX   @DPTR,A
                HLT:
411D  80 FE           SJMP   HLT
```

VERIFICATION:

Connect a loop back connector to the Serial Port Connector. Enter the above program from 4100 and execute it. Check at location 4500 using substitute command. If the contents of this location is 54, then the Serial Port is OK. Else check the loop back connector and verify RTS*, CTS*, TXD and RXD using an oscilloscope.

4.6. EXERCISES:

1. Write a program to input data from a peripheral device as a parallel stream and print it using the Printer interface provided in the trainer kit.

2. Write a program to print the data received serially, using the Printer interface available in the trainer.

3. Transfer a program using the serial interface to the trainer and store the same in an EPROM using the EPROM programmer provided in Micro-51.
4. Write a Program to Display a 24-hour format Digital Clock using Timer / Counter 1 in Mode 2 i.e. Auto Reload facility.

5. Give an external input to T0 (pin no. 14) of 8051 by connecting a de-bounced switch to it. Throwing switch low each time will represent an event. The number of these events can be stored in TL0 of TC0. Read TL0 and display its contents.

6. Write a program to transmit a byte using 8051 in Mode 1 at the baud rate of 2400 and receive it using the on-board 8251.

7. Write a program to transmit a block of data using onboard 8251 and receive it using 8051 at the baud rate of 1200 in Mode 3 and store it from location 4200 onwards.

CHAPTER - 5

SYSTEM ORIENTED FUNCTIONS

OBJECTIVES:

i. To equip you with usage of built in systems calls.

5.1 INTRODUCTION

You have come a long way with 8031 / 8051.You have worked out all the software and hardware examples as well as the exercises. Hence now you definitely or a very good programmer, you need to learn the facilities which are provided by the system namely system calls. Use of system calls shall shorten your code for any program and transform it into small and efficient as was said in chapter - 2.

5.2 SYSTEM CALLS

Having worked out all the software, hardware examples and exercises in the previous chapters, you have now come a long way with the 8051. To further enhance your proficiency as a system programmer, it is imperative that you learn the facilities provided by the system as user callable routines. Use of these system calls shall shorten your code for any problem and make your program small and efficient.

The list of calls available with the Micropower-i (8051) is given in the manual CAT #MP1-51-001 and is available with Micro-51 and Micro-51 EB kit too and are given in the Technical reference manual for Micro-51, CAT #M51-001, and Micro-51 EB Technical Reference Manual. Let us employ these routines like resetting the trainer and so on. The entry point for these subroutines is 8020/E020 for Micropower-i depending on the EPROM address and 0020 for Micro-51 and Micro-51 EB.

5.2.1 EXAMPLE-1:

OBJECTIVE:

To reset the kit using a system call.

THEORY:

Initialise A to 00 and then branch to the subroutine at E020 (use 0020 for Micro-51 and Micro-51 EB).

PROGRAM:

```
4100  E4          CLR     A
4101  12          LCALL   E020
4102  E0
4103  20
4104  80    HLT:  SJMP    HLT
4105  FE
```

VERIFICATION:

Enter the above program and execute it. The trainer gets reset.

5.2.2 EXAMPLE-2:

OBJECTIVE:

To read a 2 byte address from the keyboard.

THEORY:

To read a key from the keyboard, initialise A to 03 and if we are to read 4 keys, R1 is to be set to 00. The keys are input and the input keys are displayed in the address field and the key code is available in the DPTR register.

PROGRAM:

```
4100  74 03          MOV     A,#3
4102  79 00          MOV     R1,#0
4104  12 E0 20       LCALL   E020
4107  80 FE    STOP: SJMP    STOP
```

VERIFICATION:

The above program when executed waits for you to enter the keys. Only hex keys are input and displayed. The program has to be suitably modified to display error when any other key has been input, other than a hex key.

EXERCISES:

i) Flash the message "dISPLAY" using Function Calls 02 and 0B.

ii) Display the following, using Function Call Routines.

a).

	.		.		.		.

b)

.	µ	-	-	8	0	A	d.

c)

					.	5	F

d)

	.	H	E	L	P		

iii) Display the message "Err." using the Function call 04.

iv) Generate a tone of 1KHz to the speaker using the Function call routines.

v) Using Function Call 0A, write a routine to read from Channel 0 of ADC (in Micropower-i) and display the digital data using Function Call 02.

vi) Serially input data from another system and print them on a Parallel printer using Function calls 07 and 09.

CHAPTER - 6

INTERRUPTS

OBJECTIVES

i. To familiarise the On-chip interrupts of 8031/8051.

6.1 INTRODUCTION:

8051 has 5 interrupt sources: 2 external interrupts, 2 timer interrupts and one serial port interrupt The **External interrupts INT0* and INT1*** can each be either level-activated or transition-activated, depending on bits IT0 and IT1 in Register TCON (Please refer to the Chapter 1 for the complete details of various registers mentioned here). The flags that actually generate these interrupts are bits IE0 and IE1 in TCON. When an external interrupt is generated, the flag that generated it is cleared by the hardware, when the service routine is vectored to, only if the interrupt was transition-activated. If the interrupt was level-activated, then, the external requesting source is what controls the request flag, rather than the on-chip hardware.

The **Timer 0 and Timer 1** Interrupts are generated by TF0 and TF1, which are set by a rollover in their respective timer/counter registers. When a timer interrupt is generated, the flag that generated it is cleared by the on-chip hardware when the service routine is vectored to.

The **Serial Port** Interrupt is generated by the logical OR of RI and TI. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine will normally have to determine whether it was RI or TI that generated the interrupt and the bit will have to be cleared in software.

All the bits that generate interrupts can be set or cleared by software, with the same result as though it had been set or cleared by hardware. That is, interrupts can be generated or pending interrupts can be cancelled by the software.

Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in the Special Function Register IE. Note that IE also contains a global disable bit, EA, which disables all interrupts at once.

IE: INTERRUPT ENABLE REGISTER - BIT ADDRESSABLE:

If the bit is 0, the corresponding interrupt is disabled. If the bit is 1, the corresponding interrupt is enabled.

D7	D6	D5	D4	D3	D2	D1	D0
EA	-	ET2	ES	ET1	EX1	ET0	EX0

EA IE.7 - Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

- IE.6 - Not implemented, reserved for future use.

ET2 IE.5 - Enable or disable the Timer 2 overflow interrupt.

ES IE.4 - Enable or Disable the Serial Port interrupt.

ET1 IE.3 - Enable or Disable the Timer 1 overflow interrupt.

EX1 IE.2 - Enable or Disable the External Interrupt 1.

ET0 IE.1 - Enable or Disable the Timer 0 overflow interrupt.

EX0 IE.0 - Enable or Disable the External Interrupt 0.

6.2 PRIORITY LEVEL STRUCTURE:

Each interrupt source can also be individually programmed to one of the two priority levels by setting or clearing a bit in the Special Function Register IP. A low priority interrupt can itself be interrupted by a high priority interrupt, but not by another low-priority interrupt. A high priority interrupt can't be interrupted by another interrupt source.

If two requests of different priority levels are received simultaneously, the request of higher priority level will be serviced first. If requests of the **same** priority level are received simultaneously, an internal polling sequence determines which request has to be serviced. Thus, within each priority level, there is a second priority structure determined by the polling sequence, as follows:

No.	SOURCE	PRIORITY WITHIN LEVEL
1	IE0	(HIGHEST)
2	TF0	
3	IE1	
4	TF1	
5	RI + TI	(LOWEST)

Note that the "priority within level" structure is only used to resolve **simultaneous requests of the same priority level.**

IP: INTERRUPT PRIORITY REGISTER. BIT ADDRESSABLE:

If the bit is 0, the corresponding interrupt has a lower priority and if the bit is 1, the corresponding interrupt has a higher priority.

D7	D7	D7	D7	D7	D7	D7	D7
-	-	PT2	PS	PT1	PX1	PT0	PX0

IP.7, IP.6 - Not implemented, reserved for future use.

PT2 IP.5 - Defines the Timer 2 interrupt priority.

PS IP.4 - Defines the Serial Port interrupt priority.

PT1 IP.3 - Defines the Timer 1 interrupt priority.

PX1 IP.2 - Defines External Interrupt 1 priority.

PT0 IP.1 - Defines the Timer 0 interrupt priority.

PX0 IP.0 - Defines External Interrupt 0 priority.

6.3 HOW INTERRUPTS ARE HANDLED:

The interrupt flags are sampled at S5P2 of every machine cycle (In 8051, a machine cycle consists of a sequence of 6 states, numbered S1 through S6. Each state time lasts for two oscillator periods. Each state is divided into a Phase 1 half and a Phase 2 half). The samples are polled during the following machine cycle. If one of the flags was in a set condition at S5P2 of the preceding cycle, the polling cycle will find it and the interrupt system will generate a **LCALL** to the appropriate service routine, provided this hardware-generated **LCALL** is not blocked by any of the following conditions:

1. An interrupt of equal or higher priority level is already in progress.

2. The current (polling) cycle is not the final cycle in the execution of the instruction in progress.

3. The instruction in progress is RETI or any access to the IE or IP registers.

Any of these three conditions will block the generation of the **LCALL** to the Interrupt Service Routine. Condition 2 ensures that the instruction in progress will be completed before vectoring to any service routine. Condition 3 ensures that if the instruction in progress is RETI or any access to IE or IP, then at least one more instruction will be executed before any interrupt is vectored to.

The polling cycle is repeated with each machine cycle and the values polled are the values that were present at S5P2 of the previous machine cycle. Note that if an interrupt flag is active but not being responded too for one of the above conditions, if the flag is not still active when the blocking condition is removed, the denied interrupt will not be serviced. In other words, the fact that the interrupt flag was once active but not serviced is not remembered. Every polling cycle is new.

Note that if an interrupt of higher priority level goes active prior to S5P2 of the machine cycle labelled C3 in the Figure F6.1, then, in accordance with the above rules, it will be vectored to during C5 and C6, without any instruction of the lower priority routine having been executed.

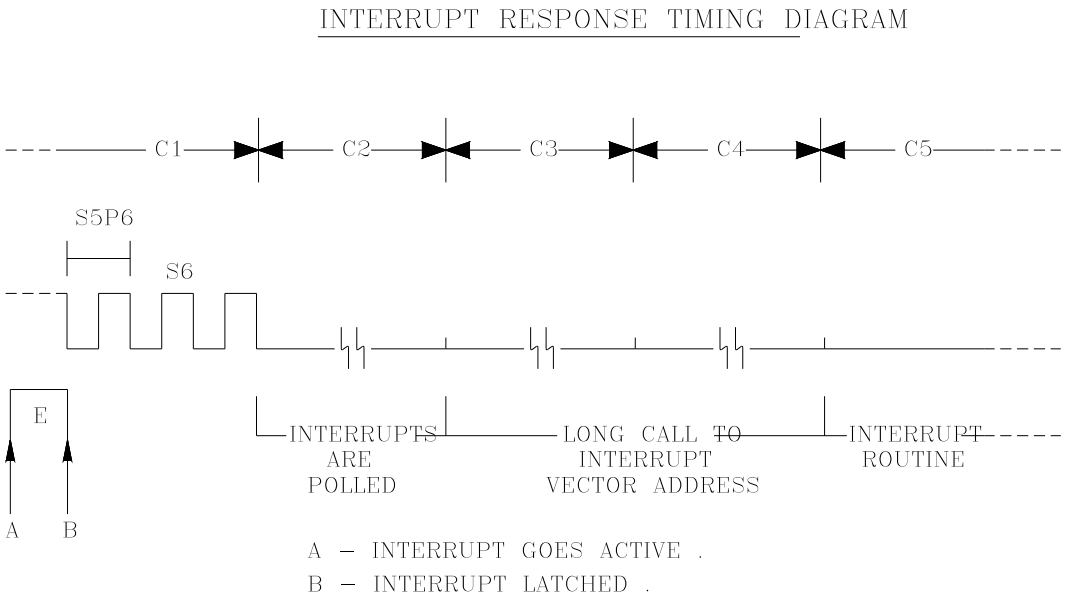


Figure F6.1

Thus, the processor acknowledges an interrupt request by executing a hardware-generated LCALL to the appropriate servicing routine. In some cases, it also clears the flag that generated the interrupt and in other cases it does not. It never clears the Serial Port or Timer 2 flags. This has to be done in the user's software. It clears an external interrupt flag (IE0 or IE1) only if it was transition-activated. The hardware-generated LCALL pushes the contents of the Program Counter onto the stack (but it does not save the PSW) and reloads the PC with an address that depends on the source of the interrupt being vectored to. The vector address of all the interrupt sources available in 8051 are as listed below.

SOURCE	VECTOR ADDRESS	VECTOR ADDRESS in the user RAM (For Micro-51 and Micropower-i)	VECTOR ADDRESS in the user RAM (Micro-51EB)
IE0	fs0003	Not available to user	Not available to user
TF0	000B	4015	Not available to user
IE1	0013	4018	4038
TF1	001B	401B	403B
RI + TI	0023	401E	403E

Execution proceeds from that location until the RETI instruction informs the processor that this interrupt routine is no longer in progress, then pops the top two bytes from the stack and reloads the Program Counter. Execution of the interrupted program continues from where it left off.

Note that a simple RET instruction would also have returned execution to the interrupted program, but it would have left the interrupt control system thinking that an interrupt was still in progress.

The interrupt function 'INT' available on the keypad of the trainer is provided using the external interrupt 0 of 8051. Hence, this interrupt is not available for the user.

6.4 EXTERNAL INTERRUPTS:

The external sources can be programmed to be level-activated or transition-activated by setting or clearing bit IT1 or IT0 in Register TCON. If ITx = 0, external interrupt x is triggered by a detected low at the INTx* pin. If ITx = 1, external interrupt x is edge-triggered. In this mode, if successive samples of the INTx* pin show a high in one cycle and a low in the new cycle, interrupt request flag IEx in TCON is set. Flag bit IEx then requests the interrupt.

Since the external interrupt pins are sampled once each machine cycle, an input high or low should hold for atleast 12 oscillator periods to ensure sampling. If the external interrupt is transition-activated, the external source has to hold the request pin high for at least one cycle and then hold it low for atleast one cycle to ensure that the transition is seen so that interrupt request flag IEx will be set. IEx will be automatically cleared by the CPU when the service routine is called.

If the external interrupt is level-activated, the external source has to hold the request active until the requested interrupt is actually generated. Then, it has to deactivate the request before the interrupt service routine is completed or else another interrupt will be generated.

6.5 RESPONSE TIME:

The INT0* and INT1 levels are inverted and latched into IE0 and IE1 at S5P2 of every machine cycle. The values are not actually polled by the circuitry until the next machine cycle. If a request is active and conditions are right for it to be acknowledged, a hardware subroutine call to the requested service routine will be the next instruction to be executed. The call itself takes two cycles. Thus, a minimum of three complete machine cycles elapse between the activation of an external interrupt request and the beginning of execution of the first instruction of the service routine.

A longer response time would result if the request is blocked by one of the 3 previously listed conditions. If an interrupt of equal or higher priority level is already in progress, the additional wait time obviously depends on the nature of the other interrupt's service routine. If the instruction in progress is not in its final cycle, the additional wait time cannot be more than 3 cycles, since the longest instructions (MUL and DIV) are only 4 cycles long and if the instruction in progress is RETI or an access to IE or IP, the additional wait time cannot be more than 5 cycles (a maximum of one more cycle to complete the instruction in progress, plus 4 cycles to complete the next instruction if the instruction is MUL and DIV).

Thus, in a single-interrupt system, the response time is always more than 3 cycles and less than 9 cycles.

CHAPTER - 7

SERIAL DATA COMMUNICATION

OBJECTIVE

i. To familiarise the serial data communication of 8031 / 8051.

7.1 INTRODUCTION

It becomes a very tedious task to find out the opcodes and enter them into the trainer in the form of hex codes, when the program size becomes large. It then becomes a necessity to use a PC based assembler while developing large programs, for effective time utilisation. Now, the opcodes can be transferred from the PC to the trainer. We provide a **Data Communication** Package for such a transfer from the PC/trainer to the trainer/PC, which transfers data in a serial fashion.

In this chapter, we will be dealing with the usage of this Communication Package called **DATACOM**, with Micropower-i, Micro-51 and Micro-51 EB trainer kits.

Note that the messages given in **boldface**, in the succeeding paragraphs denote the messages that will be displayed on the host while working with DATACOM.

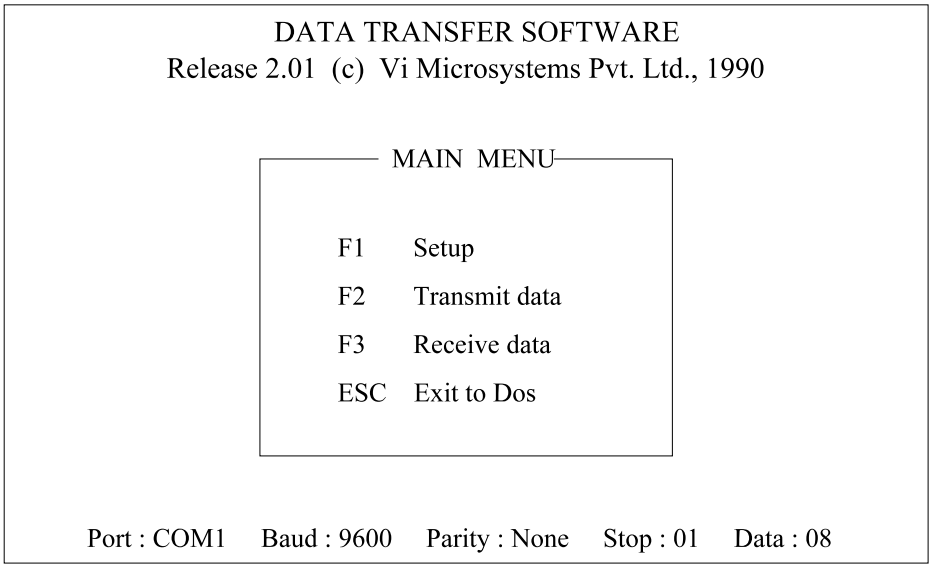
7.2 DATACOM - BASIC FEATURES:

This DATACOM is a Menu-oriented user friendly Serial Communication Package. You can either download to a trainer or upload to a host. Selection of COM1 or COM2, baud rates from 110 to 9600, parity bits and stop bits are user determined.

When your host is ready with the DOS prompt, type in,

A>DATACOM <CR>

Now the MAIN MENU is displayed as:



As seen, the Function keys F1, F2 and F3 are used for the three main operations. The three functions can also be selected using either Cursor keys or pressing the first letter of the function, for example "S" takes you to "Setup", "T" to "Transmit" and so on.

7.2.1 END OF FILE (EOF):

The DATACOM can transmit any type of data. To denote the End Of File to the receiving system, it sends out five question marks ("?????") as the EOF.

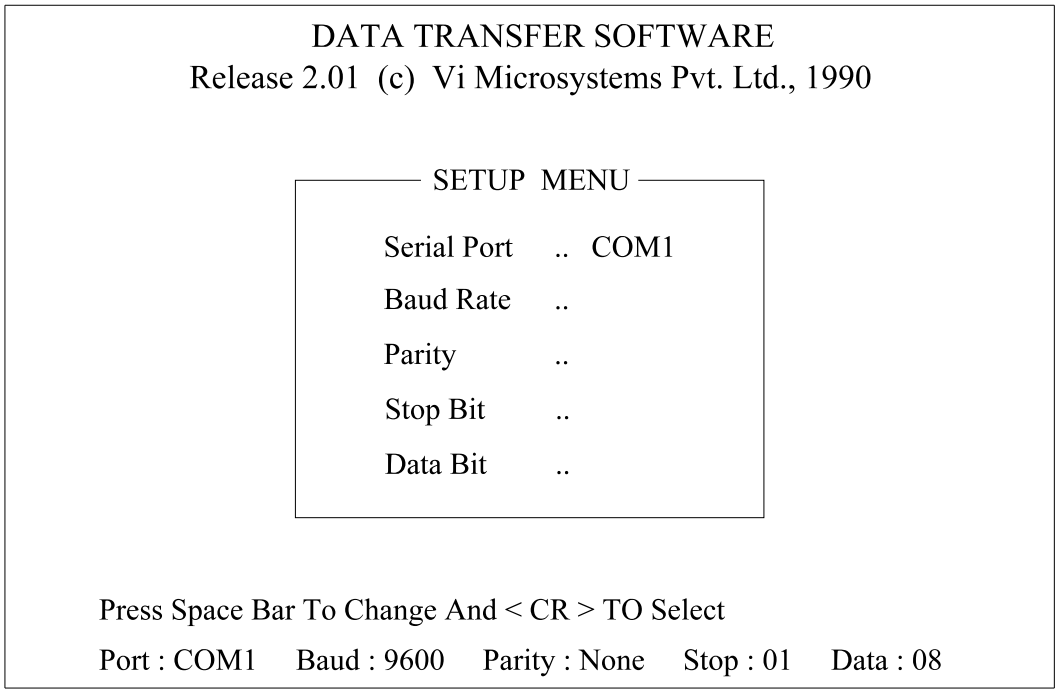
Similarly, the DATACOM expects "?????" as an EOF when it is receiving the data.

The software written for serial communication in Micropower-i, Micro-51 and Micro-51 EB trainer kits supports this EOF requirement of DATACOM.

The above mentioned are the features available in the DATACOM package. And now to the part which explains communication with Micropower-i, Micro-51 and Micro-51 EB trainers.

7.3 SETUP HOST SERIAL PORT:

Select the "Setup" option in the MAIN MENU using either "S" or "F1" or using Cursor keys. The SETUP MENU now appears on the screen as:



From the MENU, you have got options to change the,

Serial Port —— COM1, COM2

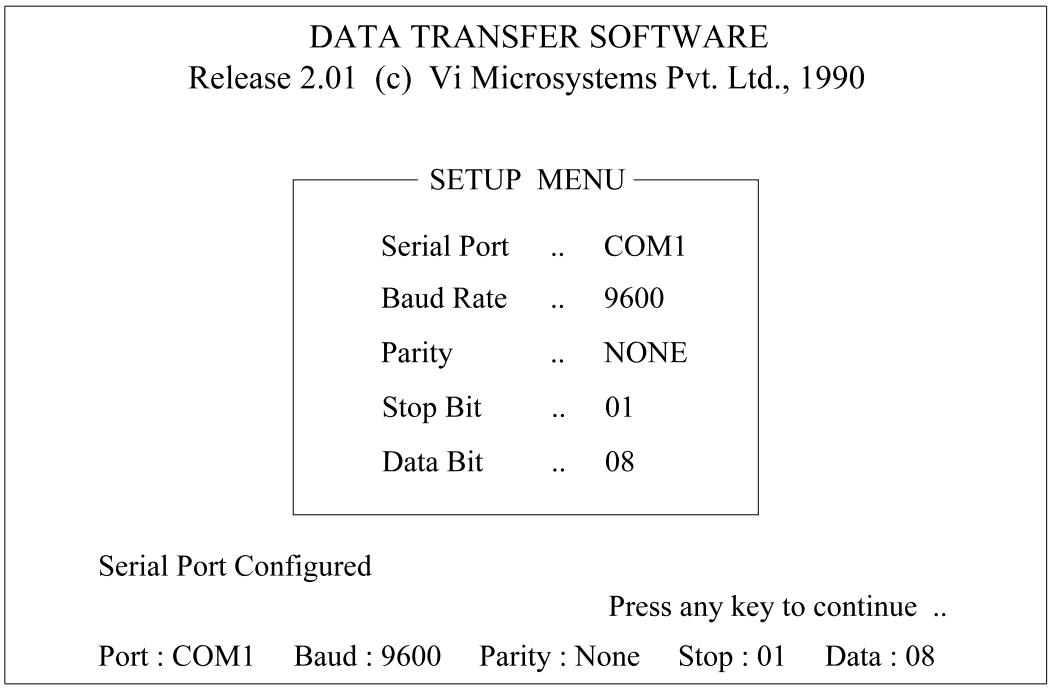
Baud rates —— 110, 150, 300, 600, 1200, 2400, 4800, 9600

Parity —— Even, Odd, None

Stop bits —— 1,2

Data bits —— 7,8

Now use the SPACEBAR to select the protocols and confirm your selection using <CR>. After selection of all the protocols, the current serial port setting is displayed as a Status in the MENU.



Now, the host system's serial port has been configured as per your selection and is ready for serial transmission.

NOTE:

i) Be sure that the other system that is sending/ receiving data is also configured with the same protocols. Else, proper transmission of data will not occur.

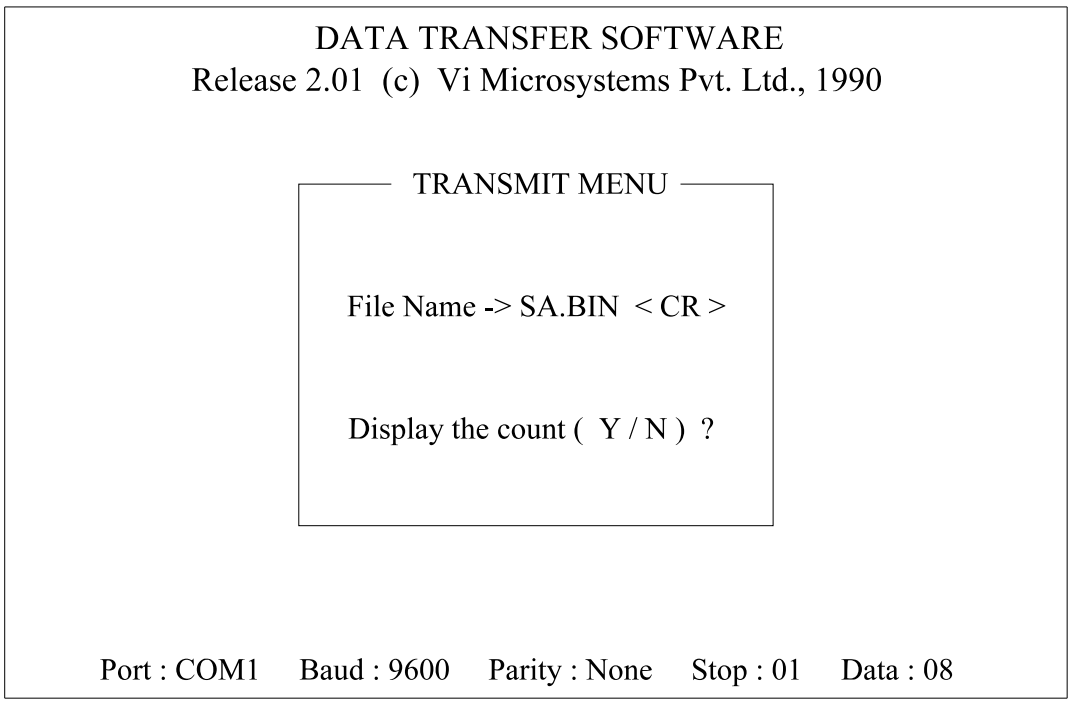
ii) Default settings of DATACOM are COM1 Serial Port, 9600 Baud Rate, No Parity, 1 Stop Bit and 8 Data bits.

7.4 TRANSMIT DATA TO TRAINER:

Ensuring yourself that both the host and the trainer are initialised with the same protocols, connect the Serial cable from the host to the trainer. For an IBM PC/XT, the cable is a 25-pin D female to a 9-pin D female, the 9-pin D female side going to the trainer. For an IBM PC/AT, it is a 9/25-pin D female to a 9-pin D female. Distinguish between the trainer side female connector and the host side connector.

In the Micropower-i, Micro-51 and Micro-51 EB trainer kits, the default Serial Port settings are 9600 baud, 8 data bits, no parity and 1 stop bit.

Select the "Transmit data" option in the MAIN MENU of DATACOM. Enter the file name, that is to be transmitted. Assuming a file name of "SA.BIN", the display is as:



Only if the file specified is found in the default drive and directory setting, will the option for displaying the count be displayed. Else, the user response whether to try again or not is confirmed as,

Cannot open the File SA.BIN Wish to try again (Y/N) ?

If the file is present, the display is as in above menu. Now, press "Y" or "N" depending upon whether you want the system to display the number of bytes that is being transferred or not. After confirmation of the display of the count, the system is ready for transmission as,

Press any key to continue...

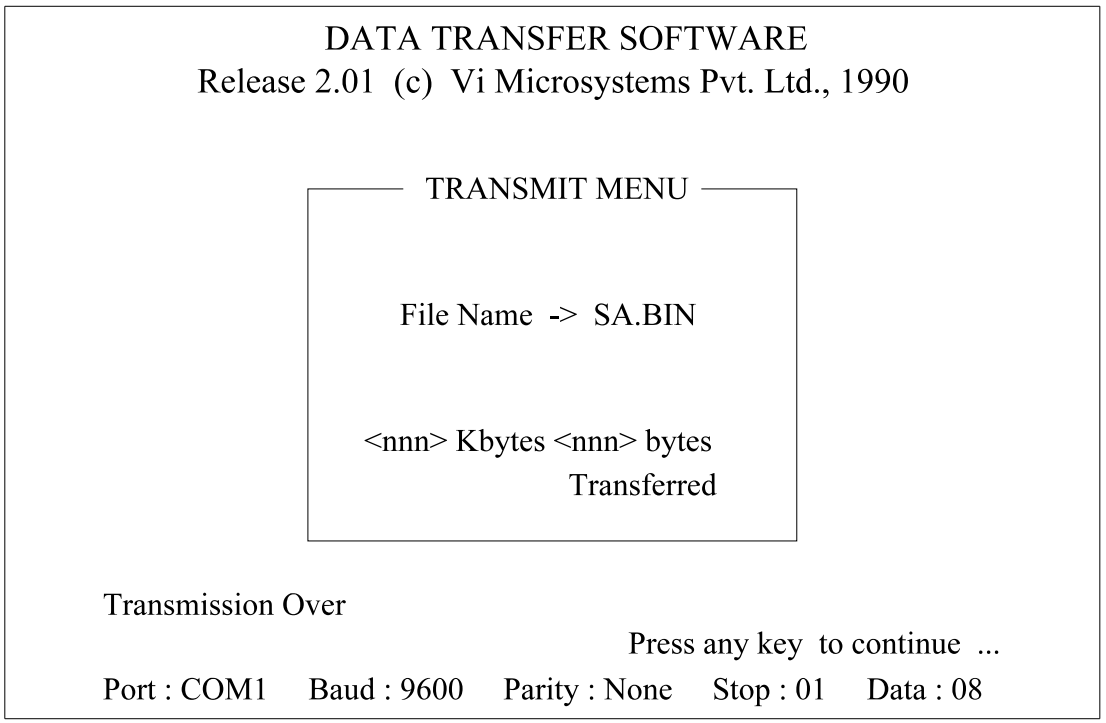
Now, setup the trainer in the receive mode. Have a quick reference at the Technical Reference manual to set the trainer in the receive mode using the SYNTAX SER, IN, address and NEXT.

Now , if the trainer is ready to receive data, press any key in the host to start transmission. During transmission, the message displayed is,

Press ESC to abort

Transmission in progress...

The count is displayed according to user setup to display or not. Without the display of the count, the transmission is fast. You can abort serial transmission halfway by using ESC key. After successful transmission of data, the display will be as given below:



Now, if the trainer's serial port is proper, then it will display command prompt, indicating successful reception of data.

In case, if the communication is not proper, then the display will be as follows:

Time out Error

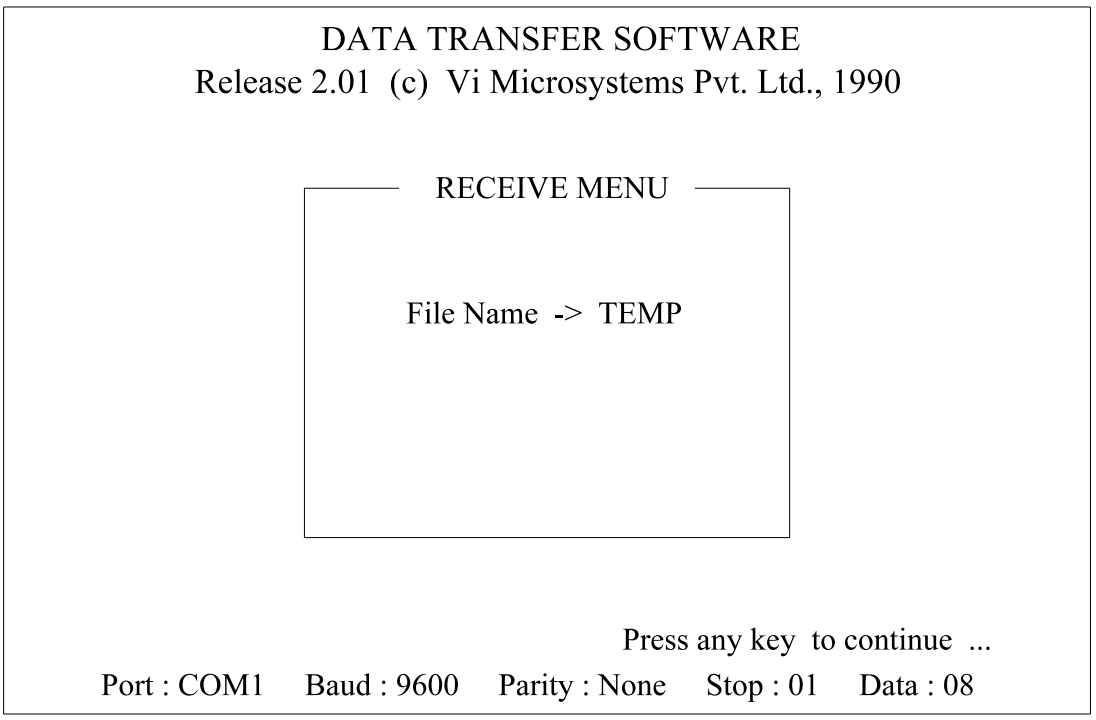
Press any key to continue...

Try once again now. Check for mismatches in the protocols of the two systems. Even now if the communication is not proper, then contact our Customer-Support Division for further clarification.

7.5 RECEIVE DATA FROM TRAINER:

Ensure that the protocols of the two systems are identical. If not, initialise properly before transmission of data from the trainer.

Selection of "R" or "F2" or the cursor keys takes you to the RECEIVE MENU of DATACOM. Now enter the file name under which the incoming serial data from the trainer will be stored. Assuming the file name to be "TEMP", the display is as:



Now the host is ready for reception. Now use the SYNTAX SER, OUT, Start address, NEXT, End address in the trainer to make it ready for serial transmission.

Put the host in the receive mode by pressing any key. Instantaneously, press the NEXT key in the trainer.

Notice one practical problem now. The HOST cannot be put in the receive mode for long. It checks for data only around 50 times and displays Error if it doesn't find any data at its serial port. So, do not put the host in the receive mode for long time. Send data from the trainer as soon as the host has come to the receive mode. Hence additional care must be taken during uploading. If a mismatch occurs between the two systems, or if the trainer does not send data within that specified 50 checks, the display is as:

Time out Error, Task aborted

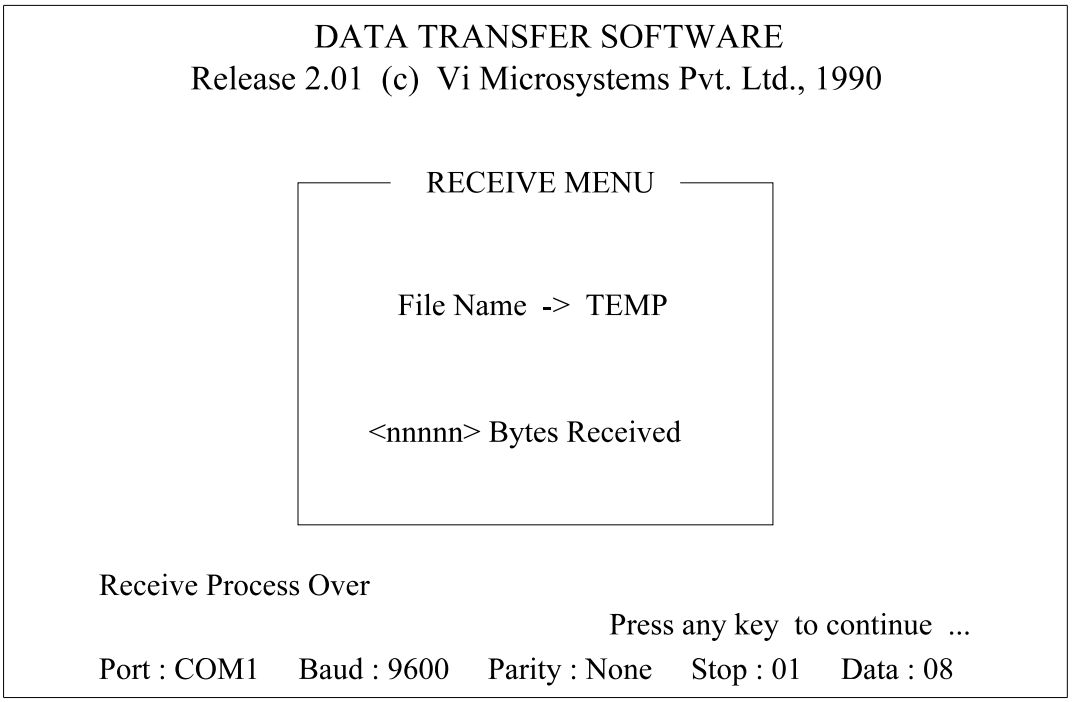
Press any key to continue...

When it is receiving data, the display is as,

Receiving to file TEMP

Ensure that the file you are transmitting contains the "?????" EOF mark.

After successful reception, the DATACOM displays as,



This ends the DATACOM's Serial Transmission and Reception of data. Quit DATACOM by using either "ESC" or "E" or using Cursor keys.

APPENDIX - A

80C51 PROGRAMMER'S GUIDE AND INSTRUCTION SET

Philips Semiconductors

80C51 Family

80C51 family programmer's guide and instruction set

PROGRAMMER'S GUIDE AND INSTRUCTION SET

Memory Organization

Program Memory

The 80C51 has separate address spaces for program and data memory. The Program memory can be up to 64k bytes long. The lower 4k can reside on-chip. Figure 1 shows a map of the 80C51 program memory.

The 80C51 can address up to 64k bytes of data memory to the chip. The MOVX instruction is used to access the external data memory.

The 80C51 has 128 bytes of on-chip RAM, plus a number of Special Function Registers (SFRs). The lower 128 bytes of RAM can be accessed either by direct addressing (MOV data addr) or by indirect addressing (MOV @Ri). Figure 2 shows the Data Memory organization.

Direct and Indirect Address Area

The 128 bytes of RAM which can be accessed by both direct and indirect addressing can be divided into three segments as listed below and shown in Figure 3.

1. Register Banks 0-3: Locations 0 through 1FH (32 bytes). The device after reset defaults to register bank 0. To use the other register banks, the user must select them in software. Each register bank contains eight 1-byte registers 0 through 7. Reset initializes the stack pointer to location 07H, and it is incremented once to start from location 08H, which is the first register (R0) of the second register bank. Thus, in order to use more than one register bank, the SP should be initialized to a different location of the RAM where it is not used for data storage (i.e., the higher part of the RAM).
2. Bit Addressable Area: 16 bytes have been assigned for this segment, 20H-2FH. Each one of the 128 bits of this segment can be directly addressed (0-7FH). The bits can be referred to in two ways, both of which are acceptable by most assemblers. One way is to refer to their address (i.e., 0-7FH). The other way is with reference to bytes 20H to 2FH. Thus, bits 0-7 can also be referred to as bits 20.0-20.7, and bits 8-FH are the same as 21.0-21.7, and so on. Each of the 16 bytes in this segment can also be addressed as a byte.
3. Scratch Pad Area: 30H through 7FH are available to the user as data RAM. However, if the stack pointer has been initialized to this area, enough bytes should be left aside to prevent SP data destruction.

Figure 2 shows the different segments of the on-chip RAM.

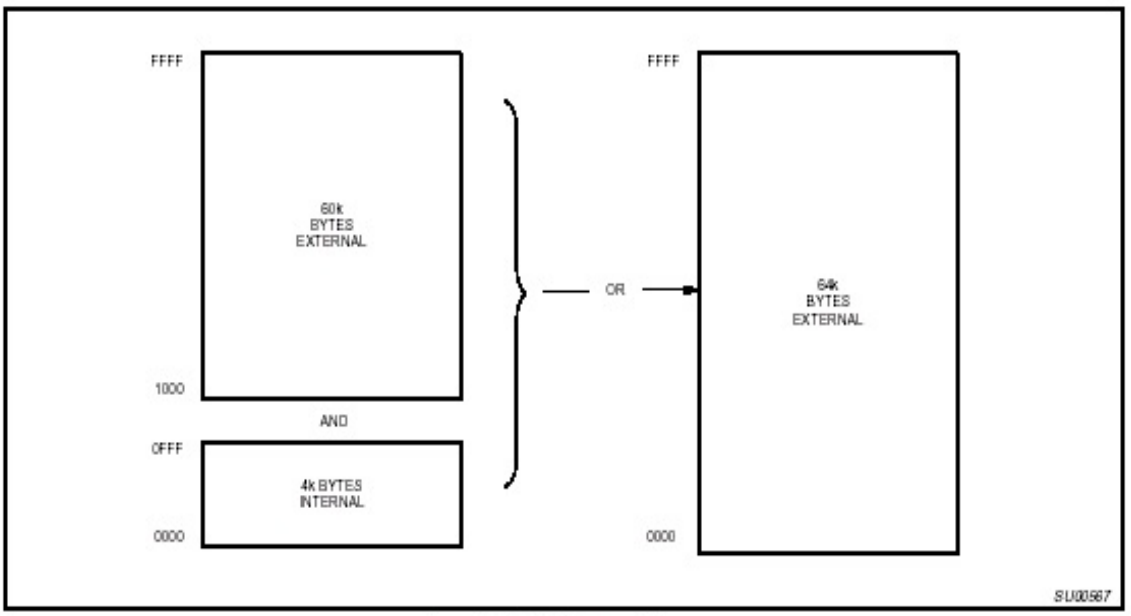


Figure 1. 80C51 Program Memory

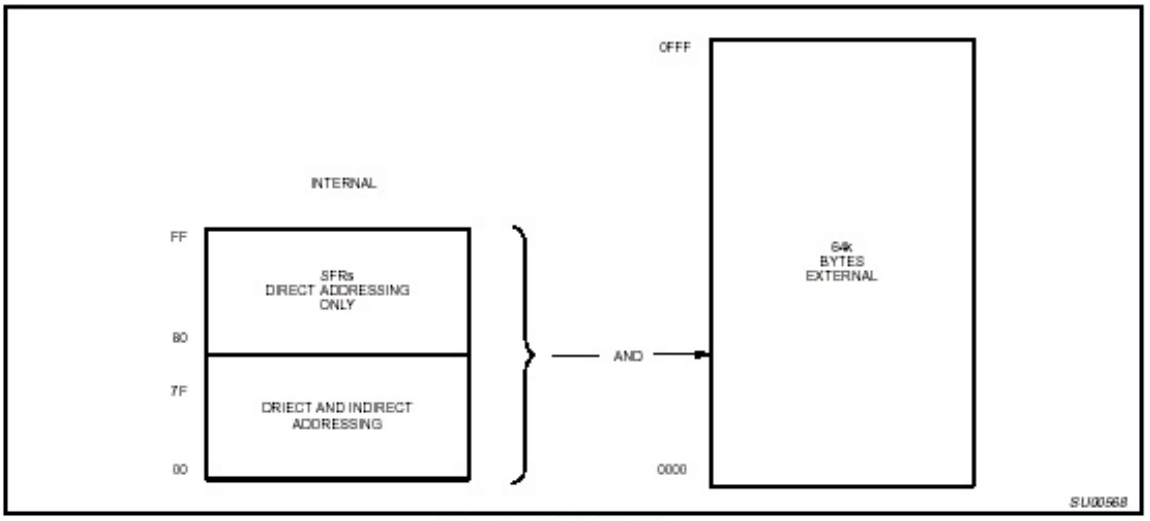


Figure 2. 80C51 Data Memory

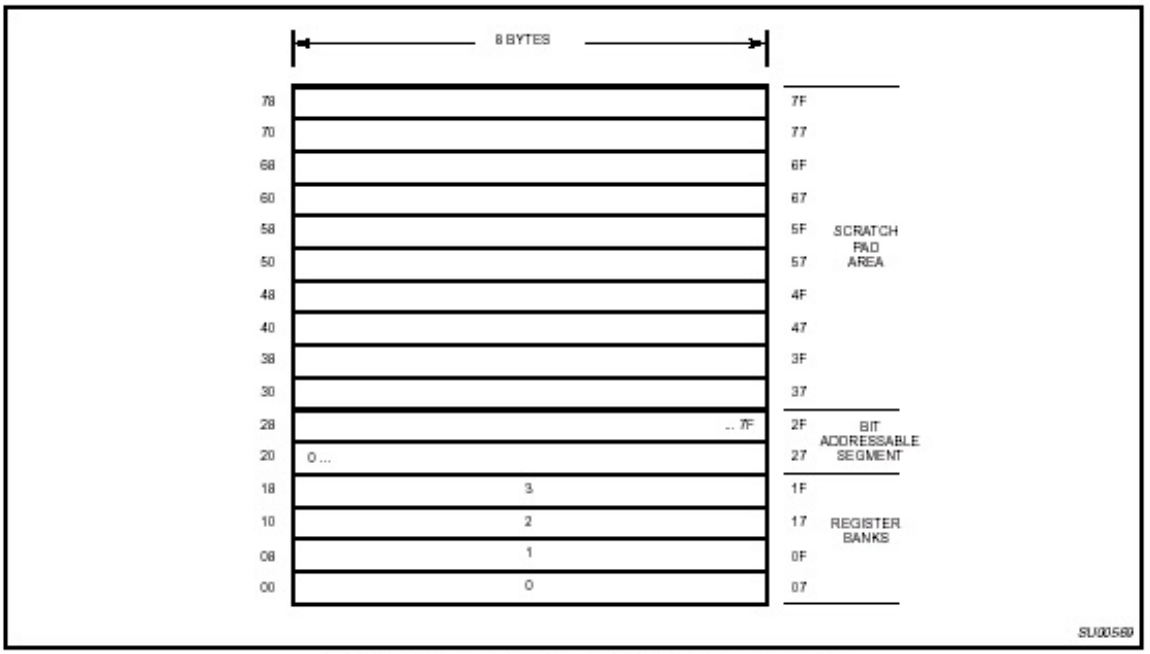


Figure 3. 128 Bytes of RAM Direct and Indirect Addressable

Table 1. 80C51 Special Function Registers

SYMBOL	DESCRIPTION	DIRECT ADDRESS	BIT ADDRESS, SYMBOL, OR ALTERNATIVE PORT FUNCTION MSB							LSB	RESET VALUE
ACC*	Accumulator	E0H	E7	E6	E5	E4	E3	E2	E1	E0	00H
B*	B register	F0H	F7	F6	F5	F4	F3	F2	F1	F0	00H
DPTR	Data pointer (2 bytes)										
DPH	Data pointer high	83H									00H
DPL	Data pointer low	82H									00H
			AF	AE	AD	AC	AB	AA	A9	A8	
IE*	Interrupt enable	A8H	EA	–	–	ES	ET1	EX1	ET0	EX0	0x000000B
			BF	BE	BD	BC	BB	BA	B9	B8	
IP*	Interrupt priority	B8H	–	–	–	PS	PT1	PX1	PT0	PX0	xx000000B
			87	86	85	84	83	82	81	80	
P0*	Port 0	80H	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	FFH
			97	96	95	94	93	92	91	90	
P1*	Port 1	90H	–	–	–	–	–	–	T2EX	T2	FFH
			A7	A6	A5	A4	A3	A2	A1	A0	
P2*	Port 2	A0H	A15	A14	A13	A12	A11	A10	A9	A8	FFH
			B7	B6	B5	B4	B3	B2	B1	B0	
P3*	Port 3	B0H	RD	WR	T1	T0	INT1	INT0	TxD	Rxd	FFH
PCON[1]	Power control	87H	SMOD	–	–	–	GF1	GF0	PD	IDL	0xxxxxxB
			D7	D6	D5	D4	D3	D2	D1	D0	
PSW*	Program status word	D0H	CY	AC	F0	RS1	RS0	OV	–	P	00H
SBUF	Serial data buffer	99H									xxxxxxxB
			9F	9E	9D	9C	9B	9A	99	98	
SCON*	Serial controller	98H	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	00H
SP	Stack pointer	81H									07H
			8F	8E	8D	8C	8B	8A	89	88	
TCON*	Timer control	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	
TH0	Timer high 0	8CH									00H
TH1	Timer high 1	8DH									00H
TL0	Timer low 0	8AH									00H
TL1	Timer low 1	8BH									00H
TMOD	Timer mode	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	00H

NOTES:
* Bit addressable
1. Bits GF1, GF0, PD, and IDL of the PCON register are not implemented on the NMOS 8051/8031.

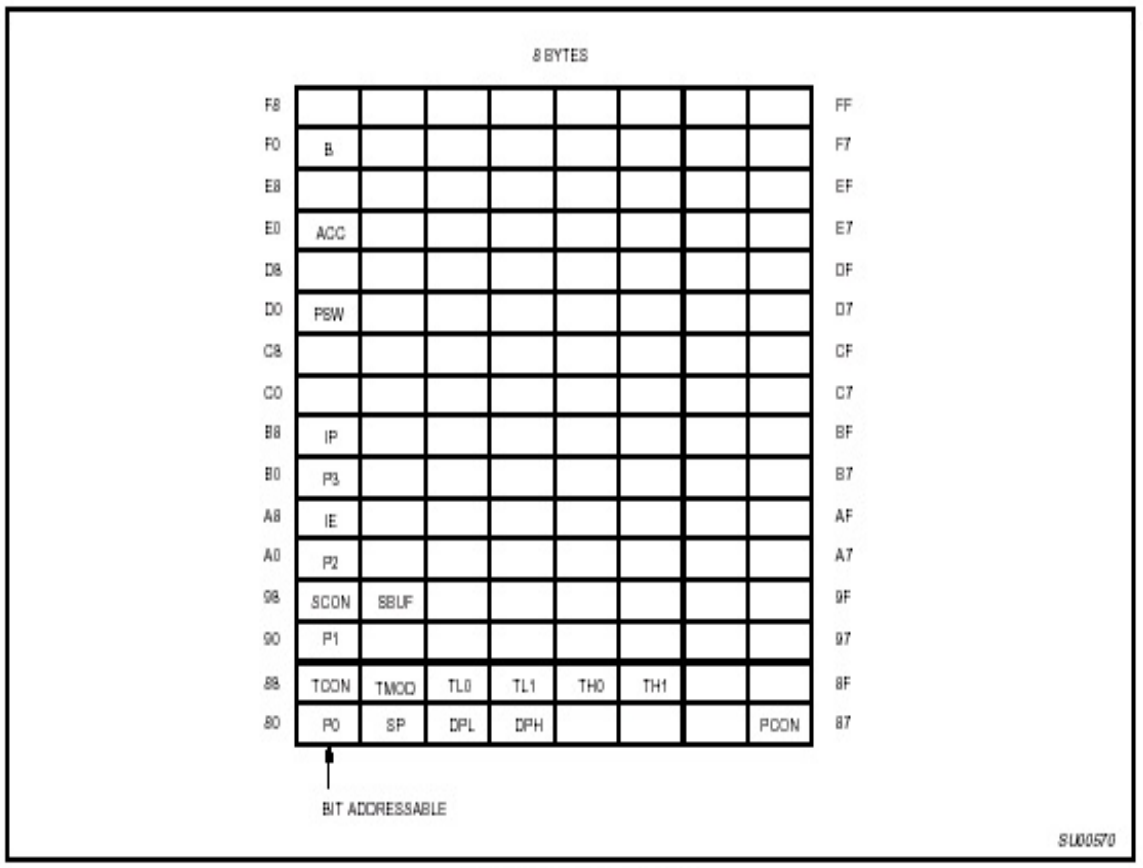


Figure 4. SFR Memory Map

Those SFRs that have their bits assigned for various functions are listed in this section. A brief description of each bit is provided for quick reference. For more detailed information refer to the Architecture Chapter of this book.

PSW: PROGRAM STATUS WORD. BIT ADDRESSABLE.

CY	AC	F0	RS1	RS0	OV	–	P

CY	PSW.7	Carry Flag.
AC	PSW.6	Auxiliary Carry Flag.
F0	PSW.5	Flag 0 available to the user for general purpose.
RS1	PSW.4	Register Bank selector bit 1 (SEE NOTE 1).
RS0	PSW.3	Register Bank selector bit 0 (SEE NOTE 1).
OV	PSW.2	Overflow Flag.
–	PSW.1	Usable as a general purpose flag.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of '1' bus in the accumulator.

NOTE:
1. The value presented by RS0 and RS1 selects the corresponding register bank.

RS1	RS0	REGISTER BANK	ADDRESS
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

PCON: POWER CONTROL REGISTER. NOT BIT ADDRESSABLE.

SMOD	–	–	–	GF1	GF0	PD	IDL

SMOD	Double baud rate bit. If Timer 1 is used to generate baud rate and SMOD = 1, the baud rate is doubled when the Serial Port is used in modes 1, 2, or 3.
–	Not implemented, reserved for future use.*
–	Not implemented reserved for future use.*
–	Not implemented reserved for future use.*
GF1	General purpose flag bit.
GF0	General purpose flag bit.
PD	Power Down Bit. Setting this bit activates Power Down operation in the 80C51. (Available only in CMOS.)
IDL	Idle mode bit. Setting this bit activates Idle Mode operation in the 80C51. (Available only in CMOS.)

If 1s are written to PD and IDL at the same time, PD takes precedence.

* User software should not write 1s to reserved bits. These bits may be used in future 8051 products to invoke new features.

INTERRUPTS:

To use any of the interrupts in the 80C51 Family, the following three steps must be taken.

1. **Set the EA (enable all) bit in the IE register to 1.**
2. **Set the corresponding individual interrupt enable bit in the IE register to 1.**
3. **Begin the interrupt service routine at the corresponding Vector Address of that interrupt. See Table below.**

INTERRUPT SOURCE	VECTOR ADDRESS
IE0	0003H
TF0	000BH
IE1	0013H
TF1	001BH
RI & TI	0023H

In addition, for external interrupts, pins INT0 and INT1 (P3.2 and P3.3) must be set to 1, and depending on whether the interrupt is to be level or transition activated, bits IT0 or IT1 in the TCON register may need to be set to 1.

ITx = 0 level activated

ITx = 1 transition activated

IE: INTERRUPT ENABLE REGISTER. BIT ADDRESSABLE.

If the bit is 0, the corresponding interrupt is disabled. If the bit is 1, the corresponding interrupt is enabled.

EA	–	–	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
—	IE.6	Not implemented, reserved for future use.*
—	IE.5	Not implemented, reserved for future use.*
ES	IE.4	Enable or disable the serial port interrupt.
ET1	IE.3	Enable or disable the Timer 1 overflow interrupt.
EX1	IE.2	Enable or disable External Interrupt 1.
ET0	IE.1	Enable or disable the Timer 0 overflow interrupt.
EX0	IE.0	Enable or disable External Interrupt 0.

* User software should not write 1s to reserved bits. These bits may be used in future 80C51 products to invoke new features.

ASSIGNING HIGHER PRIORITY TO ONE OR MORE INTERRUPTS:

In order to assign higher priority to an interrupt the corresponding bit in the IP register must be set to 1.

Remember that while an interrupt service is in progress, it cannot be interrupted by a lower or same level interrupt.

PRIORITY WITHIN LEVEL:

Priority within level is only to resolve simultaneous requests of the same priority level.

From high to low, interrupt sources are listed below:

IE0
TF0
IE1
TF1
RI or TI

IP: INTERRUPT PRIORITY REGISTER. BIT ADDRESSABLE.

If the bit is 0, the corresponding interrupt has a lower priority and if the bit is 1 the corresponding interrupt has a higher priority.

–	–	–	PS	PT1	PX1	PT0	PX0

–	IP.7	Not implemented, reserved for future use.*
–	IP.6	Not implemented, reserved for future use.*
–	IP.5	Not implemented, reserved for future use.*
PS	IP.4	Defines the Serial Port interrupt priority level.
PT1	IP.3	Defines the Timer 1 interrupt priority level.
PX1	IP.2	Defines External Interrupt 1 priority level.
PT0	IP.1	Defines the Timer 0 interrupt priority level.
PX0	IP.0	Defines the External Interrupt 0 priority level.

* User software should not write 1s to reserved bits. These bits may be used in future 80C51 products to invoke new features.

TCON: TIMER/COUNTER CONTROL REGISTER. BIT ADDRESSABLE.

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1	TCON.7	Timer 1 overflow flag. Set by hardware when the Timer/Counter 1 overflows. Cleared by hardware as processor vectors to the interrupt service routine.
TR1	TCON.6	Timer 1 run control bit. Set/cleared by software to turn Timer/Counter 1 ON/OFF.
TF0	TCON.5	Timer 0 overflow flag. Set by hardware when the Timer/Counter 0 overflows. Cleared by hardware as processor vectors to the service routine.
TR0	TCON.4	Timer 0 run control bit. Set/cleared by software to turn Timer/Counter 0 ON/OFF.
IE1	TCON.3	External Interrupt 1 edge flag. Set by hardware when External Interrupt edge is detected. Cleared by hardware when interrupt is processed.
IT1	TCON.2	Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.
IE0	TCON.1	External Interrupt 0 edge flag. Set by hardware when External Interrupt edge detected. Cleared by hardware when interrupt is processed.
IT0	TCON.0	Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.

TMOD: TIMER/COUNTER MODE CONTROL REGISTER. NOT BIT ADDRESSABLE.

GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
Timer 1				Timer 0			

GATE	When TRx (in TCON) is set and GATE = 1, TIMER/COUNTERx will run only while INTx pin is high (hardware control). When GATE = 0, TIMER/COUNTERx will run only while TRx = 1 (software control).
C/$\overline{T}$	Timer or Counter selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from Tx input pin).
M1	Mode selector bit. (NOTE 1)
M0	Mode selector bit. (NOTE 1)

NOTE 1:

M1	M0		Operating Mode
0	0	0	13-bit Timer (8048 compatible)
0	1	1	16-bit Timer/Counter
1	0	2	8-bit Auto-Reload Timer/Counter
1	1	3	(Timer 0) TL0 is an 8-bit Timer/Counter controlled by the standart Timer 0 control bits. TH0 is an8-bit Timer and is controlled by Timer 1 control bits.
1	1	3	(Timer 1) Timer/Counter 1 stopped.

TIMER SET-UP

Tables 2 through 5 give some values for TMOD which can be used to set up Timer 0 in different modes.

It is assumed that only one timer is being used at a time. If it is desired to run Timers 0 and 1 simultaneously, in any mode, the value in TMOD for Timer 0 must be ORed with the value shown for Timer 1 (Tables 5 and 6).

For example, if it is desired to run Timer 0 in mode 1 GATE (external control), and Timer 1 in mode 2 COUNTER, then the value that must be loaded into TMOD is 69H (09H from Table 2 ORed with 60H from Table 5).

Moreover, it is assumed that the user, at this point, is not ready to turn the timers on and will do that at a different point in the program by setting bit TRx (in TCON) to 1.

TIMER/COUNTER 0

Table 2. As a Timer:

MODE	TIMER 0 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13-bit Timer	00H	08H
1	16-bit Timer	01H	09H
2	8-bit Auto-Reload	02H	0AH
3	Two 8-bit Timers	03H	0BH

Table 3. As a Counter:

MODE	COUNTER 0 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13-bit Timer	04H	0CH
1	16-bit Timer	05H	0DH
2	8-bit Auto-Reload	06H	0EH
3	One 8-bit Counter	07H	0FH

NOTES:
1. The timer is turned ON/OFF by setting/clearing bit TR0 in the software.
2. The Timer is turned ON/OFF by the 1-to-0 transition on INT0 (P3.2) when TR0 = 1 (hardware control).

TIMER/COUNTER 1

Table 4. As a Timer:

MODE	TIMER 1 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13-bit Timer	00H	80H
1	16-bit Timer	10H	90H
2	8-bit Auto-Reload	20H	A0H
3	Does not run	30H	B0H

Table 5. As a Counter:

MODE	COUNTER 1 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13-bit Timer	40H	C0H
1	16-bit Timer	50H	D0H
2	8-bit Auto-Reload	60H	E0H
3	Not available	–	–

NOTES:
1. The timer is turned ON/OFF by setting/clearing bit TR1 in the software.
2. The Timer is turned ON/OFF by the 1-to-0 transition on $\overline{INT1}$ (P3.2) when TR1 = 1 (hardware control).

SCON: SERIAL PORT CONTROL REGISTER. BIT ADDRESSABLE.

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SCON.7	Serial Port mode specifier. (NOTE 1)
SM1	SCON.6	Serial Port mode specifier. (NOTE 1)
SM2	SCON.5	Enables the multiprocessor communication feature in modes 2 & 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0. (See Table 6.)
REN	SCON.4	Set/Cleared by software to Enable/Disable reception.
TB8	SCON.3	The 9th bit that will be transmitted in modes 2 & 3. Set/Cleared by software.
RB8	SCON.2	In modes 2 & 3, is the 9th data bit that was received. In mode 1, if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.
TI	SCON.1	Transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes. Must be cleared by software.
RI	SCON.0	Receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or halfway through the stop bit time in the other modes (except see SM2). Must be cleared by software.

NOTE 1:

SM0	SM1	Mode	Description	Baud Rate
0	0	0	Shift Register	$F_{OSC.}/12$
0	1	1	8-bit UART	Variable
1	0	2	9-bit UART	$F_{OSC.}/64$ or $F_{OSC.}/32$
1	1	3	9-bit UART	Variable

SERIAL PORT SET-UP:

Table 6.

MODE	SCON	SM2 VARIATION
0 1 2 3	10H 50H 90H D0H	Single Processor Environment (SM2 = 0)
0 1 2 3	NA 70H B0H F0H	Multiprocessor Environment (SM2 = 1)

GENERATING BAUD RATES

Serial Port in Mode 0:

Mode 0 has a fixed baud rate which is 1/12 of the oscillator frequency. To run the serial port in this mode none of the Timer/Counters need to be set up. Only the SCON register needs to be defined.

$$\text{Baud Rate} = \frac{\text{Osc Freq}}{12}$$

Serial Port in Mode 1:

Mode 1 has a variable baud rate. The baud rate is generated by Timer 1.

USING TIMER/COUNTER 1 TO GENERATE BAUD RATES:

For this purpose, Timer 1 is used in mode 2 (Auto-Reload). Refer to Timer Setup section of this chapter.

$$\text{Baud Rate} = \frac{K \times \text{Osc Freq}}{32 \times 12 \times [256 - (TH1)]}$$

If SMOD = 0, then K = 1.
If SMOD = 1, then K = 2 (SMOD is in the PCON register).

Most of the time the user knows the baud rate and needs to know the reload value for TH1.

$$TH1 = 256 - \frac{K \times \text{Osc Freq}}{384 \times \text{baud rate}}$$

TH1 must be an integer value. Rounding off TH1 to the nearest integer may not produce the desired baud rate. In this case, the user may have to choose another crystal frequency.

Since the PCON register is not bit addressable, one way to set the bit is logical ORing the PCON register (i.e., ORL PCON,#80H). The address of PCON is 87H.

SERIAL PORT IN MODE 2:

The baud rate is fixed in this mode and is 1/32 or 1/64 of the oscillator frequency, depending on the value of the SMOD bit in the PCON register.

In this mode none of the Timers are used and the clock comes from the internal phase 2 clock.

SMOD = 1, Baud Rate = 1/32 Osc Freq.

SMOD = 0, Baud Rate = 1/64 Osc Freq.

To set the SMOD bit: ORL PCON,#80H. The address of PCON is 87H.

SERIAL PORT IN MODE 3:

The baud rate in mode 3 is variable and sets up exactly the same as in mode 1.

80C51 FAMILY INSTRUCTION SET

Table 7. 80C51 Instruction Set Summary

Interrupt Response Time: Refer to Hardware Description Chapter.

Instructions that Affect Flag Settings[1]

Instruction	Flag			Instruction	Flag		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	0	X		ANL C,/bit	X		
DIV	0	X		ORL C,bit	X		
DA	X			ORL C,/bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

[1]Note that operations on SFR byte address 208 or bit addresses 209-215 (i.e., the PSW or bits in the PSW) will also affect flag settings.

Notes on instruction set and addressing modes:

Rn	Register R7-R0 of the currently selected Register Bank.
direct	8-bit internal data location's address. This could be an Internal Data RAM location (0-127) or a SFR [i.e., I/O port, control register, status register, etc. (128-255)].
@Ri	8-bit internal data RAM location (0-255) addressed indirectly through register R1 or R0.
#data	8-bit constant included in the instruction.
#data 16	16-bit constant included in the instruction
addr 16	16-bit destination address. Used by LCALL and LJMP. A branch can be anywhere within the 64k-byte Program Memory address space.
addr 11	11-bit destination address. Used by ACALL and AJMP. The branch will be within the same 2k-byte page of program memory as the first byte of the following instruction.
rel	Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is −128 to +127 bytes relative to first byte of the following instruction.
bit	Direct Addressed bit in Internal Data RAM or Special Function Register.

MNEMONIC		DESCRIPTION	BYTE	OSCILLATOR PERIOD
ARITHMETIC OPERATIONS				
ADD	A,Rn	Add register to Accumulator	1	12
ADD	A,direct	Add direct byte to Accumulator	2	12
ADD	A,@Ri	Add indirect RAM to Accumulator	1	12
ADD	A,#data	Add immediate data to Accumulator	2	12
ADDC	A,Rn	Add register to Accumulator with carry	1	12
ADDC	A,direct	Add direct byte to Accumulator with carry	2	12
ADDC	A,@Ri	Add indirect RAM to Accumulator with carry	1	12
ADDC	A,#data	Add immediate data to A_{CC} with carry	2	12
SUBB	A,Rn	Subtract Register from A_{CC} with borrow	1	12
SUBB	A,direct	Subtract direct byte from A_{CC} with borrow	2	12
SUBB	A,@Ri	Subtract indirect RAM from A_{CC} with borrow	1	12
SUBB	A,#data	Subtract immediate data from A_{CC} with borrow	2	12
INC	A	Increment Accumulator	1	12
INC	Rn	Increment register	1	12

Table 7. 80C51 Instruction Set Summary (Continued)

MNEMONIC		DESCRIPTION	BYTE	OSCILLATOR PERIOD
ARITHMETIC OPERATIONS (Continued)				
INC	direct	Increment direct byte	2	12
INC	@Ri	Increment indirect RAM	1	12
DEC	A	Decrement Accumulator	1	12
DEC	Rn	Decrement Register	1	12
DEC	direct	Decrement direct byte	2	12
DEC	@Ri	Decrement indirect RAM	1	12
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A and B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12
LOGICAL OPERATIONS				
ANL	A,Rn	AND Register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,#data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct byte	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12
RL	A	Rotate Accumulator left	1	12
RLC	A	Rotate Accumulator left through the carry	1	12
RR	A	Rotate Accumulator right	1	12
RRC	A	Rotate Accumulator right through the carry	1	12
SWAP	A	Swap nibbles within the Accumulator	1	12
DATA TRANSFER				
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12

Table 7. 80C51 Instruction Set Summary (Continued)

MNEMONIC		DESCRIPTION	BYTE	OSCILLATOR PERIOD
DATA TRANSFER (Continued)				
MOV	A,#data	Move immediate data to Accumulator	2	12
MOV	Rn,A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	RN,#data	Move immediate data to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct,#data	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12
MOV	@Ri,direct	Move direct byte to indirect RAM	2	24
MOV	@Ri,#data	Move immediate data to indirect RAM	2	12
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	24
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A_{CC}	1	24
MOVC	A,@A+PC	Move Code byte relative to PC to A_{CC}	1	24
MOVX	A,@Ri	Move external RAM (8-bit addr) to A_{CC}	1	24
MOVX	A,@DPTR	Move external RAM (16-bit addr) to A_{CC}	1	24
MOVX	A,@Ri,A	Move A_{CC} to external RAM (8-bit addr)	1	24
MOVX	@DPTR,A	Move A_{CC} to external RAM (16-bit addr)	1	24
PUSH	direct	Push direct byte onto stack	2	24
POP	direct	Pop direct byte from stack	2	24
XCH	A,Rn	Exchange register with Accumulator	1	12
XCH	A,direct	Exchange direct byte with Accumulator	2	12
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	12
XCHD	A,@Ri	Exchange low-order digit indirect RAM with A_{CC}	1	12
BOOLEAN VARIABLE MANIPULATION				
CLR	C	Clear carry	1	12
CLR	bit	Clear direct bit	2	12
SETB	C	Set carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to carry	2	24
ANL	C,/bit	AND complement of direct bit to carry	2	24
ORL	C,bit	OR direct bit to carry	2	24
ORL	C,/bit	OR complement of direct bit to carry	2	24
MOV	C,bit	Move direct bit to carry	2	12
MOV	bit,C	Move carry to direct bit	2	24
JC	rel	Jump if carry is set	2	24
JNC	rel	Jump if carry not set	2	24

Table 7. 80C51 Instruction Set Summary (Continued)

MNEMONIC		DESCRIPTION	BYTE	OSCILLATOR PERIOD
BOOLEAN VARIABLE MANIPULATION (Continued)				
JB	rel	Jump if direct bit is set	3	24
JNB	rel	Jump if direct bit is not set	3	24
JBC	bit,rel	Jump if direct bit is set and clear bit	3	24
PROGRAM BRANCHING				
ACALL	addr11	Absolute subroutine call	2	24
LCALL	addr16	Long subroutine call	3	24
RET		Return from subroutine	1	24
RETI		Return from interrupt	1	24
AJMP	addr11	Absolute jump	2	24
LJMP	addr16	Long jump	3	24
SJMP	rel	Short jump (relative addr)	2	24
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is zero	2	24
JNZ	rel	Jump if Accumulator is not zero	2	24
CJNE	A,direct,rel	Compare direct byte to A_{CC} and jump if not equal	3	24
CJNE	A,#data,rel	Compare immediate to A_{CC} and jump if not equal	3	24
CJNE	RN,#data,rel	Compare immediate to register and jump if not equal	3	24
CJNE	@Ri,#data,rel	Compare immediate to indirect and jump if not equal	3	24
DJNZ	Rn,rel	Decrement register and jump if not zero	2	24
DJNZ	direct,rel	Decrement direct byte and jump if not zero	3	24
NOP		No operation	1	12

INSTRUCTION DEFINITIONS

ACALL addr11

Function: Absolute Call

Description: ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the Stack Pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, opcode bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2k block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

Example: Initially SP equals 07H. The label "SUBRTN" is at program memory location 0345 H. After executing the instruction,

ACALL SUBRTN

at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

Bytes: 2

Cycles: 2

Encoding:

a10 a9 a8 1	0 0 0 1	a7 a6 a5 a4	a3 a2 a1 a0

Operation: ACALL
(PC) ← (PC) + 2
(SP) ← (SP) + 1
(SP) ← (PC_{7-0})
(SP) ← (SP) + 1
(SP) ← (PC_{15-8})
(PC_{10-0}) ← page address

ADDC A,<src-byte>

Function: Add with Carry

Description: ADDC simultaneously adds the byte variable indicated, the carry flag and the Accumulator contents, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) with the carry flag set. The instruction,

ADDC A,R0

will leave 6EH (01101110B) in the Accumulator with AC cleared and both the Carry flag and OV set to 1.

ADDC A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	1 r r r

Operation: ADDC
(A) ← (A) + (C) + (R_n)

ADDC A,direct

Bytes: 2

Cycles: 1

Encoding:

0 0 1 1	0 1 0 1	direct address

Operation: ADDC
(A) ← (A) + (C) + (direct)

ADDC A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	0 1 1 i

Operation: ADDC
(A) ← (A) + (C) + ((R_i))

ADDC A,#data

Bytes: 2

Cycles: 1

Encoding:

0 0 1 1	0 1 0 0	immediate data

Operation: ADDC
(A) ← (A) + (C) + #data

ADD A,<src-byte>

Function: Add

Description: ADD adds the byte variable indicated to the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B). The instruction,

ADD A,R0

will leave 6DH (01101101B) in the Accumulator with the AC flag cleared and both the Carry flag and OV set to 1.

ADD A,Rn

Bytes: 1

Cycles: 1

Encoding:

0	0	1	0	1	r	r	r

Operation:
ADD
(A) ← (A) + (R_n)

ADD A,direct

Bytes: 2

Cycles: 1

Encoding:

0	0	1	0	0	1	0	1	direct address

Operation:
ADD
(A) ← (A) + (direct)

ADD A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0	0	1	0	0	1	1	i

Operation:
ADD
(A) ← (A) + ((R_i))

ADD A,#data

Bytes: 2

Cycles: 1

Encoding:

0	0	1	0	0	1	0	0	immediate data

Operation:
ADD
(A) ← (A) + #data

AJMP addr11

Function: Absolute Jump

Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (*after* incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2k block of program memory as the first byte of the instruction following AJMP.

Example: The label "JMPADR" is at program memory location 0123H. The instruction,

AJMP JMPADR

is at location 0345H and will load the PC with 0123H.

Bytes: 2

Cycles: 2

Encoding:

a10 a9 a8 0	0 0 0 1	a7 a6 a5 a4	a3 a2 a1 a0

Operation: AJMP

$(PC) \leftarrow (PC) + 2$

$(PC_{10-0}) \leftarrow$ page address

ANL <dest-byte>,<src-byte>

Function: Logical-AND for byte variables

Description: ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: If the Accumulator holds 0C3H (11000011B) and register 0 holds 55H (01010101B) then the instruction,

ANL A,R0

will leave 41H (01000001B) in the Accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the Accumulator at run-time. The instruction,

ANL P1,#01110011B

will clear bits 7, 3, and 2 of output port 1.

ANL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 1 0 1	1 r r r

Operation: ANL
(A) ← (A) ∧ (R_n)

ANL A,direct

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 1 0 1	direct address

Operation: ANL
(A) ← (A) ∧ (direct)

ANL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 1 0 1	0 1 1 i

Operation: ANL
(A) ← (A) ∧ ((R_i))

ANL A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 1 0 0	immediate data

Operation: ANL
(A) ← (A) ∧ #data

ANL direct,A

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 0 1 0	direct address

Operation: ANL
(A) ←(direct) ∧ (A)

ANL direct,#data

Bytes: 3

Cycles: 2

Encoding:

0 1 0 1	0 0 1 1	direct address	immediate data

Operation: ANL
(direct) ←(direct) ∧ #data

ANL C,<src-bit>

Function: Logical-AND for bit variables

Description: If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected.* No other flags are affected.

Only direct addressing is allowed for the source operand.

Example: Set the carry flag if, and only if, P1.0 = 1, ACC.7 = 1, and OV = 0:

```
MOV     C,P1.0   ;LOAD CARRY WITH INPUT PIN STATE
ANL     C,ACC.7;AND CARRY WITH ACCUM. BIT 7
ANL     C,/OV    ;AND WITH INVERSE OF OVERFLOW FLAG
```

ANL C,bit

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 0 1 0	bit address

Operation: ANL

$(C) \leftarrow (C) \wedge (bit)$

ANL C,/bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 1	0 0 0 0	bit address

Operation: ANL

$(C) \leftarrow (C) \wedge \overline{(bit)}$

CJNE <dest-byte>,<src-byte>,rel

Function: Compare and Jump If Not Equal

Description: CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

The first two operands allow four addressing mode combinations: the Accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Example: The Accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence,

```
        CJNE    R7,#60H,NOT_EQ
;       ...     ....        ;       R7 = 60H.
NOT_EQ  JC      REQ_LOW     ;       IF R7 < 60H.
;       ...     ....        ;       R7 > 60H.
```

sets the carry flag and branches to the instruction at label NOT_EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to Port 1 is also 34H, then the instruction,

WAIT: CJNE A,P1,WAIT

clears the carry flag and continues with the next instruction in sequence, since the Accumulator does equal the data read from P1. (If some other value was being input on P1, the program will loop at this point until the P1 data changes to 34H.)

CJNE A,direct,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 0 1	direct address	rel. address

Operation:

(PC) ← (PC) + 3
IF (A) < > *(direct)*
THEN
(PC) ← (PC) + *relative offset*
IF (A) < *(direct)*
THEN.
(C) ← 1
ELSE
(C) ← 0

CJNE A,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 0 0	immediate data	rel. address

Operation:

(PC) ← (PC) + 3
IF (A) < > *data*
THEN
(PC) ← (PC) + *relative offset*
IF (A) < *data*
THEN
(C) ← 1
ELSE
(C) ← 0

CJNE Rn,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	1 r r r	immediate data	rel. address

Operation:

(PC) ← (PC) + 3
IF (R_n) < > *data*
THEN
(PC) ← (PC) + *relative offset*
IF (R_n) < *data*
THEN
(C) ← 1
ELSE
(C) ← 0

CJNE @Ri,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 1 i	immediate data	rel. address

Operation:

(PC) ← (PC) + 3
IF ((R_i)) < > *data*
THEN
(PC) ← (PC) + *relative offset*
IF ((R_i)) < *data*
THEN
(C) ← 1
ELSE
(C) ← 0

CPL A

Function: Complement Accumulator

Description: Each bit of the Accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to a zero and vice-versa. No flags are affected.

Example: The Accumulator contains 5CH (01011100B). The instruction,

CPL A

will leave the Accumulator set to 0A3H (10100011B).

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	0 1 0 0

Operation: CPL
(A) ← ¬ (A)

CPL bit

Function: Complement bit

Description: The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CPL can operate on the carry or any directly addressable bit.

Note: When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.

Example: Port 1 has previously been written with 5DH (01011101B). The instruction sequence,

CPL P1.1

CPL P1.2

will leave the port set to 5BH (01011011B).

CPL C

Bytes: 1

Cycles: 1

Encoding:

1 0 1 1	0 0 1 1

Operation: CPL
(C) ← ¬ (C)

CPL bit

Bytes: 2

Cycles: 1

Encoding:

1 0 1 1	0 0 1 0	bit address

Operation: CPL
(bit) ← ¬ (bit)

CPL A

Function: Complement Accumulator

Description: Each bit of the Accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to a zero and vice-versa. No flags are affected.

Example: The Accumulator contains 5CH (01011100B). The instruction,

CPL A

will leave the Accumulator set to 0A3H (10100011B).

Bytes: 1

Cycles: 1

Encoding:

1	1	1	1	0	1	0	0

Operation: CPL
(A) ← ¬ (A)

CPL bit

Function: Complement bit

Description: The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CPL can operate on the carry or any directly addressable bit.

Note: When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.

Example: Port 1 has previously been written with 5DH (01011101B). The instruction sequence,

CPL P1.1

CPL P1.2

will leave the port set to 5BH (01011011B).

CPL C

Bytes: 1

Cycles: 1

Encoding:

1	0	1	1	0	0	1	1

Operation: CPL
(C) ← ¬ (C)

CPL bit

Bytes: 2

Cycles: 1

Encoding:

1	0	1	1	0	0	1	0	bit address

Operation: CPL
(bit) ← ¬ (bit)

DA A

Function: Decimal-adjust Accumulator for Addition

Description: DA A adjusts the eight-bit value in the Accumulator resulting from the earlier addition of two variable (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If Accumulator bits 3-0 are greater than nine (xxx1010-xxx1111), or if the AC flag is one, six is added to the Accumulator, producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxx-111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the Accumulator, depending on initial Accumulator and PSW conditions.

Note: DA A *cannot* simply convert a hexadecimal number in the Accumulator to BCD notation, nor does DA A apply to decimal subtraction.

Example: The Accumulator holds the value 56H (01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67. The carry flag is set.. The instruction sequence,

```
ADDC    A,R3
DA      A
```

will first perform a standard two's-complement binary addition, resulting in the value 0BEH (10111110B) in the Accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the Accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56, 67, and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the Accumulator initially holds 30H (representing the digits of 30 decimal), the the instruction sequence,

```
ADD     A,#99H
DA      A
```

will leave the carry set and 29H in the Accumulator, since 30 + 99 = 129. The low-order byte of the sum can be interpreted to mean 30 – 1 = 29.

Bytes: 1

Cycles: 1

Encoding:

1	1	0	1	0	1	0	0

Operation:

```
DA
–contents of Accumulator are BCD
IF   [[(A3-0) > 9] ∨ [(AC) = 1]]
     THEN(A3-0) ← (A3-0) + 6
          AND
IF   [[(A7-4) > 9] ∨ [(C) = 1]]
     THEN(A7-4) ← (A7-4) + 6
```

DEC byte

Function: Decrement

Description: The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original data will be read from the output data latch, *not* the input pin.

Example: Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,

```
DEC   @R0
DEC   R0
DEC   @R0
```

will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH.

DEC A

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 1 0 0

Operation:
DEC
$(A) \leftarrow (A) - 1$

DEC Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	1 r r r

Operation:
DEC
$(R_n) \leftarrow (R_n) - 1$

DEC direct

Bytes: 2

Cycles: 1

Encoding:

0 0 0 1	0 1 0 1	direct address

Operation:
DEC
(direct) ← (direct) – 1

DEC @Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 1 1 i

Operation:
DEC
$((R_i)) \leftarrow ((R_i)) - 1$

DIV AB

Function: Divide

Description: DIV AB divides the unsigned eight-bit integer in the Accumulator by the unsigned eight-bit integer in register B.

The Accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

Exception: if B had originally contained 00H, the values returned in the Accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Example: The Accumulator contains 251 (0FBH or 11111011B) and B contains 18 (12H or 00010010B). The instruction,

DIV AB

will leave 13 in the Accumulator (0DH or 00001101B) and the value 17 (11H or 00010001B) in B, since 251 = (13 x 18) + 17. Carry and OV will both be cleared.

Bytes: 1

Cycles: 4

Encoding:

1	0	0	0	0	1	0	0

Operation: DIV

$(A)_{15-8} \leftarrow (A)/(B)$

$(B)_{7-0}$

DJNZ <byte>,<rel-addr>

Function: Decrement and Jump if Not Zero

Description: DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

```
DJNZ    40H,LABEL_1
DJNZ    50H,LABEL_2
DJNZ    60H,LABEL_3
```

will cause a jump to the instruction at LABEL_2 with the values 00h, 6FH, and 15H in the three RAM locations. The first jump was *not* taken because the result was zero.

This instruction provides a simple was of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

```
        MOV     R2,#8
TOGGLE: CPL     P1.7
        DJNZ    R2,TOGGLE
```

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output Port 1. Each pulse will last three machine cycles, two for DJNZ and one to alter the pin.

DJNZ Rn,rel

Bytes: 2

Cycles: 2

Encoding:

1 1 0 1	1 r r r	rel. address

Operation:

```
DJNZ
(PC) ← (PC) + 2
(Rn) ← (Rn) – 1
IF (Rn) > 0 or (Rn) < 0
    THEN
        (PC) ← (PC) + rel
```

DJNZ direct,rel

Bytes: 3

Cycles: 2

Encoding:

1 1 0 1	0 1 0 1	direct data	rel. address

Operation:

```
DJNZ
(PC) ← (PC) + 2
(direct) ← (direct) – 1
IF (direct) > 0 or (direct) < 0
    THEN
        (PC) ← (PC) + rel
```

INC <byte>

Function: Increment

Description: INC increments the indicated variable by 1. An original value of 0FFH will overflow to 00H. No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Register 0 contains 7EH (01111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

```
INC     @R0
INC     R0
INC     @R0
```

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

INC A

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 1 0 0

Operation: INC
(A) ← (A) + 1

INC Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	1 r r r

Operation: INC
(R_n) ← (R_n) + 1

INC direct

Bytes: 2

Cycles: 1

Encoding:

0 0 0 0	0 1 0 1	direct address

Operation: INC
(direct) ← (direct) + 1

INC @Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 1 1 i

Operation: INC
((R_i)) ← ((R_i)) + 1

INC DPTR

Function: Increment Data Pointer

Description: Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2^{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order byte (DPH). No flags are affected.

This is the only 16-bit register which can be incremented.

Example: Registers DPH and DPL contain 12H and 0FEH, respectively. The instruction sequence,

```
INC     DPTR
INC     DPTR
INC     DPTR
```

will change DPH and DPL to 13H and 01H.

Bytes: 1

Cycles: 2

Encoding:

1 0 1 0	0 0 1 1

Operation:

```
INC
(DPTR) ← (DPTR) + 1
```

JB bit,rel

Function: Jump if Bit set

Description: If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56 (01010110B). The instruction sequence,

```
JB      P1.2,LABEL1
JB      ACC.2,LABEL2
```

will cause program execution to branch to the instruction at label LABEL2.

Bytes: 3

Cycles: 2

Encoding:

0 0 1 0	0 0 0 0	bit address	rel. address

Operation:

```
JB
(PC) ← (PC) + 3
IF (bit) = 1
   THEN
        (PC) ← (PC) + rel
```

JBC bit,rel

Function: Jump if Bit is set and Clear bit

Description: If the indicated bit is a one, branch to the address indicated; otherwise proceed with the next instruction. *The bit will not be cleared if it is already a zero.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note: When this instruction is used to test an output pin, the value used as the original data will read from the output data latch, not the input pin.

Example: The Accumulator holds 56H (01010110B). The instruction sequence,

```
JBC     ACC.3,LABEL1
JBC     ACC.2,LABEL2
```

will cause program execution to continue at the instruction identified by the LABEL2, with the Accumulator modified to 52H (01010010B).

Bytes: 3

Cycles: 2

Encoding:

0 0 0 1	0 0 0 0	bit address	rel. address

Operation:

```
JBC
(PC) ← (PC) + 3
IF (bit) = 1
   THEN
(bit) ← 0
(PC) ← (PC) + rel
```

JC rel

Function: Jump if Carry is set

Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Example: The carry flag is cleared. The instruction sequence,

```
JC      LABEL1
CPL     C
JC      LABEL2
```

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 0 0	0 0 0 0	rel. address

Operation:

```
JC
(PC) ← (PC) + 2
IF (C) = 1
   THEN
                (PC) ← (PC) + rel
```

JMP @A+DPTR

Function: Jump Indirect

Description: Add the eight-bit unsigned contents of the Accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^{16}): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the Accumulator nor the Data Pointer is altered. No flags are affected.

Example: An even number from 0 to 6 is in the Accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP_TBL:

```
        MOV   DPTR,#JMP_TBL
        JMP   @A+DPTR
JMP_TBL: AJMP  LABEL0
        AJMP  LABEL1
        AJMP  LABEL2
        AJMP  LABEL3
```

If the Accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

Bytes: 1

Cycles: 2

Encoding:

0 1 1 1	0 0 1 1

Operation: JMP

(PC) ← (A) + (DPTR)

JNB bit,rel

Function: Jump if Bit Not set

Description: If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified*. No flags are affected.

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56H (01010110B). The instruction sequence,

```
JNB     P1.3,LABEL1
JNB     ACC.3,LABEL2
```

will cause program execution to continue at the instruction at label LABEL2.

Bytes: 3

Cycles: 2

Encoding:

0 0 1 1	0 0 0 0	bit address	rel. address

Operation: JNB

(PC) ← (PC) + 3

IF (bit) = 0

THEN

(PC) ← (PC) + rel

JNC rel

Function: Jump if Carry Not set

Description: If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

Example: The carry flag is set. The instruction sequence,

```
JNC     LABEL1
CPL     C
JNC     LABEL2
```

will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 0 1	0 0 0 0	rel. address

Operation:

```
JNC
(PC) ← (PC) + 2
IF (C) = 0
    THEN
(PC) ← (PC) + rel
```

JNZ rel

Function: Jump if Accumulator Not Zero

Description: If any bit of the Accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally holds 00H. The instruction sequence,

```
JNZ     LABEL1
INC     A
JNZ     LABEL2
```

will set the Accumulator to 01H and continue at label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 1	0 0 0 0	rel. address

Operation:

```
JNZ
(PC) ← (PC) + 2
IF A ≠ 0
    THEN  (PC) ← (PC) + rel
```

JZ rel

Function: Jump If Accumulator Zero

Description: If all bits of the Accumulator are zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally holds 01H. The instruction sequence,

```
JZ      LABEL1
DEC     A
JZ      LABEL2
```

will change the Accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 0	0 0 0 0	rel. address

Operation:

JZ
(PC) ← (PC) + 2
IF A = 0
THEN (PC) ← (PC) + rel

LCALL addr16

Function: Long Call

Description: LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the Stack Pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64k-byte program memory address space. No flags are affected.

Example: Initially the Stack Pointer equals 07H. The label "SUBRTN" is assigned to program memory location 1234H. After executing the instruction,

```
LCALL   SUBRTN
```

at location 0123H, the Stack Pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1235H.

Bytes: 3

Cycles: 2

Encoding:

0 0 0 1	0 0 1 0	addr15-addr8	addr7-addr0

Operation:

LCALL
(PC) ← (PC) + 3
(SP) ← (SP) + 1
((SP)) ← (PC_{7-0})
(SP) ← (SP) + 1
((SP)) ← (PC_{15-8})
(PC) ← $addr_{15-0}$

LJMP addr16 (Implemented in 87C751 and 87C752 for in-circuit emulation only.)

Function: Long Jump

Description: LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64k program memory address space. No flags are affected.

Example: The label "JMPADR" is assigned to the instruction at program memory location 1234H. The instruction,

LJMP JMPADR

at location 0123H will load the program counter with 1234H.

Bytes: 3

Cycles: 2

Encoding:

0 0 0 0	0 0 1 0	addr15-addr8	addr7-addr0

Operation: LJMP

$(PC) \leftarrow addr_{15-0}$

MOV <dest-byte>,<src-byte>

Function: Move byte variable

Description: The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

Example: Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH). The instruction sequence,

```
MOV     R0,#30H     ;R0 < = 30H
MOV     A,@R0       ;A < = 40H
MOV     R1,A        ;R1 < = 40H
MOV     B,@R1       ;B < = 10H
MOV     @R1,P1      ;RAM (40H) < = 0CAH
MOV     P2,P1       ;P2 #0CAH
```

leaves the value 30H in register 0, 40H in both the Accumulator and register 1, 10H in register B, and 0CAH (11001010B) both in RAM location 40H and output on port 2.

MOV A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	1 r r r

Operation: MOV

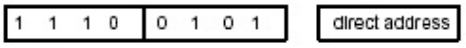
$(A) \leftarrow (R_n)$

*MOV A,direct

Bytes: 2

Cycles: 1

Encoding:



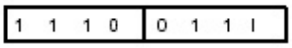
Operation: MOV
(A) ← (direct)

MOV A,@RI

Bytes: 1

Cycles: 1

Encoding:



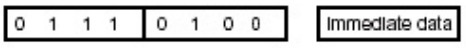
Operation: MOV
(A) ← ((Ri))

MOV A,#data

Bytes: 2

Cycles: 1

Encoding:



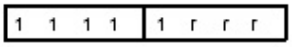
Operation: MOV
(A) ← #data

MOV Rn,A

Bytes: 1

Cycles: 1

Encoding:



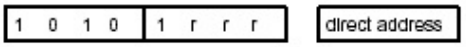
Operation: MOV
(Rn) ← (A)

MOV Rn,direct

Bytes: 2

Cycles: 2

Encoding:



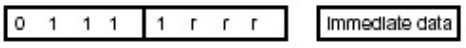
Operation: MOV
(Rn) ← (direct)

MOV Rn,#data

Bytes: 2

Cycles: 1

Encoding:



Operation: MOV
(Rn) ← #data

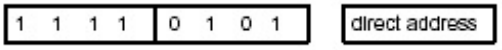
*MOV A,ACC is not a valid instruction.

MOV direct,A

Bytes: 2

Cycles: 1

Encoding:



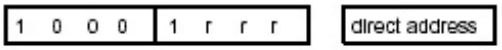
Operation: MOV
(direct) ← (A)

MOV direct,Rn

Bytes: 2

Cycles: 2

Encoding:



Operation: MOV
(direct) ← (R_n)

MOV direct,direct

Bytes: 3

Cycles: 2

Encoding:

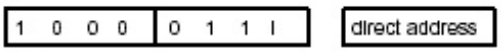
Operation: MOV
(direct) ← (direct)

MOV direct,@RI

Bytes: 2

Cycles: 2

Encoding:



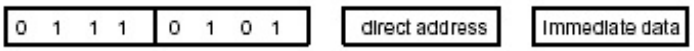
Operation: MOV
(direct) ← ((R_i))

MOV direct,#data

Bytes: 3

Cycles: 2

Encoding:



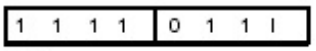
Operation: MOV
(direct) ← #data

MOV @RI,A

Bytes: 1

Cycles: 1

Encoding:



Operation: MOV
((R_i)) ← (A)

MOV @RI,direct

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	0 1 1 I	direct address

Operation: MOV
((Ri)) ← (direct)

MOV @RI,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1	0 1 1 I	immediate data

Operation: MOV
((Ri)) ← #data

MOV <dest-bit>,<src-bit>

Function: Move bit data

Description: The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

Example: The carry flag is originally set. The data present at input Port 3 is 11000101B. The data previously written to output Port 1 is 35H (00110101B). The instruction sequence,

```
MOV   P1.3,C
MOV   C,P3.3
MOV   P1.2,C
```

will leave the carry cleared and change Port 1 to 39H (00111001B).

MOV C,bit

Bytes: 2

Cycles: 1

Encoding:

1 0 1 0	0 0 1 0	bit address

Operation: MOV
(C) ← (bit)

MOV bit,C

Bytes: 2

Cycles: 2

Encoding:

1 0 0 1	0 0 1 0	bit address

Operation: MOV
(bit) ← (C)

MOV DPTR,#data16

Function: Load Data Pointer with a 16-bit constant

Description: The Data Pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

This is the only instruction which moves 16 bits of data at once.

Example: The instruction,

MOV DPTR,#1234H

will load the value 1234H into the Data Pointer: DPH will hold 12H and DPL will hold 34H.

Bytes: 3

Cycles: 2

Encoding:

1 0 0 1	0 0 0 0	Immed. data15-8	Immed. data7-0

Operation: MOV

(DPTR) ← (#data$_{15-0}$)

DPH ■ DPL ← #data$_{15-8}$ ■ #data$_{7-0}$

MOVC A,@A+<base-reg>

Function: Move Code byte

Description: The MOVC instructions load the Accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit Accumulator contents and the contents of a sixteen-bit base register, which may be either the Data Pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the Accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Example: A value between 0 and 3 is in the Accumulator. The following instructions will translate the value in the Accumulator to one of four values defined by the DB (define byte) directive:

```
REL_PC:  INC   A
         MOVC  A,@A+PC
         RET
         DB    66H
         DB    77H
         DB    88H
         DB    99H
```

If the subroutine is called with the Accumulator equal to 01H, it will return with 77H in the Accumulator. The INC A before the MOVC instruction is needed to "get around" the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the Accumulator instead.

MOVC A,@A+DPTR

Bytes: 1

Cycles: 2

Encoding:

1 0 0 1	0 0 1 1

Operation: MOVC

(A) ← ((A) + (DPTR))

MOVC A,@A+PC

Bytes: 1

Cycles: 2

Encoding:

1	0	0	0	0	0	1	1

Operation: MOVC
(PC) ← (PC) + 1
(A) ← ((A) + (PC))

MOVX <dest-byte>,<src-byte> (Not Implemented In the 8XC752 or 8XC752)

Function: Move External

Description: The MOVX instructions transfer data between the Accumulator and a byte of external data memory, hence the "X" appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, The Data Pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64k bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the Data Pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

Example: An external 256 byte RAM using multiplexed address/data lines is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

```
MOVX  A,@R1
MOVX  @R0,A
```

copies the value 56H into both the Accumulator and external RAM location 12H.

MOVX A,@RI

Bytes: 1

Cycles: 2

Encoding:

1	1	1	0	0	0	1	I

Operation: MOVX
(A) ← ((Ri))

MOVX A,@DPTR

Bytes: 1

Cycles: 2

Encoding:

1	1	1	0	0	0	0	0

Operation: MOVX
(A) ← ((DPTR))

MOVX @RI,A

Bytes: 1

Cycles: 2

Encoding:

1	1	1	1	0	0	1	I

Operation: MOVX
$((R_i)) \leftarrow (A)$

MOVX @DPTR,A

Bytes: 1

Cycles: 2

Encoding:

1	1	1	1	0	0	0	0

Operation: MOVX
((DPTR)) ← (A)

MUL AB

Function: Multiply

Description: MUL AB multiplies the unsigned eight-bit integers in the Accumulator and register B. The low-order byte of the sixteen-bit product is left in the Accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

Example: Originally the Accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H).The instruction,

MUL AB

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the Accumulator is cleared. The overflow flag is set, carry is cleared.

Bytes: 1

Cycles: 4

Encoding:

1	0	1	0	0	1	0	0

Operation: MUL
$(A)_{7-0} \leftarrow (A) \times (B)$
$(B)_{15-8}$

NOP

Function: No Operation

Description: Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

Example: It is desired to produce a low-going output pulse on bit 7 of Port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming are enabled) with the instruction sequence,

```
CLR    P2.7
NOP
NOP
NOP
NOP
SETB   P2.7
```

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 0 0 0

Operation: NOP
(PC) ← (PC) + 1

ORL <dest-byte>,<src-byte>

Function: Logical-OR for byte variables

Description: ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: If the Accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

```
ORL   A,R0
```

will leave the Accumulator holding the value 0D7H (11010111B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the Accumulator at run-time. The instruction,

```
ORL   P1,#00110010B
```

will set bits 5, 4, and 1 of output Port 1.

ORL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 1 0 0	1 r r r

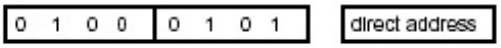
Operation: ORL
(A) ← (A) ∨ (R_n)

ORL A,direct

Bytes: 2

Cycles: 1

Encoding:



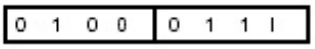
Operation: ORL
(A) ← (A) ∨ (direct)

ORL A,@RI

Bytes: 1

Cycles: 1

Encoding:



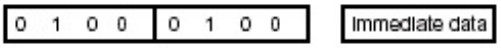
Operation: ORL
(A) ← (A) ∨ ((Ri))

ORL A,#data

Bytes: 2

Cycles: 1

Encoding:



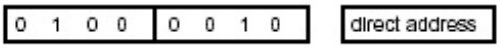
Operation: ORL
(A) ← (A) ∨ #data

ORL direct,A

Bytes: 2

Cycles: 1

Encoding:



Operation: ORL
(direct) ← (direct) ∨ (A)

ORL direct,#data

Bytes: 3

Cycles: 2

Encoding:

Operation: ORL
(direct) ← (direct) ∨ #data

ORL C,<src-bit>

Function: Logical-OR for bit variables

Description: Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

Example: Set the carry flag if and only if P1.0 = 1, ACC.7 = 1, or OV = 0:

```
ORL   C,P1.0     ;LOAD CARRY WITH INPUT PIN P10
ORL   C,ACC.7    ;OR CARRY WITH THE ACC. BIT 7
ORL   C,/OV      ;OR CARRY WITH THE INVERSE OF OV.
```

ORL C,bit

Bytes: 2

Cycles: 2

Encoding:

0 1 1 1	0 0 1 0	bit address

Operation: ORL
(C) ← (C) ∨ (bit)

ORL C,/bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	0 0 0 0	bit address

Operation: ORL
(C) ← (C) ∨ ($\overline{\text{bit}}$)

POP direct

Function: Pop from stack

Description: The contents of the internal RAM location addressed by the Stack Pointer is read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

Example: The Stack Pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,

```
POP   DPH
POP   DPL
```

will leave the Stack Pointer equal to the value 30H and the Data Pointer set to 0123H. At this point the instruction,

```
POP   SP
```

will leave the Stack Pointer set to 20H. Note that in this special case the Stack Pointer was decremented to 2FH before being loaded with the value popped (20H).

Bytes: 2

Cycles: 2

Encoding:

1 1 0 1	0 0 0 0	direct address

Operation:

```
POP
(direct) ← ((SP))
(SP) ← (SP) – 1
```

PUSH direct

Function: Push onto stack

Description: The Stack Pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the Stack Pointer. Otherwise no flags are affected.

Example: On entering an interrupt routine the Stack Pointer contains 09H. The Data Pointer holds the value 0123H. The instruction sequence,

```
PUSH   DPL
PUSH   DPH
```

will leave the Stack Pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 0BH, respectively.

Bytes: 2

Cycles: 2

Encoding:

1 1 0 0	0 0 0 0	direct address

Operation:

```
PUSH
(SP) ← (SP) + 1
((SP)) ← (direct)
```

RET

Function:	Return from subroutine
Description:	RET pops the high- and low-order bytes of the PC successively from the stack, decrementing the Stack Pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.
Example:	The Stack Pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction, RET will leave the Stack Pointer equal to the value 09H. Program execution will continue at location 0123H.
Bytes:	1
Cycles:	2
Encoding:	0 0 1 0 | 0 0 1 0
Operation:	RET $(PC_{15-8}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$ $(PC_{7-0}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$

RETI

Function:	Return from interrupt
Description:	RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The Stack Pointer is left decremented by two. No other registers are affected; the PSW is not automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt has been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.
Example:	The Stack Pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction, RETI will leave the Stack Pointer equal to 09H and return program execution to location 0123H.
Bytes:	1
Cycles:	2
Encoding:	0 0 1 1 | 0 0 1 0
Operation:	RETI $(PC_{15-8}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$ $(PC_{7-0}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$

RL A

Function:	Rotate Accumulator Left
Description:	The eight bits in the Accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.
Example:	The Accumulator holds the value 0C5H (11000101B). The instruction, RL A leaves the Accumulator holding the value 8BH (10001011B) with the carry unaffected.
Bytes:	1
Cycles:	1
Encoding:	0 0 1 0 | 0 0 1 1
Operation:	RL $(A_{n+1}) \leftarrow (A_n)$, n = 0 – 6 (A0) ← (A7)

RLC A

Function:	Rotate Accumulator Left through the Carry flag
Description:	The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.
Example:	The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction, RLC A leaves the Accumulator holding the value 8AH (10001010B) with the carry set.
Bytes:	1
Cycles:	1
Encoding:	0 0 1 1 | 0 0 1 1
Operation:	RLC $(A_{n+1}) \leftarrow (A_n)$, n = 0 – 6 (A0) ← (C) (C) ← (A7)

RR A

Function:	Rotate Accumulator Right
Description:	The eight bits in the Accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.
Example:	The Accumulator holds the value 0C5H (11000101B). The instruction, RR A leaves the Accumulator holding the value 0E2H (11100010B) with the carry unaffected.
Bytes:	1
Cycles:	1
Encoding:	0 0 0 0 | 0 0 1 1
Operation:	RR $(A_n) \leftarrow (A_{n+1})$, n = 0 – 6 (A7) ← (A0)

RRC A

Function:	Rotate Accumulator Right through the Carry flag
Description:	The eight bits in the Accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original state of the carry flag moves into the bit 7 position. No other flags are affected.
Example:	The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction, RRC A leaves the Accumulator holding the value 62 (01100010B) with the carry set.
Bytes:	1
Cycles:	1
Encoding:	0 0 0 1 | 0 0 1 1
Operation:	RRC $(A_n) \leftarrow (A_{n+1})$, n = 0 – 6 (A7) ← (C) (C) ← (A0)

SETB <bit>

Function: Set Bit

Description: SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.

Example: The carry flag is cleared. Output Port 1 has been written with the value 34H (00110100B). The instructions,

```
SETB  C
SETB  P1.0
```

will leave the carry flag set to 1 and change the data output on Port 1 to 35H (00110101B).

SETB C

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 0 1 1

Operation:

```
SETB
(C) ← 1
```

SETB bit

Bytes: 2

Cycles: 1

Encoding:

1 1 0 1	0 0 1 0	bit address

Operation:

```
SETB
(bit) ← 1
```

SJMP rel

Function: Short Jump

Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 126 bytes preceding this instruction to 127 bytes following it.

Example: The label "RELADR" is assigned to an instruction at program memory location 0123H. The instruction,

```
SJMP  RELADR
```

will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.

(*Note:* Under the above conditions the instruction following SJMP will be at 102H. Therefore, the displacement byte of the instruction will be the relative offset (0123H-0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH would be a one-instruction infinite loop.)

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 0 0 0	rel. address

Operation:

```
SJMP
(PC) ← (PC) + 2
(PC) ← (PC) + rel
```

SUBB A, <src-byte>

Function: Subtract with borrow

Description: SUBB subtracts the indicated variable and the carry flag together from the Accumulator, leaving the result in the Accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set *before* executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the source operand.) AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

SUBB A,R2

will leave the value 74H (01110100B) in the Accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9H minus 54H is 75H The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR C instruction

SUBB A,Rn

Bytes: 1

Cycles: 1

Encoding:

1	0	0	1	1	r	r	r

Operation: SUBB
(A) ← (A) – (C) – (R_n)

SUBB A,direct

Bytes: 2

Cycles: 1

Encoding:

1	0	0	1	0	1	0	1	direct address

Operation: SUBB
(A) ← (A) – (C) – (direct)

SUBB A,@RI

Bytes: 1

Cycles: 1

Encoding:

1	0	0	1	0	1	1	i

Operation: SUBB
(A) ← (A) – (C) – (R_i)

SUBB A,#data

Bytes: 2

Cycles: 1

Encoding:

1	0	0	1	0	1	0	0	immediate data

Operation: SUBB
(A) ← (A) – (C) – (#data)

SWAP A

Function: Swap nibbles within the Accumulator

Description: SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the Accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

SWAP A

leaves the Accumulator holding the value 5CH (01011100B).

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	0	1	0	0

Operation: SWAP
$(A_{3-0}) \rightleftarrows (A_{7-4})$

XCH A,<byte>

Function: Exchange Accumulator with byte variable

Description: XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

Example: R0 contains the address 20H. The Accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCH A,@R0

will leave the RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the Accumulator.

XCH A,Rn

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	1	r	r	r

Operation: XCH
$(A) \rightleftarrows (R_n)$

XCH A,direct

Bytes: 2

Cycles: 1

Encoding:

1	1	0	0	0	1	0	1	direct address

Operation: XCH
$(A) \rightleftarrows (direct)$

XCH A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	0	1	1	i

Operation: XCH
$(A) \rightleftarrows ((R_i))$

XCHD A,@RI

Function: Exchange Digit

Description: XCHD exchanges the low-order nibble of the Accumulator (bits 3-0), generally representing a hexadecimal or BCD digit, with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.

Example: R0 contains the address 20H. The Accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCHD A,@R0

will leave RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the Accumulator.

Bytes: 1

Cycles: 1

Encoding:

1	1	0	1	0	1	1	1

Operation: XCHD

$(A_{3-0}) \rightleftarrows ((RI_{3-0}))$

XRL <dest-byte>,<src-byte>

Function: Logical Exclusive-OR for byte variables

Description: XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

(*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.)

Example: If the Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

XRL A,R0

will leave the Accumulator holding the value 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combinations of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the Accumulator at run-time. The instruction,

XRL P1,#00110001B

will complement bits 5, 4, and 0 of output Port 1.

XRL A,Rn

Bytes: 1

Cycles: 1

Encoding: 0 1 1 0 | 1 r r r

Operation: XRL
(A) ← (A) ⩛ (R_n)

XRL A,direct

Bytes: 2

Cycles: 1

Encoding: 0 1 1 0 | 0 1 0 1 | direct address

Operation: XRL
(A) ← (A) ⩛ (direct)

XRL A,@RI

Bytes: 1

Cycles: 1

Encoding: 0 1 1 0 | 0 1 1 i

Operation: XRL
(A) ← (A) ⩛ (R_i)

XRL A,#data

Bytes: 2

Cycles: 1

Encoding: 0 1 1 0 | 0 1 0 0 | immediate data

Operation: XRL
(A) ← (A) ⩛ #data

XRL direct,A

Bytes: 2

Cycles: 1

Encoding: 0 1 1 0 | 0 0 1 0 | direct address

Operation: XRL
(direct) ← (direct) ⩛ (A)

XRL direct,#data

Bytes: 3

Cycles: 2

Encoding: 0 1 1 0 | 0 0 1 1 | direct address | immediate data

Operation: XRL
(direct) ← (direct) ⩛ #data

APPENDIX - B

ASCII TABLE

ASCII TABLE

Dec	Hex	Sym	Dec	Hex	Sym	Dec	Hex	Sym	Dec	Hex	Sym	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
0	00	NUL	32	20	SP	64	40	@	96	60	`	128	80	160	A0	192	C0	224	E0
1	01	SOH	33	21	!	65	41	A	97	61	a	129	81	161	A1	193	C1	225	E1
2	02	STX	34	22	"	66	42	B	98	62	b	130	82	162	A2	194	C2	226	E2
3	03	ETX	35	23	#	67	43	C	99	63	c	131	83	163	A3	195	C3	227	E3
4	04	EOT	36	24	$	68	44	D	100	64	d	132	84	164	A4	196	C4	228	E4
5	05	ENQ	37	25	%	69	45	E	101	65	e	133	85	165	A5	197	C5	229	E5
6	06	ACK	38	26	&	70	46	F	102	66	f	134	86	166	A6	198	C6	230	E6
7	07	BEL	39	27	'	71	47	G	103	67	g	135	87	167	A7	199	C7	231	E7
8	08	BS	40	28	(	72	48	H	104	68	h	136	88	168	A8	200	C8	232	E8
9	09	HT	41	29	)	73	49	I	105	69	i	137	89	169	A9	201	C9	233	E9
10	0A	LF	42	2A	*	74	4A	J	106	6A	j	138	8A	170	AA	202	CA	234	EA
11	0B	VT	43	2B	+	75	4B	K	107	6B	k	139	8B	171	AB	203	CB	235	EB
12	0C	FP	44	2C	‘	76	4C	L	108	6C	l	140	8C	172	AC	204	CC	236	EC
13	0D	CR	45	2D	-	77	4D	M	109	6D	m	141	8D	173	AD	205	CD	237	ED
14	0E	SO	46	2E	.	78	4E	N	110	6E	n	142	8E	174	AE	206	CE	238	EE
15	0F	SI	47	2F	/	79	4F	O	111	6F	o	143	8F	175	AF	207	CF	239	EF
16	10	DLE	48	30	0	80	50	P	112	70	p	144	90	176	B0	208	D0	240	F0
17	11	DC1	49	31	1	81	51	Q	113	71	q	145	91	177	B1	209	D1	241	F1
18	12	DC2	50	32	2	82	52	R	114	72	r	146	92	178	B2	210	D2	242	F2
19	13	DC3	51	33	3	83	53	S	115	73	s	147	93	179	B3	211	D3	243	F3
20	14	DC4	52	34	4	84	54	T	116	74	t	148	94	180	B4	212	D4	244	F4
21	15	NAK	53	35	5	85	55	U	117	75	u	149	95	181	B5	213	D5	245	F5
22	16	SYN	54	36	6	86	56	V	118	76	v	150	96	182	B6	214	D6	246	F6
23	17	ETB	55	37	7	87	57	W	119	77	w	151	97	183	B7	215	D7	247	F7
24	18	CAN	56	38	8	88	58	X	120	78	x	152	98	184	B8	216	D8	248	F8
25	19	EM	57	39	9	89	59	Y	121	79	y	153	99	185	B9	217	D9	249	F9
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z	154	9A	186	BA	218	DA	250	FA
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{	155	9B	187	BB	219	DB	251	FB
28	1C	FS	60	3C	<	92	5C	\	124	7C	|	156	9C	188	BC	220	DC	252	FC
29	1D	GS	61	3D	=	93	5D	]	125	7D	}	157	9D	189	BD	221	DD	253	FD
30	1E	RS	62	3E	>	94	5E	^	126	7E	~	158	9E	190	BE	222	DE	254	FE
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL	159	9F	191	BF	223	DF	255	FF

ASCII SYMBOLS

NUL	-	NULL
SOH	-	START OF HEADING
SIX	-	START OF TEXT
ETX	-	END OF TEXT
EOT	-	END OF TRANSMISSION
ELQ	-	ENQUIRY
ACK	-	ACKNOWLEDGE
BEL	-	BELL
BS	-	BACKSPACE
HT	-	HORIZONTAL TABULATION
LF	-	LINE FEED
VT	-	VERTICAL TABULATION
FF	-	FORM FEED
CR	-	CARRAIGE RETURN
SO	-	SHIFT OUT
SI	-	SHIFT IN
DLE	-	DATA LINK ESCAPE
DC	-	DEVICE CONTROL
NAK	-	NEGATIVE ACKNOWLEDGE
SYN	-	SYNCHRONOUS IDLE
ETB	-	END OF TRANSMISSION BLOCK
CAN	-	CANCEL
EM	-	END OF MEDIUM
SUB	-	SUBSTITUTE
ESC	-	ESCAPTE
FS	-	FILE SEPERATOR
GS	-	GROUP SEPERATOR
RS	-	RECORD SEPERATOR
US	-	UNIT SEPERATOR
SP	-	SPACE (BLANK)
DEL	-	DELETE

APPENDIX - C

HEX CONVERSION TABLE

EX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	00	000
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	256	4096
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	512	8192
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	768	12288
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	1024	16384
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	1280	20480
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	1536	24576
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	1792	28672
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	2048	32768
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	2304	36864
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	2560	40960
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	2816	45056
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	3072	49152
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	3328	53248
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	3584	57344
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	3840	61440

HEXADECIMAL CONVERSION TABLE

5		4		3		2		1		0	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3840	F	240	F	15

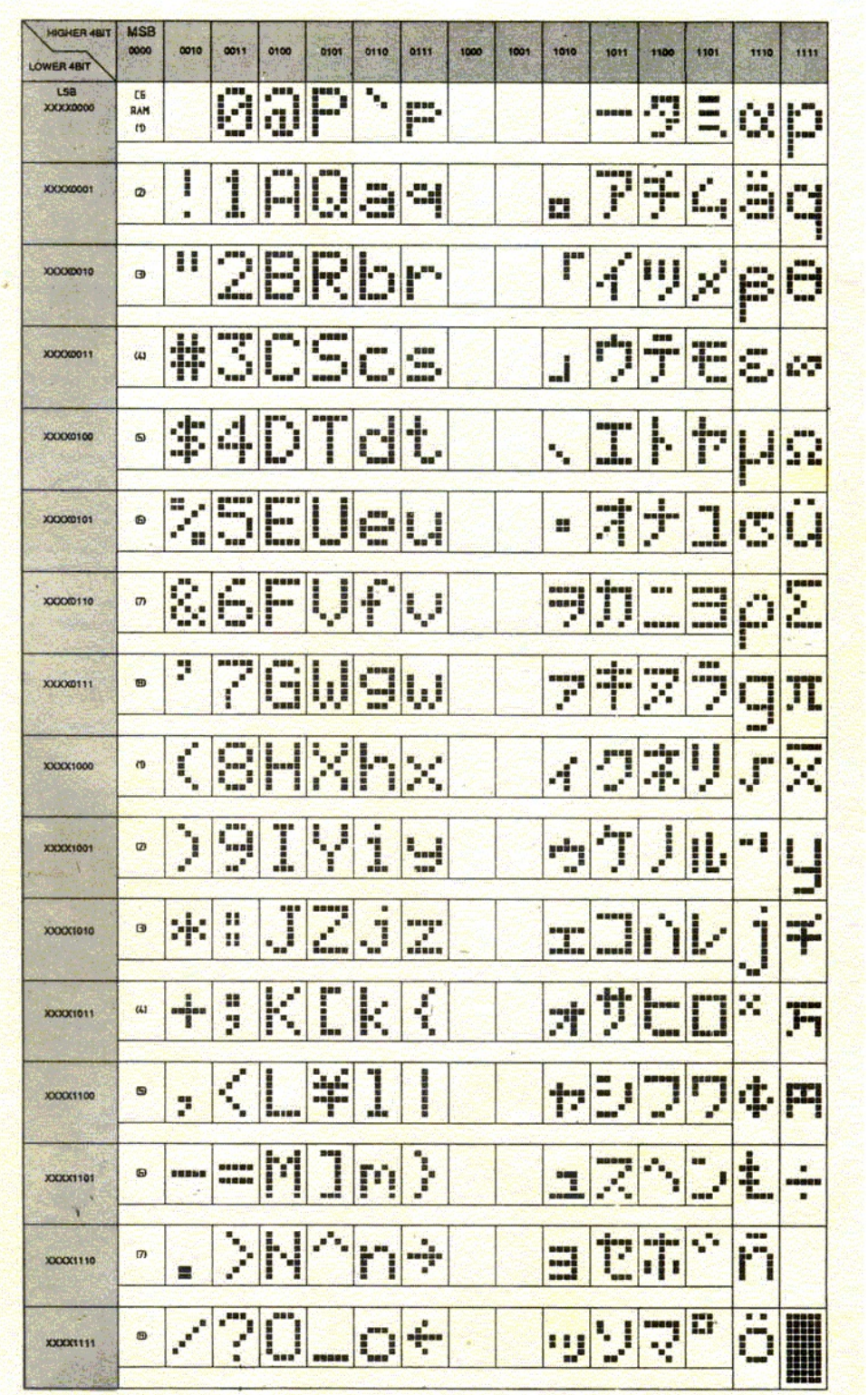
APPENDIX - D

SYSTEM CALLS QUICK REFERENCE GUIDE

A	R1	DPTR	FUNCTION
00	-	-	System Reset.
01	-	-	Blank / Prompt Display.
01	00	-	Blank Address Display.
01	01	-	Blank Date Display.
01	02	-	Prompt Address Display.
01	03	-	Prompt Date Display.
01	04	-	Clears the Entire Display.
02	-	-	Display Data using register.
02	00	-	Contents of address held by DPTR is displaced in field whose row is given in register B.
02	01	-	Contents of DPTR is displayed in Address field.
02	02	-	Contents of "DPL" is displayed in Data field
03	-	-	Read Data from Keyboard and display it.
03	00	-	Read Address from Keyboard and display it in address field.
03	01	-	Read Data from Keyboard and display it in data field.
04	-	-	Display "Err."
05	-	-	Display data using Pointer
05	00	Pointer	Display the contents of address held by DPTR
0B	-	Counter	Delay is generated by decrementing contents of DPTR till it becomes zero (A=0A).
0C	-	Pointer	Load DPH with the contents of address held by DPTR and DPL with the contents of next location (A=0B).

APPENDIX - E

LCD CHARACTER FONT TABLE


HIGHER 4BIT
LOWER 4BIT
MSB
0000
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
LSB
XXXX0000
CG RAM (1)
XXXX0001 (2)
XXXX0010 (3)
XXXX0011 (4)
XXXX0100 (5)
XXXX0101 (6)
XXXX0110 (7)
XXXX0111 (8)
XXXX1000 (1)
XXXX1001 (2)
XXXX1010 (3)
XXXX1011 (4)
XXXX1100 (5)
XXXX1101 (6)
XXXX1110 (7)
XXXX1111 (8)