| January - 2018<br>Microcontroller<br>[CBCS Scheme] | Solved Paper<br>Sem - V (EEE) |
|---|---|

Time : Three Hours] [Maximum Marks : 80

**Note :** ***Answer any FIVE full questions, choosing one full question from each module.***

## MODULE - 1

**Q.1 a)** *Discuss the need for stack memory in microcontroller. How stack is operated in 8051 μc ? What is the default location of stack ?* **(Refer section 1.5.6)** **[6]**

**b)** *With an example explain the various addressing modes used in 8051 μc (any four)* **(Refer section 2.1)** **[6]**

**c)** *Compare RISC and CISC micro controllers.* **(Refer section 1.2.2.3)** **[4]**

**OR**

**Q.2 a)** *Explain the bit pattern of program status word.* **(Refer section 1.5.4)** **[6]**

**b)** *With a neat diagram, explain the steps to interface 8 k byte of program RAM and 6 k bytes of data ROM to 8031 based system.* **(Refer section 1.8)** **[6]**

**c)** *Identify the addressing modes of the following instruction.*

*i) MOVCA, @A + DPTR* **(Index addressing)**

*ii) MOV DPTR # 1234* **{Immediate addressing}**

*iii) MOV A,4* **(Direct addressing)**

*iv) CLRC* **(Bit inherent addressing)** **[4]**

## MODULE - 2

**Q.3 a)** *Write a program to find the square root of a given number.* **(Refer example 3.7.28)** **[6]**

**b)** *With a neat diagram explain the range of JUMP and CALL instructions.* **(Refer section 2.9.1)** **[8]**

**c)** *Explain the following intersection : i) DAA, ii) ANLC, P2.5* **(Refer sections 2.6.5 and 2.7)** **[2]**

**OR**

**Q.4 a)** *What are assembler directives ? Explain any four of them with an example.* **(Refer section 3.6)** **[6]**

**b)** *Assume that register 'A' is loaded with number 'N' (any integer value from 0 to 255). Write a program to count the number of ones in even numbered bits of accumulator.* **(Refer similar example 3.7.21)** **[5]**

**c)** *Write a program to complement the content of accumulator 62500 times.* **(Refer similar example 3.7.60)** **[5]**

## MODULE - 3

**Q.5 a)** *Explain the different data types supported by 80561C microcontroller.* **(Refer section 4.2)** **[8]**

**b)** *Write a program to create a square wave of 100 Hz with a duty cycle of 80 % on port 1.1. Use timer '0' and operate that timer '0' in mode '1'. Assume XTAL* $f_{mov} = 12\,MHz$. **(Refer similar example 5.3.11)** **[8]**

**OR**

**Q.6 a)** *A switch is connected to pin P1.2. Write an 8051 C program to monitor 'SW' and create the following frequencies on pin P1.7.*
*SW = 0 : 500 Hz*
*SW = 1 : 750 Hz*

*Use timer '0', mode' 1' for both of them. Assume crystal frequency = 11.0592 MHz.* **(Refer example 5.3.6)** **[8]**

**b)** *Write an 8051 C program to turn bit P1.5 ON and OFF 50000 times.* **(Refer similar example 4.4.9)** **[3]**

**c)** *Write a program for counter '1' in mode '2' to count the clock pulse and display the state of the TL, count on P2.* **(Refer similar example 5.2.8)** **[5]**

## MODULE - 4

**Q.7 a)** *Write a program to retrieve the data serially and put them in P1. Set the baud rate at 4800, 8-bit data and one stop bit.* **(Refer example 6.5.11)** **[6]**

**b)** *Write an 8051 C program to transfer the message :"INDIA" serially at 9600 baud rate, 8 bit data and one stop bit, continuously.* **(Refer similar example 6.6.2)** **[6]**

**c)** *Explain the importance of TI and RI flags.* **(Refer sections 6.5.1 and 6.5.2)** **[4]**

**OR**

**Q.8 a)** *What is an interrupt ? List the various interrupts of the 8051 with their corresponding vector address.* **(Refer sections 7.1 and 7.2)** **[6]**

**b)** *Write a program that continuously gets 8-bit data from 'P0' and sends it to 'P1' where simultaneously creating a square wave of 200 μs period on pin P2.1. Use timer '0' to create square wave. Assume XTAL = 11.0952 MHz.*
**(Refer similar example 7.4.1)** **[7]**

**c)** *Explain simplex, half duplex and full duplex serial data transfer.*
**(Refer section 6.2.1)** **[3]**

## MODULE - 5

**Q.9 a)** *A switch is connected to pin P2.7. Write a 'C' program to monitor the status of 'SW' and perform the following :*
*i) If SW = 0 : the stepper motor moves clock wise.*
*ii) If SW = 1 : the stepper motor moves counter clock wise.*
**(Refer similar example 8.11.4)** **[10]**

**b)** *Explain the control word format of 8255.* **(Refer section 9.2.2)** **[6]**

**OR**

**Q.10 a)** *Explain the various modes of 8255 and find the control word for following configurations :*

*i) All ports of A, B and C are O/P parts (mode '0')*

*ii) PA = IN, PB = OUT, PCL = OUT and PCH = OUT.*
**(Refer section 9.2.1.2 and example 9.2.1)** **[8]**

**b)** *Explain the steps to interface ADC 0808 to the 8051 microcontroller.*
**(Refer section 8.5.1)** **[8]**

## 1.5.6 Stack and Stack Pointer

**VTU : Jan.-18, Marks 6**

- The stack refers to an area of internal RAM that is used to store and retrieve data quickly. The stack pointer register is used by the 8051 to hold an internal RAM address that is called **top of stack.** The stack pointer register is 8-bit wide. It is increased **before** data is stored during PUSH and CALL instructions and decremented **after** data is restored during POP and RET instructions.
- The stack array can reside anywhere in on-chip RAM. The stack pointer is initialized to 07H after a reset. This causes the stack to begin at location 08H. We can modify default location of stack by loading new location in stack pointer. For example,

  MOV SP, # 13H
- The operation of stack and stack pointer is illustrated in Fig. 1.5.5.

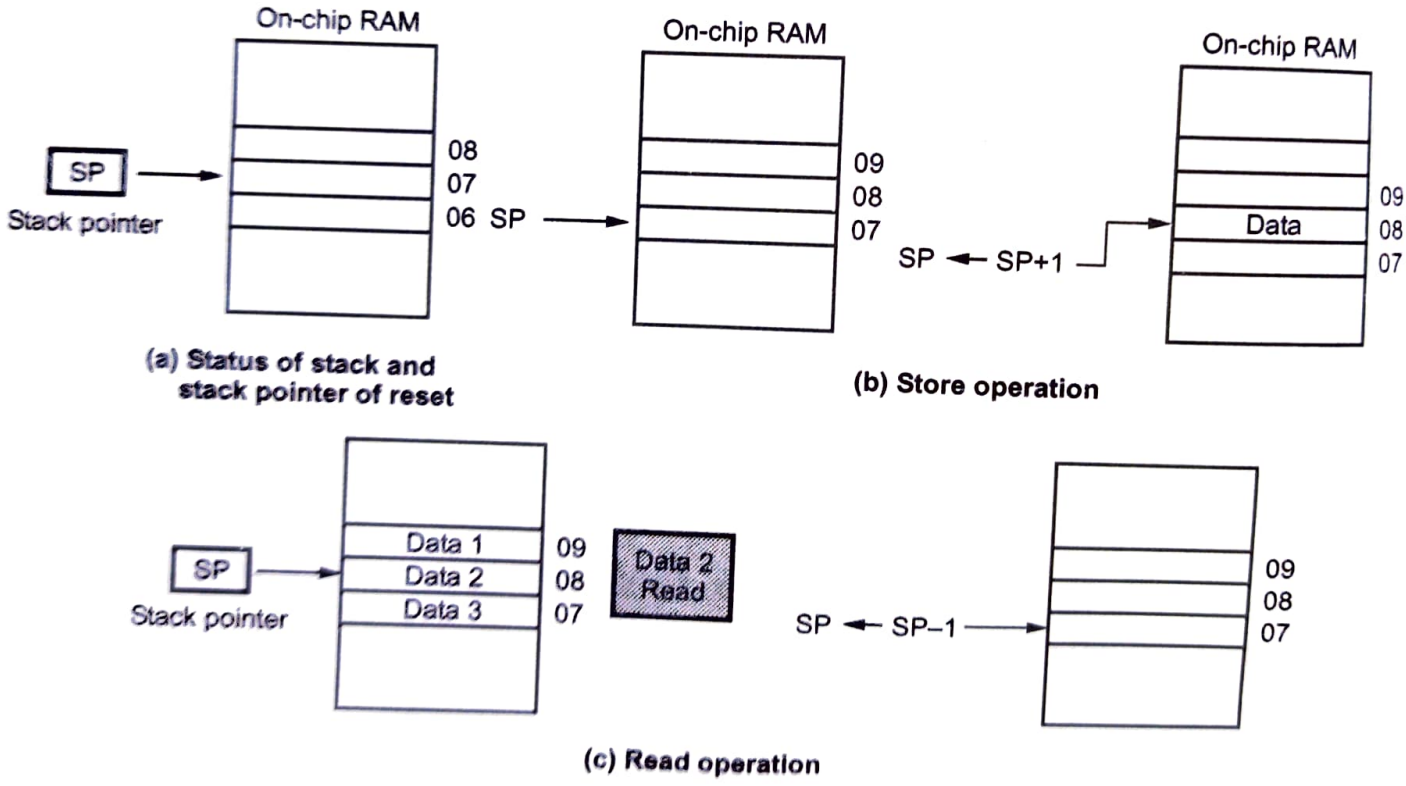



**Fig. 1.5.5**

- The stack may overwrite data in the register banks, bit-addressable RAM and scratch-pad RAM. Thus to avoid conflict with the register, bit-addressable RAM and scratch-pad RAM data, the stack is initialized at a higher location in the internal RAM.

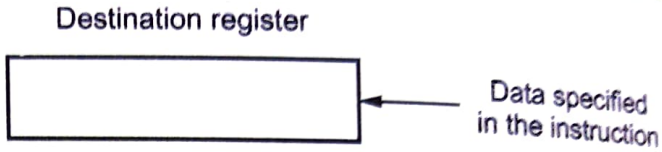
## 2.1 Addressing Modes

**VTU : Feb.-06, 08, 10, 11, 16, 17, Aug.-06, 07, 09, 11, 16, Dec.-11, 12, June-12, 13, 14, 15, Jan.-14, 18, July-18, 19**

- The way, using which the data sources or destination addresses are specified in the instruction mnemonic for moving the data, is called '**addressing mode**'. This section explains addressing modes used in 8051 with examples.

### 2.1.1 Immediate Addressing

- In this addressing mode source operand is a constant rather than a variable. So the constant can be incorporated in the instruction. Sign "#" indicates it is a immediate addressing mode.




**Example :** Add the constant 52 decimal in accumulator.

```
MOV A, #52
```

### 2.1.2 Register Addressing

- The 8051 can access eight "working registers" (R0-R7). Three bit code within the instruction selects one of the eight registers from the selected register bank. In the register addressing mode, the data to be operated is within the registers.

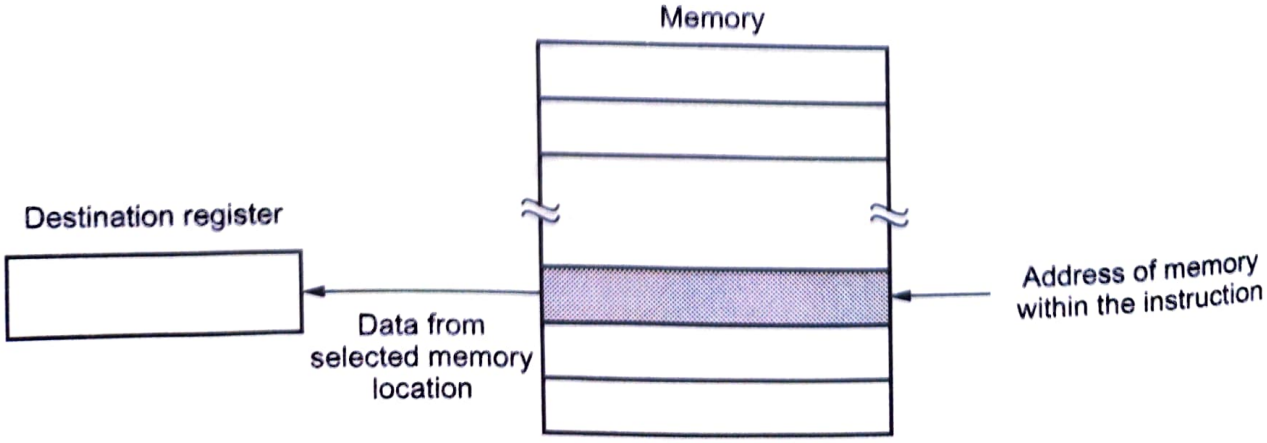
**Example :**

```
MOV A, R3     ;    Copies content of R3 to A
ADD A, R4     ;    A ← A + R4
```

**Note :** The advantage of register addressing is that the instruction length is small; it is a single byte.

### 2.1.3 Direct Addressing

- Direct addressing can access any on-chip variable or hardware register. i.e. on-chip RAM and special function register.



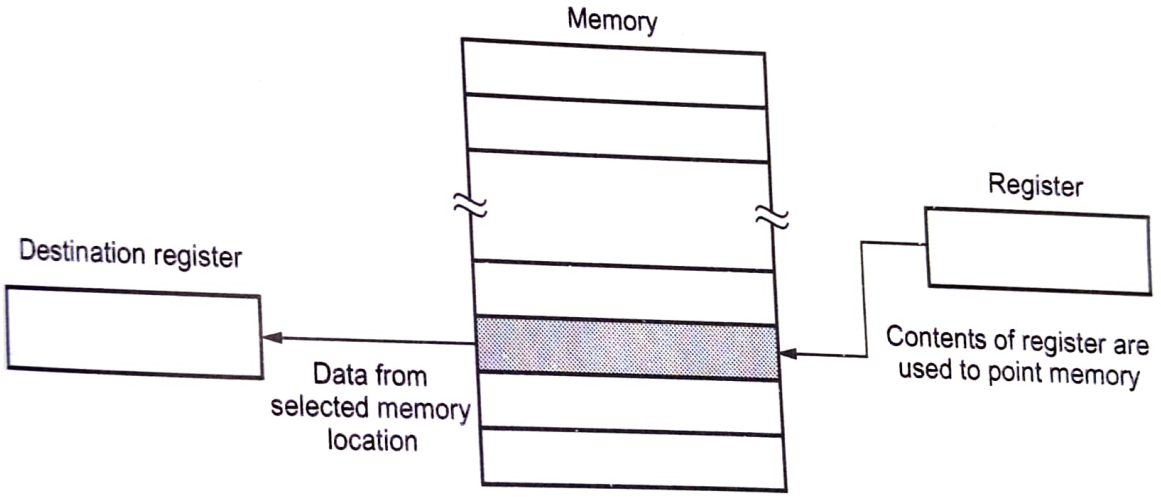
- The most significant bit of the address decides whether it is a location within on-chip RAM (MSB = 0) or in special function register (MSB = 1).

**Example :** Add the contents of locations 50H and 51H

```
MOV  A, 50H      ; Load byte from address 50H into A
ADD  A, 51H      ; Add the contents of A and the contents at memory
                 ; location 51H.
```

### 2.1.4 Indirect or Register Indirect Addressing

- In this addressing mode R0 and R1 of each register bank can be used as an index or pointer register. R0 and R1 point to the contents in the RAM. The instruction with indirect addressing uses the '@' sign.




- Indirect addressing accesses data in dynamic manner rather than static manner. Looping is not possible in direct addressing mode. In indirect addressing we can increment the index or pointer register to access successive locations.
- R0, R1 and DPTR are the only registers that can be used for pointers in register indirect addressing mode.

**Example :** ADD the contents of memory location addressed by register 1 to the contents of RAM location pointed by register 0.

```
MOV A, @R0        ; Load the contents pointed by R0 in A
ADD A, @R1        ; Add the contents of A and the contents pointed by R1
MOVX A, @ DPTR    ; Copy the contents of external data memory
                  ; pointed by DPTR into the accumulator
```

### 2.1.5 Relative Addressing

- The JMP (JUMP) and CALL instructions in 8051 use the relative addressing.

### 1.2.2.3 CISC vs RISC Microcontrollers

- CISC is an acronym for Complex Instruction Set computers or computing. It is based on the concept of using very large instruction set having simple as well as complex instructions and making instruction set more flexible to keep program length as small as possible.
- RISC refers to Reduced Instruction Set Computers or Computing. RISC microcontrollers are very different from CISC microcontrollers. RISC use concept of keeping the instruction set as simple as possible to allow the microcontroller's program to be written using only simple instructions.

| Sr. No. | RISC | CISC |
|---|---|---|
| 1 | Instruction takes one or two cycles | Instruction takes multiple cycles |
| 2 | Only load/store instruction are used to access memory | In addition to load and store instructions, memory access in possible with other instructions also |
| 3 | Instruction executed by hardware | Instruction executed by the micro program |
| 4 | Fixed format instructions | Variable format instructions |
| 5 | Few addressing modes | Many addressing modes |
| 6 | Few instructions | Complex instruction set |
| 7 | Most of them have multiple register banks | Single register bank |

### 1.2.3 Criteria for Selecting Microcontroller

Criteria that designer should consider in choosing microcontrollers are :

- It should satisfy the computing needs of the task efficiently and cost effectively. It should also satisfy the speed requirements, packaging format, RAM and ROM capacity, number I/O pins, on-chip timers and power consumption needs of the application.
- Availability of software development tools such as compilers, assemblers and debuggers.

## 1.5.3 Program Counter

- The 8051 has a 16-bit program counter. It is used to hold the address of memory location from which the next instruction is to be fetched.

## 1.5.4 8051 Flag Bits/PSW Registers

- The Fig. 1.5.3 shows the bit pattern of Program Status Word (PSW) of 8051. PSW is also known as **flag register**.

| $B_7$ | $B_6$ | $B_5$ | $B_4$ | $B_3$ | $B_2$ | $B_1$ | $B_0$ |
|---|---|---|---|---|---|---|---|
| CY | AC | FO | RS1 | RS0 | OV | - | P |

**Fig. 1.5.3**

- The 8051 consists of following flags.
- **CY-Carry Flag :** This flag is set if there is an overflow out of bit 7. The carry flag also serves as a borrow flag for subtraction. In both the examples shown below, the carry flag is set.
- **AC-Auxiliary Carry Flag :** This flag is set if there is an overflow out of bit 3 i.e., carry from lower nibble to higher nibble ($D_3$ bit to $D_4$ bit).

**ADDITION**

```
          9B H        1001 1011
        + 75 H      + 0111 0101
Carry [1] 10 H      [1] 0001 0000
```

**SUBTRACTION**

```
          89 H        1000 1001
        - AB H      - 1010 1011
Borrow [1] DE H     [1] 1101 1110
```

- **FO** - Available for user for general purpose.
- **RS1 - RS0 (Register Bank Select) :** They select the working register bank as follows :

| RS1 | RS0 | Bank Selection | |
|---|---|---|---|
| 0 | 0 | 00H - 07H | Bank 0 |
| 0 | 1 | 08H - 0FH | Bank 1 |
| 1 | 0 | 10H - 17H | Bank 2 |
| 1 | 1 | 18H - 1FH | Bank 3 |

- **OV-Over Flow Flag :** This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit.
- **P-Parity Flag :** Parity is defined by the number of ones present in the accumulator. P = 0, if number of ones are even and P = 1, if number of ones are odd.

1. *Draw and explain the memory structure of 8051.* **VTU : Feb.-06, 09, Marks 10**
2. *Explain the internal RAM organization of 8051 microcontroller.*
**VTU : Feb.-07, 11, Aug.-08, Dec.-12, June-14, July-18, Marks 6**
3. *Comment on ROM space in the 8051.*
4. *With neat diagram, with the programming model of 8051 with addresses of SFR's and ports. Also give 128 bytes RAM allocation.* **VTU : Jan.-16, Marks 12**
5. *Draw the programming model of 8051 µc. Explain the function of following :*
*i) Accumulator, register B and CPU registers ii) Program controller stack and stack pointer*
**VTU : Jan.-19, Marks 10**

## 1.8 External Memory Interfacing

**VTU : Dec.-11, June-13, 16, 18, July-18, Jan.-16, 18, 19**

- We have seen that 8051 has internal data and code memory with limited memory capacity. This memory capacity may not be sufficient for some applications. In such situations, we have to connect external ROM/EPROM and RAM to 8051 microcontroller to increase the memory capacity. We also know that ROM is used as a program memory and RAM is used as a data memory. Let us see how 8051 accesses these memories.

### 1.8.1 External Program Memory

- Fig. 1.8.1 shows a map of the 8051 program memory.

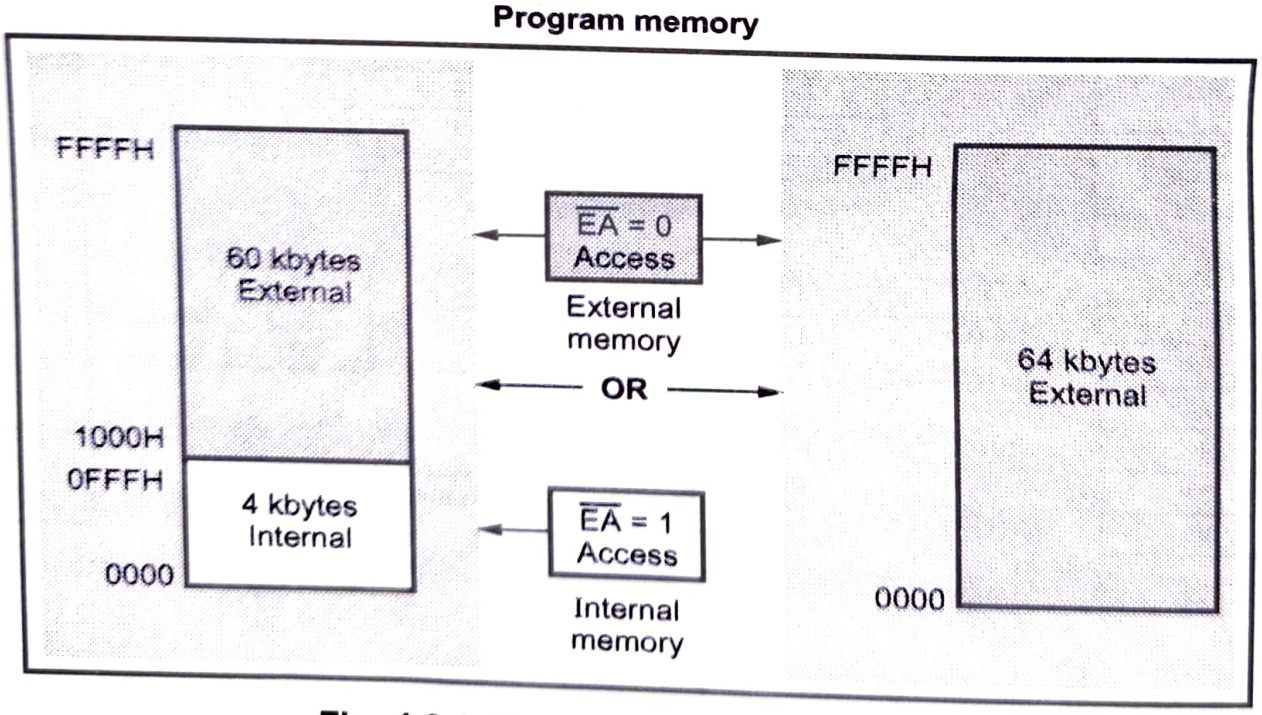



**Fig. 1.8.1 The 8051 program memory**

- In 8051, when the $\overline{EA}$ pin is connected to $V_{CC}$, program fetches to addresses 0000H through 0FFFH are directed to the internal ROM and program fetches to addresses 1000H through FFFFH are directed to external ROM/EPROM. On the other hand when $\overline{EA}$ pin is grounded, all addresses (0000H to FFFFH) fetched by program are directed to the external ROM/EPROM. The $\overline{PSEN}$ signal is used to activate output enable signal of the external ROM/EPROM, as shown in the Fig. 1.8.2.

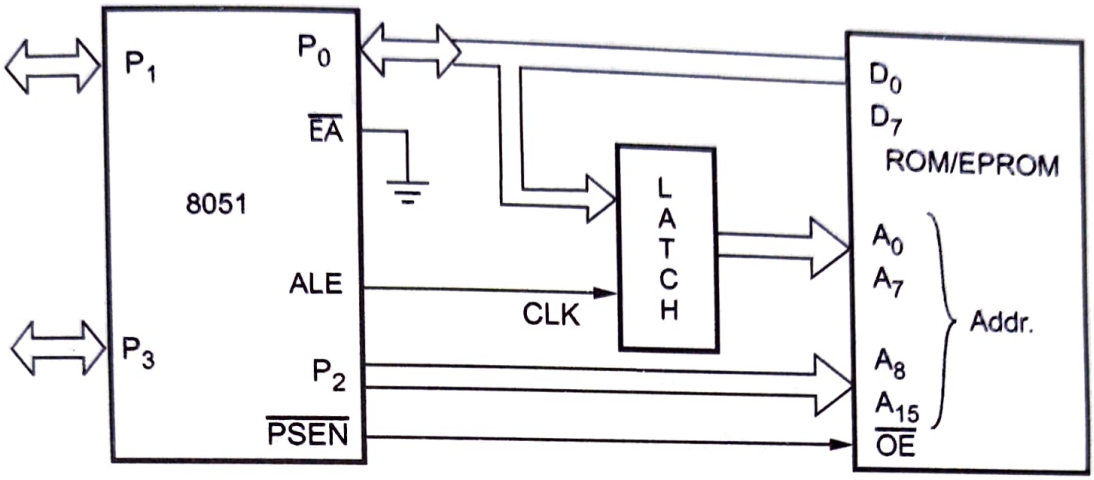



**Fig. 1.8.2 Accessing external program memory**

- As shown in the Fig. 1.8.2, the port 0 is used as a multiplexed address/bus. It gives lower order 8-bit address in the initial T-cycle and later it is used as a data bus. The 8-bit address is latched using external latch and ALE signal generated by 8051.

- The port 2 provides the higher order 8-bit address. Fig. 1.8.3 shows the timing waveforms for external program memory read cycle.

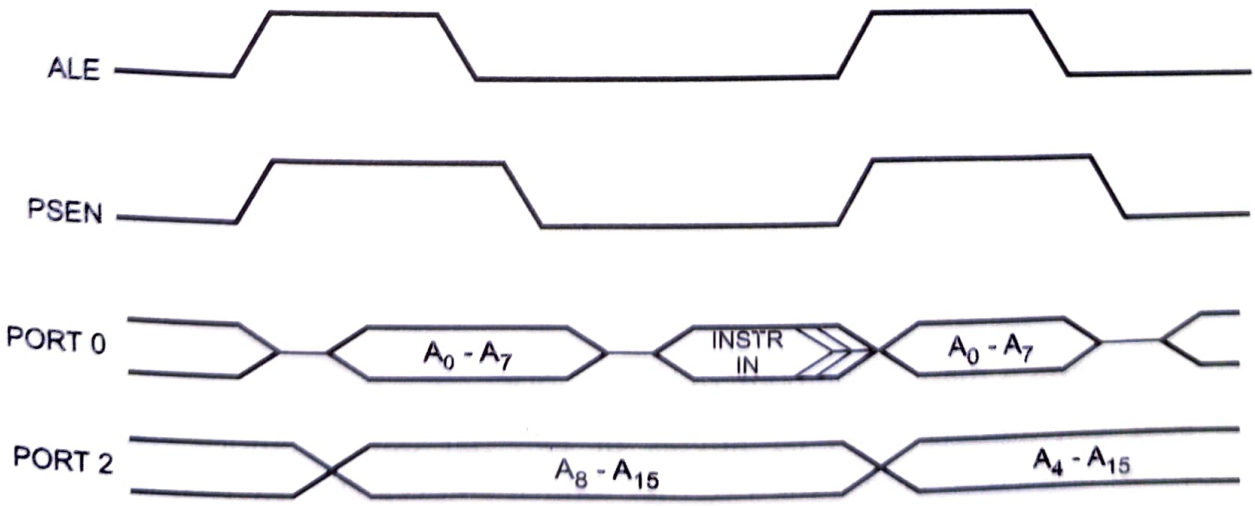



**Fig. 1.8.3 Timing waveforms for external program memory read cycle**

- The lower part of program memory stores the vector addresses for various interrupt service routines. Fig. 1.8.4 shows the vector address map. Each interrupt

```
        DJNZ R0, BACK        ;    If count #0, repeat
```

**Example 3.7.37** *Write an assembly language program to realize following logic circuit using Boolean instructions of 8051.*

**Solution :** Let us assume that input a is connected to P1.0, input b is connected by P1.1 and output Y is connected at P2.0.

```
        MOV P1, #0FFH        ;    Configure port1 as input
        MOV A, P1            ;    Read port 1
        JB P1.0 NEXT         ;    If P1.0 is set go to NEXT
        JB P1.1, NEXT1       ;    If P1.1 is set goto NEXT1
        CLR P2.0             ;    If P1.0=P1.1=0 make P2.0=0
        SJMP LAST
NEXT1 :  SETB P2.0           ;    If P1.0 = 0, P1.1=1 make P2.0=1
        SJMP LAST
NEXT :  JB P1.1, NEXT2
        SETB P2.0            ;    If P1.0=1, P1.1=0 make P2.0=1
        SJMP LAST
NEXT2 :  CLR P2.0            ;    If P1.0 = 1, P1.1 = 1
                             ;    make P2.0 = 0
LAST :  RET
```

**Example 3.7.38** *Write an assembly language program to find the square root of a number and store in R0.* **VTU : Feb.-06, Marks 10**

**Solution :** We can find out square root of 8-bit numbers using interactive method. The algorithm shows the method.

**Algorithm :**

1. Start
2. Get the square number
3. Set number = 0
4. Compare square with number × number

5. If equal goto step 8
6. Number = number + 1
7. Goto step 4
8. Store the number
9. Stop

**Program**

```
        MOV R0,#square
        MOV R1,#00
BACK :  MOV A,R1
        MOV B,A
        MUL AB
        CJNE A,R0,NEXT
        SJMP LAST
NEXT :  INC R1
        SJMP BACK
LAST :  MOV R2,A
```

In this program we used R0 register to store the square **term**, and result is in R2 register.

**Example 3.7.39** *Write a subroutine to decrement the content of DPTR.*

**VTU : Feb.-07, Marks 4**

**Solution :**

```
DE:  MOV A, DPL
     CLR C
     SUBB A, #01
     MOV DPL, A
     MOV A, DPH
     SUBB A, #00
     MOV DPH, A
     RET
```

**Example 3.7.40** *Write a program to set the carry flag to 1 if the number in register A is even and set the carry flag to 0, if the number in register A is odd.* **VTU : Aug.-07, Marks 4**

**Solution :**

```
RRC A    ;  MOVE LSB to carry
CPL C    ;  Complement carry
```

**Example**

- Absolute range : 0000H to 07FFH
- Long range : 0000H to FFFFH

## 2.9.2 Jump

- The Table 2.9.1 shows the list of jump instructions supported by 8051.

| | | |
|---|---|---|
| **AJMP addr11 : Absolute Jump** | **Bytes : 2** | **Cycles : 2** |
| Description : | AJMP transfers program execution to the indicated address. Since address is 11-bit the destination must therefore be within the same 2 K block of program memory as the first byte of the instruction following AJMP. No flags are affected. | |
| **LJMP addr16 : Long Jump** | **Bytes : 3** | **Cycles : 2** |
| 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 64 K program memory address space. No flags are affected. | |
| **SJMP rel : Short Jump (relative addr)** | **Bytes : 2** | **Cycles : 2** |
| 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 128 bytes preceding this instruction to 127 bytes following it. | |
| **JMP @A+DPTR : Jump indirect relative to the DPTR** | **Bytes : 1** | **Cycles : 2** |
| 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.<br>Neither the accumulator nor the data pointer is altered. No flags are affected. | |
| **JZ rel : Jump if Accumulator is zero** | **Bytes : 2** | **Cycles : 2** |
| Description : | If all bits of the Accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The accumulator is not modified. No flags are affected. | |
| **JNZ rel : Jump if Accumulator is not zero** | **Bytes : 2** | **Cycles : 2** |
| Description : | If any bit of the Accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The accumulator is not modified. No flags are affected. | |

**JC rel : Jump if Carry flag is set** **Bytes : 2** **Cycles : 2**

**Description :** If the carry flag is set branch to the address indicated; otherwise proceed with the next instruction. No flags are affected.

**JNC rel : Jump if no Carry flag** **Bytes : 2** **Cycles : 2**

**Description :** If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The carry flag is not modified.

**JB bit, rel : Jump if direct Bit set** **Bytes : 3** **Cycles : 2**

**Description :** If the indicated bit is one, jump to the address indicated; otherwise proceed with the next instruction. The bit tested is not modified. No flags are affected.

**JNB bit, rel : Jump if direct Bit not set** **Bytes : 3** **Cycles : 2**

**Description :** If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The bit tested is not modified. No flags are affected.

**JBC bit, rel : Jump if direct Bit is set and Clear bit** **Bytes : 3** **Cycles : 2**

**Description :** If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. The bit will not be cleared if it is already a zero.

**CJNE <dest-byte>,<scr-byte>, rel : Compare and Jump if Not Equal** **Bytes : 3** **Cycles : 2**

**Description :** CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. 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.

**DJNZ <byte>, <rel-addr> : Decrement and Jump if Not Zero** **Bytes : 2/3** **Cycles : 2**

**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.

**NOP** **Function : No Operation** **Bytes : 1** **Cycles : 1**

**Description :** Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

Program execution continues at the address previously pushed : The first byte of the instruction immediately following the call.

| ACALL addr11 : Absolute Call | Bytes : 2 | Cycles : 2 |
|---|---|---|
| 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. Since address is 11-bit the subroutine called must therefore start within the same 2 K block of the program memory as the first byte of the instruction following ACALL. No flags are affected. | |
| **LCALL addr16 : Long Call** | **Bytes : 3** | **Cycles : 2** |
| 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 64 kbyte program memory address space. No flags are affected. | |
| **RET : Return from subroutine** | **Bytes : 1** | **Cycles : 2** |
| 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. No flags are affected. | |

**RETI : Return from Interrupt** **Bytes : 1** **Cycles : 2**

**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 per-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-level or same-level interrupt had been pending interrupt is processed.

**Example 2.9.1** *What is the difference between RET and RETI instruction ? State why we can not use RET instead of RETI instruction as the last instruction in ISR.*

**VTU : Aug.-07, 08, Dec.-11, Marks 4**

**Solution :** 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 an interrupt was still in progress.

**Example 2.9.2** *Sketch the contents of stack memory indicating the position of SP after the execution of the following :*

*i) CALL instruction and ii) RET instruction. Assume initially SP = 20 H.*

```
                    .
                    .   2050H FACT :        .
                    .                       .
                    .                       .
                    .                       .
2000H A             CALL FACT               RET
```

**VTU : Dec.-11, Marks 4**

**Solution : i) After ACALL :** SP = 22H

**ii) After RET :** SP = 20H

**DA A : Decimal-adjust Accumulator for addition** **Bytes : 1** **Cycles : 1**

**Description :** Adjusts the eight-bit value in the Accumulator resulting from the earlier addition of two variables (each in packed-BCD format), to produce packed-BCD result.

If the lower nibble of the accumulator is greater than 9 or AF is set, it corrects the result by adding 06 in the lower nibble. If the upper nibble of the accumulator is greater than 9 or CF is set, it corrects the result by adding 06 in the upper nibble.

**Example :** A = 55H, B = 68H and CF =1. Then instruction sequence

```
ADDC A, R3
DA  A
```

will first perform a standard binary addition, resulting in the value BEH (10111110) in the Accumulator. The carry and auxiliary carry flags will be cleared.

Since lower nibble > 9, Lower nibble = EH + 6H = 4H with AF=1

Since upper nibble > 9, Upper nibble = BH + 6H + AF(1) = 2H with CF=1

Thus the final result = 124, which is valid BCD sum.

**ANL C, <src-bit> : Logical-AND for bit variables** **Bytes : 2** **Cycles : 2**

**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 flags are affected.

Only direct addressing is allowed for the source operand.

| | | | |
|---|---|---|---|
| ANL C, bit | AND direct bit to Carry flag | Byte 2 | Cycle 2 |
| **Example :** | **ANL C, ACC.7 :** And Carry with Accumulator bit 7 | | |
| ANL C, /bit | AND complement of direct bit to Carry | Byte 2 | Cycle 2 |
| **Example :** | **ANL C, /OV :** And with inverse of Overflow flag | | |

**ORL C, <src-bit> : Logical-OR for bit variables** **Bytes : 2** **Cycles : 2**

**Description :** Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise 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.

| | | | |
|---|---|---|---|
| ORL C, bit | OR direct bit to Carry flag | Byte 2 | Cycle 2 |
| **Example :** | **ORL C, ACC.7 :** Or Carry with the ACC.bit7 | | |
| ORL C, /bit | OR complement of direct bit to Carry | Byte 2 | Cycle 2 |
| **Example :** | **ORL C, /OV :** Or Carry with the inverse of OV | | |

**MOV <dest-bit>, <src-bit> : Move bit data** **Bytes : 2** **Cycles : 1/2**

**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.

| | | | |
|---|---|---|---|
| MOV C, bit | Move direct bit to Carry flag | Byte 2 | Cycle 1 |
| **Example :** | **MOV C, P3.3 :** Copies the status of P3.3 into carry. | | |
| MOV bit, C | Move Carry flag to direct bit | Byte 2 | Cycle 2 |
| **Example :** | **MOV P1.3, C :** Copies the status of carry into P1.3. | | |

1. *Explain the steps to be perform to create, assemble and run 8051 assembly language program.*

**VTU : July-19, Marks 8**

2. *Which program produces the "obj" file ?*
3. *How an assembly language program is created, assembled and made ready to run ?*

## 3.6 Assembler Directives

**VTU : June-12, Dec.-12, Jan.-14, 15, 16, 18, 19, July-16, 19**

- There are some instructions in the assembly language program which are not a part of processor instruction set. These instructions are instructions to the assembler, linker, and loader. These are referred to as **pseudo-operations** or as **assembler directives.** The assembler directives enable us to control the way in which a program assembles and lists. They act during the assembly of a program and do not generate any executable machine code.

- Let us study the commonly used assembler directives :

**EQU** : Equate : It is used to define a constant without occupying a memory location. For example, COUNT EQU 10.

**ORG** : Origin : It is used to indicate the beginning of the address. The address comes after the ORG can be specified either in hex or decimal. To specify address in hex, the address should be followed by H. For example,

ORG 0100H

**DB** : Define byte : It is used to define the 8-bit data. The data value comes after the DB can be specified in decimal, binary, hex or ASCII formats. For decimal, the 'D' after the decimal number, for binary 'B' after the binary number and for hex 'H' after the hex number should be specified. For decimal, the 'D' after the decimal number is optional. To indicate ASCII, the characters are placed in quotation marks. For example,

```
NUM 1    DB    32          ;    Num 1 equal to 32 in decimal
NUM 2    DB    32 H        ;    Num 2 equal to 32 in hex, i.e. 50 in decimal
NUM 3    B     10010001B   ;    Num 3 equal to 10010001 in binary
CHAR     DB    "A"         ;    CHAR equal to ASCII A.
```

**END** : The END directive is put after the last statement of a 8051 program to tell the assembler that this is the end of the program.

**CODE** : It assigns a name to the specified memory location in the program memory (Range 0 - 65535). **For example,**

```
LIST CODE 1020H    ;    Memory location 1020H in the program memory is now
                   ;    referred to as LIST
```

**DATA** : It assigns a name to the specified location in the internal RAM of 8051. (Range 0 - 255). **For example,**

```
TEMP DATA 52 H     ;    Register at address 52H is now named as TEMP
```

**IDATA** : It assigns a name to the memory location whose address is located in the specified register. **For example,**

```
MARKS DATA 80      ;    Register whose address is in register at
                   ;    address 80 is named as MARKS
```

**XDATA** : It assigns name to the specified memory location in the external RAM memory (Range 0 - 65535). **For example,**

```
RESULT XDATA 1000H ;    Memory location 1000H in the external RAM
                   ;    memory is now referred to as RESULT
```

**USING** : This directive is used to define which register bank (Bank0 - Bank3) will be used in the following program. **For example,**

```
USING 1            ;    Bank 1 will be used.
```

## Review Questions

1. *What are assembler directives ? Explain any four of them.*

   **VTU : Dec.-12, Jan.-16, 18, Marks 5**

2. *Describe DB, ORG, EQU, END directives in 8051, assembly.* **VTU : Jan.-15, Marks 4**

3. *Explain the function of following directives.*

   *i) DB ii) EQU iii) ORG iv) DATA v) END vi) CODE*

   **VTU : Dec.-12, Jan.-19, Marks 6**

4. *Briefly explain the different assembler directives used in an assembly language program.*

   **VTU : June-12, Marks 4**

5. *What are assembler directives? Explain the functions of the assembler directives DB, EQU, END, ORG.*

   **VTU : July-16, Marks 6**

6. *What do you understand by assembler directives ? Explain the following assembler directives :*

   *i) ORG ii) END iii) EQU*

   **VTU : July-19, Marks 8**

7. *List out and explain different assembler directives used in an ALP.* **VTU : June-14, Marks 6**

## Assembly Language Programs

**Aug.-07, 08, 09, 10, 11, Dec.-11, 12, 18, 19 July-16, 18, 19**

**Example 3.7.20** *Write an ALP (assembly language program) in 8051 to count the number of positive and negative numbers present in the internal memeory block starting with the address 20H, containing N bytes. Store the counts after the last data byte in the memory block.* **VTU : June-12, Marks 6**

**Solution :**

**Program :**

```
        MOV R2, #00         ; Initialize counter = 0 for positive numbers
        MOV R3, #00         ; Initialize counter = 0 for negative numbers
        MOV R1, #N          ; Initialize counter = N
        MOV R0, #20H        ; Initialize internal memory pointer
BACK :  MOV A, @R0          ; Read the number
        ANL A, #80H         ; Mask lower 7 bits
        JZ SKIP             ; If Z = 0 (number is positive) go to SKIP
        INC R3              ; Otherwise increment negative number count
        SJMP LAST           ; Go to LAST
SKIP :  INC R2              ; Increment positive number count
        INC R0              ; Increment memory pointer
        DJNZ R1, BACK       ; Decrement R1 if not zero go to BACK
        MOV A, R2           ; Save positive number count
        MOV @R0, A
        INC R0              ; Increment memory pointer
        MOV A, R3           ; Save negative number count
        MOV @R0, A
```

**Example 3.7.21** *Count number of one's in a number.*

**Solution : Flowchart :** Refer Fig. 3.7.3.

**Program :**

```
        MOV R2, #0          ; Initialize one's counter = 0
        MOV R1, #08         ; Initialize iteration count
        MOV R0, #56         ; Load number
        MOV A, R0           ; Get the number in accumulator
```

```
BACK :   RRC A            ; Rotate A and CY ← LSB
         JNC SKIP         ; If carry is not zero go to skip
         INC R2           ; Otherwise increment one's counter
SKIP :   DJNZ R1, BACK    ; Decrement iteration count and if not zero repeat
```

**Flow chart**

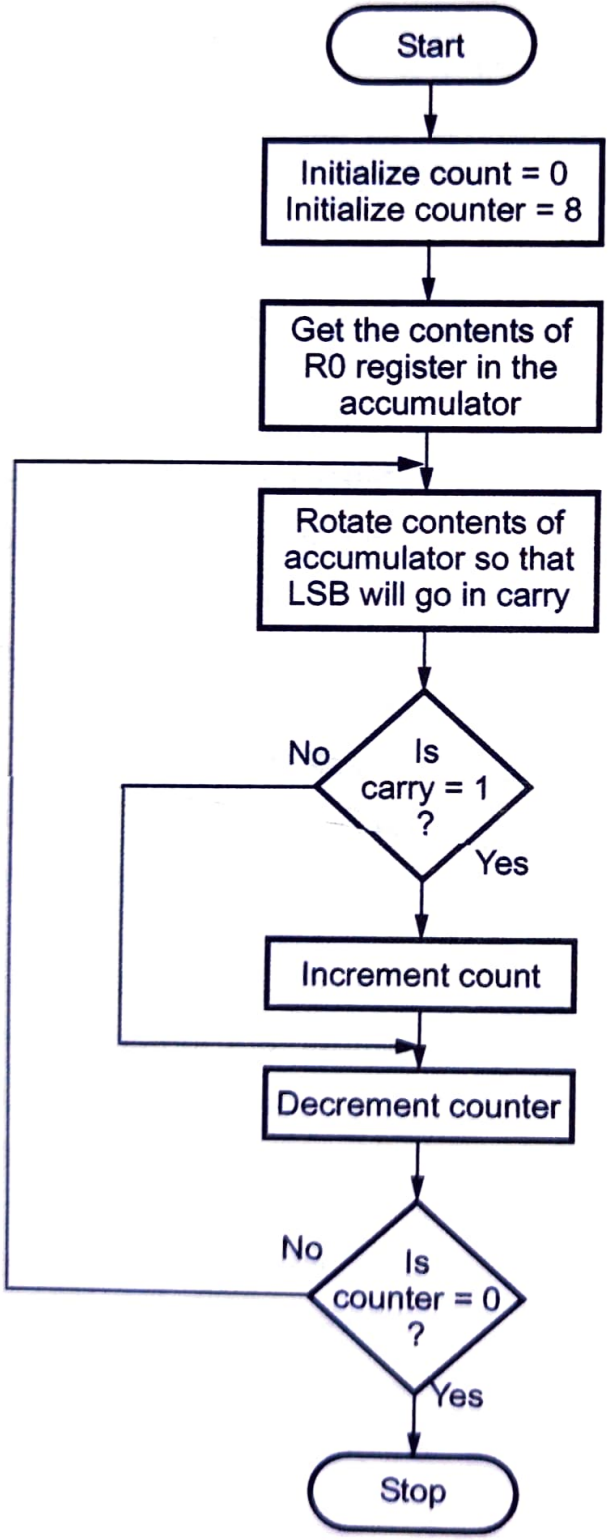



**Fig. 3.7.3**

**Flowchart**

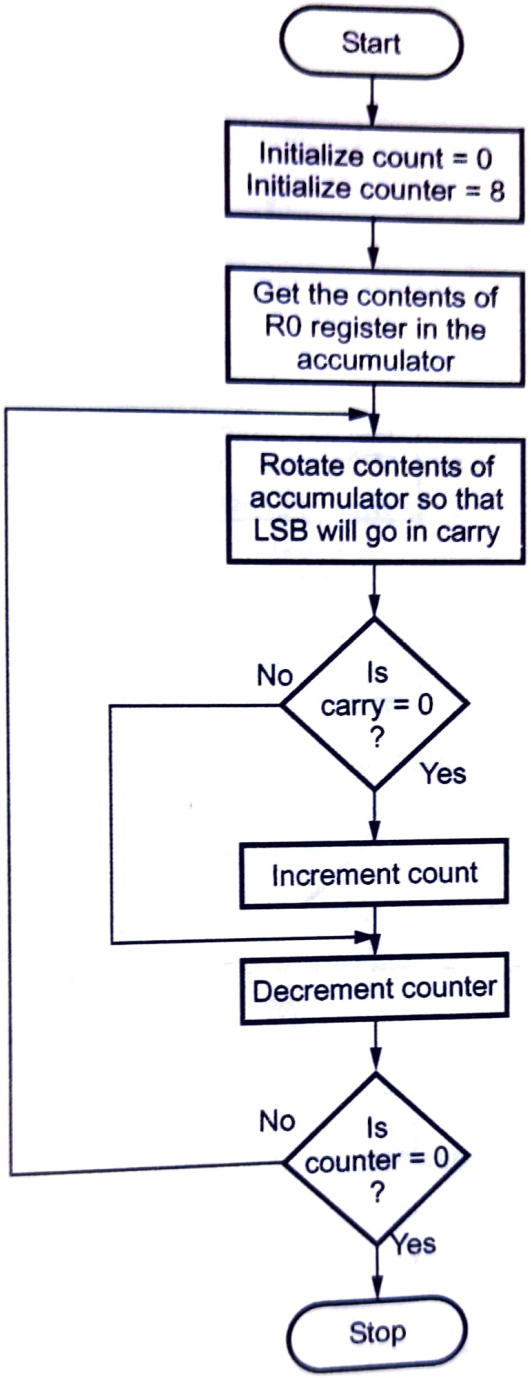



**Fig. 3.7.4**

**Program :**

```
        MOV R2, #0      ; Initialize zero's counter = 0
        MOV R1, #08     ; Initialize iteration count
        MOV R0, #56     ; Load number
        MOV A, R0       ; Get the number in accumulator
BACK :  RRC A           ; Rotate A and CY ← LSB
        JC SKIP         ; If carry is zero go to skip
        INC R2          ; Otherwise increment zero's counter
SKIP :  DJNZ R1, BACK   ; Decrement iteration count and if not zero repeat
```

**Example 3.7.61** *Write a program to load accumulator with the value 55H and complement the content of accumulator 900 times.*

**VTU : June-15, Jan.-18, 19, Marks 5**

**Solution :**

```
        MOV A, #55H        ; load 55 in accumulator
        MOV R0, #10        ; Initialize iteration counter1 = 10H
BACK :  MOV R1, #90        ; Initialize iteration counter2 = 90H
BACK1 : CPL A              ; Complement accumulator
        DJNZ R1, BACK1     ; If counter 2 ≠ 0, repeat
        DJNZ R0, BACK      ; If counter 1 ≠ 0, repeat
```

**Example 3.7.62** *Write a program segment to realize following :*

*i) Exchange contents of external data memory 8100 h with contents of internal data memory 40 h.*

*ii) Exchange contents of A-register and B-register using stack.*

**VTU : Dec.-12, Marks 6**

**Solution :**

i)

```
MOV DPTR, #8100H   ;  Initialize DPTR with external memory address
MOVX A, @DPTR      ;  get the contents of address 8100H from
                   ;  external memory
MOV R1, 40H        ;  get the contents of address 40H from internal
```

2. C is more flexible; it is easier to modify and update.
3. Programming in C allows to use code available in function libraries.
4. Program written in C for one microcontroller is portable to other microcontrollers with little or no modifications.

- In this chapter, we study the C programming for the 8051 and related topics.

## Review Questions

1. *State the reasons for writting 8051 program in C instead of assembly.*
2. *Comment on size of 8051 C program.*

## 4.2 Data Types in 8051 C

**VTU : Jan.-16, 17, 18, 19, July-18, 19**

- The Table 4.2.1 lists the data types that are available in typical C×51 compiler. The Table 4.2.1 gives the information about the size of the data variable and its value range.

| Data type | Bits | Bytes | Value range |
|---|---|---|---|
| bit | 1 | | 0 to 1 |
| signed char | 8 | 1 | – 128 to + 127 |
| unsigned char | 8 | 1 | 0 to 255 |
| enum | 8/16 | 1 or 2 | – 128 to +127 or<br>– 32768 to +32767 |

| | | | |
|---|---|---|---|
| signed short | 16 | 2 | – 32768 to +32767 |
| unsigned short | 16 | 2 | 0 to 65535 |
| signed int | 16 | 2 | – 32768 to +32767 |
| unsigned int | 16 | 2 | 0 to 65535 |
| signed long | 32 | 4 | – 2147483648 to 2147483647 |
| unsigned long | 32 | 4 | 0 to 4294967295 |
| float | 32 | 4 | ± 1.175494E–38 to ± 3.402823E+38 |
| sbit | 1 | | 0 to 1 |
| sfr | 8 | 4 | 0 to 255 |
| sfr16 | 16 | 2 | 0 to 65535 |

**Table 4.2.1 Data types**

## 4.2.1 Using the Appropriate Data Type

It is necessary to keep in mind that the code space for the 8051 is limited to 64 kbytes and it has limited on-chip ROM. Thus, it is necessary to look at the size of the created hex file. One way to keep the size of data file optimum is to use appropriate data type. The following points must be considered while selecting the data type.

- The unsigned char is an 8-bit data type. Thus, it must be used to store the value in the range of 0-255 (00-FFH). Since the 8051 is an 8-bit microcontroller, it is one of the most widely used data type.
- Use unsigned data type when there is no need for signed data.
- C compiler uses the signed data type as the default. Thus when we want to use unsigned data type, we must use **unsigned** keyword.
- S bit data type should be used to access bit addressable registers. It allows use to

**Example 5.3.11** *Write an assembly program to generate a waveform with ON time of 7 msec and OFF time of 21 msec on P0.5. Assume XTAL of 11.0592 MHz. Use timer 0.*

**VTU : June-12, Marks 12**

**Solution :** $T = \dfrac{12}{11.0592 \times 10^6} = 1.085$ m sec

Count to be loaded in TH0 : $TL0 = 65536 - \dfrac{7 \text{ msec}}{1.085\mu \text{ sec}}$

$= 65536 - 6452$ (Decimal)

$65536 - 6452 = 59084 =$ **E6CCH**

```
                ORG  0000H              ; Starting address
START :         MOV  TMOD, #01H         ; Timer 0, mode 1
                MOV  TH0, #0E6H         ; Load timer count
                MOV  TL0, #0CCH
                SETB P0.5               ; Make P0.5 HIGH
                ACALL DELAY7MS          ; Wait for 7 msec (ON period)
                CLR  P0.5               ; Make P0.5 LOW
                ACALL DELAY7MS          ; Wait for 21 msec (OFF period)
                ACALL DELAY7MS
                ACALL DELAY7MS
                SJMP START              ; Repeat
DELAY7MS :      SETB TR0                ; Start Timer 0
HERE :          JNB  TF0, HERE          ; Wait for timer roll over
                CLR TR0                 ; Turn OFF Timer 0
                CLR TF0                 ; Clear TF0
                RET                     ; Return
```

**Example 5.3.12** *Write an assembly language program to generate a square wave of frequency 5 kHz on pin P3.0 using auto reload mode of timer 0 in 8051.*

**Solution :** For 5 kHz $T = 1/5$ kHz $= 200\mu s$ and $T/2 = 100$ µs

Assume clock frequency 12 MHz so time for one cycle is 1 µs. So the value of TH0 and TL0 is 255 – 100 = 155

```
        ORG 0000H
        MOV TMOD, #02H      ;   Timer 0 in mode 2
        MOV TH0,#155        ;   TH0 = 155
        SETB TR0            ;   Start timer 0
HERE :  JNB TF0, HERE
        CPL P3.0
        ORG 001BH
        CLR TF0
        SJMP HERE
```

**Example 5.3.5** *Find the delay generated by timer 0 in the following code. Calculate the delay generated excluding the instruction overhead. What count has to be loaded in TL0 and TH0 if delay has to be increased to 25 msec ?*

```
        CLR P2.3
HERE :  MOV TMOD, #01
        MOV TL0, #3Eh
        MOV TH0, #0B8h
        SETB TF0,
AGAIN:  JNB TF0, AGAIN
        CLR TF0
        CLR TR0
        CLR P2.3.
```

**Solution :**

$$\text{Timer count} = B83E$$

$$\therefore \quad (FFFF - B83E + 1) = 47C2H = (18370)_{10}$$

$$\text{Assuming XTAL} = 11.0592 \text{ MHz}$$

$$T = \frac{12}{11.0592 \times 10^6} = 1.085 \times 10^{-6}$$

$$\therefore \quad \text{Delay} = 1.085 \times 10^{-6} \times 18370 = \mathbf{19.93\ ms}$$

**For 25 msec delay**

$$\text{Decimal count} = \frac{25 \text{ ms}}{1.085 \times 10^{-6}} = 23041$$

$$= 5A01 \text{ H}$$

$$\therefore \quad TL0 = 01H \text{ and } TH0 = 5AH$$

**Example 5.3.6** *A switch is connected to the pin P1.2. Write an 8051 C program to monitor the switch and create the following frequencies on pin P1.7. i) When SW = 0; 500 Hz ii) When SW = 1 ; 750 Hz*
*Use timer 0, mode 1 for both of them.* **VTU : July-16, Jan.-18, Marks 8**

**Solution :**

```
#include <reg51.h>
 sbit sw=P1^2;
 sbit port bit = P1^7;
void Delay (void);
void main (void)
{
```

```
    SW=1;                                   /* Make SW an input */
    while(SW==0)
    {       portbit = ~portbit;             /* toggle portbit */
            For (i=0; i<20; i++)
                    Delay();
    while(SW==1)
    {
            portbit = ~portbit;             /* toggle portbit */
            for (i=0; i<30 ; i++)
            Delay ();

    }
}
void Delay (void)
{
    TMOD = 0x01;                            /* Timer 0, mode 1 */
    TL0  = 0xFE;                            /* Load TL0 */
    TH0  = 0xA5;                            /* Load TH0 */
    TR0  = 1 ;                              /* Turn ON timer0 */
    While (TF0==0);                         /* Wait for TF0 to roll over */
    TR0=0;                                  /* Turn OFF T0 */
    TF0=0;                                  /* Clear TF0 */
}
```

$(A5FE)_H = (42494)_{10}$

$65536 - 42494 = 23092$

$23092 \times 1.085\ \mu s = 25\ ms$

$\therefore \quad 500\ ms = 25\ ms \times 20$

$750\ ms = 25\ ms \times 30$

**Example 5.3.7** *Write a 8051 C program to toggle all bits of port P0 continuously. Use timer 0 to generate the delay of 1 sec between each toggle.*

**Solution :**

<reg51.h> statement in the program.

**Example 4.4.8** *Write an 8051 C program to toggle all the bits of P0 and P2 continuously with a some delay. Use the sfr keyword to declare the port addresses.*

**Solution :**

```
/* Accessing ports as SFRs using the sfr data type */
sfr P0 = 0x80;      /* declaration of P0 using sfr data type */
sfr P2 = 0xA0;      /* declaration of P2 using sfr data type */
void Delay (unsigned int);
void main(void)
{
        while(1)            /* read it continuously */
        {
                P0=0x55;
                P2=0x55;
                Delay(250);  /* delay */
                P0=0xAA;
                P2=0xAA;
                Delay(250);
        }
}
void Delay (unsigned int count)
{
        unsigned int i, j;
        for (i=0; i<count; i++)
        for (j=0; j<1200; j++)
                ;
}
```

**Example 4.4.9** *Write an 8051 C program to turn bit P0.5 on and off 10,000 times.*

**Solution :**

```
sbit PORTBIT = 0x85;    /* another way to declare bit P0^5 */
void main(void)
{
```

```
        unsigned int i;
        for(i=0; i<10000; i++)
        {
                PORTBIT=1;
                PORTBIT=0;
        }
}
```

### 4.4.4 Using Bit Data Type for Bit-Addressable RAM

- We can use sbit data type with only bit addressable SFR registers. To access bit addressable section of the data RAM space 20-2FH, we can use bit data type. This is illustrated in example 4.4.10.

**Example 4.4.10** *Write an 8051 C program to get the status of bit P0.0, complement it and send it to P1.5 continuously.*

**Solution :**

```
#include <reg51.h>
sbit inbit = P0^0;
sbit outbit = P1^5;
bit tempbit;
void main(void)
{
        while(1)
        {
                tempbit=inbit;      /* read a bit from P0.0 */
                if(tempbit==1)      /* complement the bit */
                        tempbit=0;
                else
                        tempbit=1;
                outbit=tempbit;     /*send it to P1.5 */
        }
}
```

$5000 \times 1\ \mu s = 5\ ms$

$1 / (5\ ms \times 2) = 100\ Hz$

### C Programming of Timers 0 and 1 as Counters

Let us see how to use timers 0 and 1 as event counters. A timer can be used as a counter if we provide external clock instead of using the frequency of the crystal oscillator as the clock source. By feeding pulses to the T0 (P3.4) and T1 (P3.5) pins, we can use timer 0 and timer 1 as counter 0 and counter 1, respectively. Following examples show us how timers 0 and 1 are programmed as counters using the C language.

**Example 5.2.8** *Assume that a 1-Hz external clock is being fed into pin T1. Write a C program for counter 1 in mode 2 (8-bit auto reload) to count up and display the state of the TL1 count on P0. Start the count from 00H.*

**Solution :**

```
#include <reg51.h>
sbit T1 = P3^5;
void main(void)
{
    T1=1;                       /* make T1 an input */
    TMOD=0x60;
    TH1=0;                      /* set count to 0 */
    while(1)                    /* repeat forever */
    {
        TR1=1                       /* start timer */
        do
        {
            P0=TL1;             /* place value on port P0 */
        }
        while(TF1==0);          /* wait for TF1 to rollover */
        TR1=0;                      /* stop timer */
        TF1=0;                      /* clear flag */
    }
}
```

**Example 5.2.9** *Assume that a 1 Hz external clock is being fed into pin T0. Write a C program for counter 0 in mode 1 (16 bit) to count the pulses and display the TH0 and TL0 registers on P1 and P0, respectively.* **VTU : July-19, Marks 6**

```
        RETI        ;   Clear RI
        END         ;   Return from ISR
```

**Count for 5 kHz**

$$T = \frac{1}{5\ \text{kHz}} = 2\times 10^{-4}\ \text{s}$$

$$T_{ON} = T_{OFF} = T/2 = 1\times 10^{-4}\ \text{s}$$

$$\text{Count} = \frac{\text{Crystal frequency}}{12} \times 1\times 10^{-4}$$

$$= \frac{1}{1.085\times 10^{-6}} \times 1\times 10^{-4} \approx 92$$

**Example 6.5.11** *Write 8051 program to receive bytes of data serially and put them in P1, Set the baud rate as 4800, 8 bit data and one stop bit.* **VTU : Jan.-18, Marks 6**

**Solution :**

```
        MOV TMOD, #20H      ;   Timer 1, mode 2 (auto-reload)
        MOV TH1, #–6        ;   4800 baud rate
        MOV SCON, #50H      ;   8-bit, 1 stop bit,REN enabled
        SETB TR1            ;   start Timer 1
HERE:   JNB RI, HERE        ;   Wait for character to come in
        MOV A, SBUF         ;   save incoming byte in A
        MOV P1, A           ;   Send received byte to port 1
        CLR RI              ;   Get ready to receive next byte
        SJMP HERE           ;   Continue receiving data
```

**Example 6.5.12** *Write instructions to initialize serial port in mode 1 with baud rate of 4800 and crystal frequency of 11.059 MHz.*

**Solution :**

```
MOV TMOD, #20H        ; timer 1, mode 2 (auto reload)
MOV TH1, #FAH         ; count for baud rate 4800
MOV SCON, #50H        ; 8-bit, 1 stop, mode 1
SETB TR1              ; start timer 1
```

$$TH1 = 256 - \frac{K \times OSC\ Freq.}{384 \times Baud\ rate}$$

$$= 256 - \frac{1 \times 11.059\ MHz}{384 \times 4800}$$

$$= 250 = FAH$$

**Example 6.5.13** *Write 8051 assembly program to receive the data in serial from and send it out to port-0 in parallel from save the data in RAM location 62h. Assume baud rate = 9600. Use timer 1 in mode 2.* **VTU : Jan.-19, Marks 5**

**Solution :**

```
        MOV TMOD, #20 H     ;  Timer 1, mode 2 (auto load)
        MOV TH1, #FD H      ;  9600 band rate
        MOV SCAN, # 50 H    ;  8-bit, 1 stop, REN enabled
        SETB TR1            ;  Start timer 1
HERE :  JNB RI, HERE        ;  Wait for character receive completely
        MOV A, SBUF         ;  Save the received character
        MOV P0, A           ;  Send character to port 0
        MOV 62H, A          ;  Save the data in RAM
        CLR RI              ;  Get ready to receive next byte
        SMP HERE            ;  Go to receive next character
```

**Example 6.5.14** *Write a program using interrupts to do the following :*

*i) Receive data serially and send it to $P_0$ ii) Have $P_1$ port read and transmitted serially and a copy given to $P_2$*

**Example 6.6.2** *Write an 8051 C program to transfer the message "HELLO" serially at 9600 baud, 8-bit data 1 stop bit.* **VTU : Jan.-17, Marks 5**

**Solution :**

```
# include <reg51.h>
void Trans (unsigned char);
void main (void)
{
    TMOD = 0x20 ;       // use Timer 1 , 8-BIT auto-reload
    TH1 = 0xFD ;            // 9600 baud rate
    SCON = 0x50 ;
    TR1 = 1 ;           // start timer

    Trans ( 'H' ) ;
    Trans ( 'E' ) ;
    Trans ( 'L' ) ;
    Trans ( 'L' ) ;
    Trans ( 'O' ) ;
}
void Trans (unsigned char ch)
{
    SBUF = ch ;         // place value in buffer
    while (TI==0)       ;   // wait until transmitted
    TI = 0;
}
```

**Example 6.6.3** *Write an 8051 C program to send two different strings to the serial port. Assuming that SW is connected to pin P2.0, monitor its status and make a decision as follows :*

*SW = 0, send your first name*

*SW = 1, send your last name.*

*Assume XTAL = 11.0592 MHz, baud rate of 9600, 8 bit data 1 stop bit.* **VTU : Jan.-17, Marks 10**

**Review Questions**

1. *Explain RS-232 handshaking signals and specify the purpose of max-232 while interfacing.*

**VTU : Aug.-09, Marks 7**

2. *Explain the need for MAX 232 line driver for connecting RS232 to 8051. Show the interface of RS232 to 8051 using MAX 232.*

**VTU : July-18, Marks 6**

## 6.5 Serial Communication Programming in Assembly

**VTU : Feb.-06, 07, 08, Aug.-08, 09, Dec.--07, 09, 11, 12, June-12, 13, 14, Jan.-14, 16, 17, 18, 19, July-19**

### 6.5.1 Programming 8051 for Serial Data Transfer

- To program 8051, to transfer data serially we have to perform following sequence of actions :

1. Load the TMOD register with the value 20H to use timer 1 in mode 2 (8-bit auto-reload) to set the baud rate.
2. Load TH1 to set the desire baud rate for serial data transfer.
3. Load SCON register with the value 50H, to use serial mode 1, where an 8-bit data is framed with start and stop bits.
4. Set TR1 to 1 to start timer 1.
5. Clear TI with **CLR TI** instruction.
6. Write a character to be sent in to the SBUF register.
7. Check the TI flag bit with instruction **JNB TI, XXXX** to see if the character has been transferred completely.
8. Go to step 5 to transfer the next character.

**Example 6.5.1** *8051 uses 11.0592 MHz crystal. To get 9600 hertz baud rate how will you program it for serial transmission ?*

**Solution :** When 11.0592 MHz crystal is used and a standard baud rate of 9600 hertz is required then, the setting of TH1 can be found as

$$TH1 = 256 - \frac{k \times \text{Oscillator frequency}}{384 \times \text{Baud rate}}$$

**The Importance of the TI flag bit.**

- When a data is to be transmitted via TxD pin, first a data byte is loaded into the SBUF register. A start bit, a data byte and then the stop bit are transmitted sequentially via TxD pin.
- During the transmission of the stop bit, 8051 sets the TI flag, i.e. TI = 1. This indicates the end of data byte transmission and 8051 is ready for the transmission and 8051 is ready for the transmission of next data.
- The programmer has to clear the TI flag, i.e. TI = 0, with the 'CLR TI' instruction to transmit next data. The TI flag bit should be monitored to make sure that the SBUF register is not overwritten. If we write the next byte to be transmitted into the SBUF register before setting the TI flag bit, the untransmitted portion of the previous byte will be lost. The programmer can check the TI flag bit by JNB TI, XX' instruction or by using an interrupt.

### 6.5.2 Programming 8051 for Receiving Data Serially

- To program 8051, to receive data serially we have to perform following sequence of actions :

1. Load the TMOD register with the value 20H to use timer 1 in mode 2 (8-bit auto-reload) to set the baud rate.
2. Load TH1 to set the desire baud rate for serial data transfer.
3. Load SCON register with the value 50H, to use serial mode 1, where an 8-bit data is framed with start and stop bits.
4. Set TR1 to 1 to start timer 1.
5. Clear RI with **CLR RI** instruction.
6. Check the RI flag bit with instruction **JNB RI, XXXX** to see if an entire character has been received yet.
7. If RI is set, SBUF has the byte. Save this byte.
8. Go to step 5 to receive the next character.

**Example 6.5.4** *Write an 8051 assembly language program to receive bytes serially with baud rate 9600, 8-bit data and 1 stop bit. Simultaneously send received bytes to port 2.*

**Solution :**

| Sr. No. | Polling method | Interrupt method |
|---|---|---|
| 1. | In polling, a lot of microcontroller's processing power is wasted in checking whether the I/O devices need service. | In interrupt method, microcontroller's time is not wasted in checking whether the I/O devices need service. |
| 2. | Since all the devices are polled in a sequential manner, there is no explicit priority mechanism in the polling method. | Interrupt method can assign priorities to the interrupts so that they can be served according to priority basis. |
| 3. | In polling, there is no possibility of ignoring a device request for service. | In interrupt method, the microcontroller can ignore (mask) a device request for service. |
| 4. | Inefficient way of handling I/O devices. | Efficient way of handling I/O devices. |
| 5. | No external hardware signal required. | External hardware signal is required to interrupt microcontroller. |

**Table 7.1.1 Comparison of interrupt method and polling**

## 7.1.1 Steps in Executing the Interrupt

- In response to the interrupt, the microcontroller goes through following steps :
  1. It completes the execution of current instruction and save the address of the next instruction, i.e. the contents of PC on the stack.
  2. It also saves the current status of all the interrupts internally i.e. not on the stack.

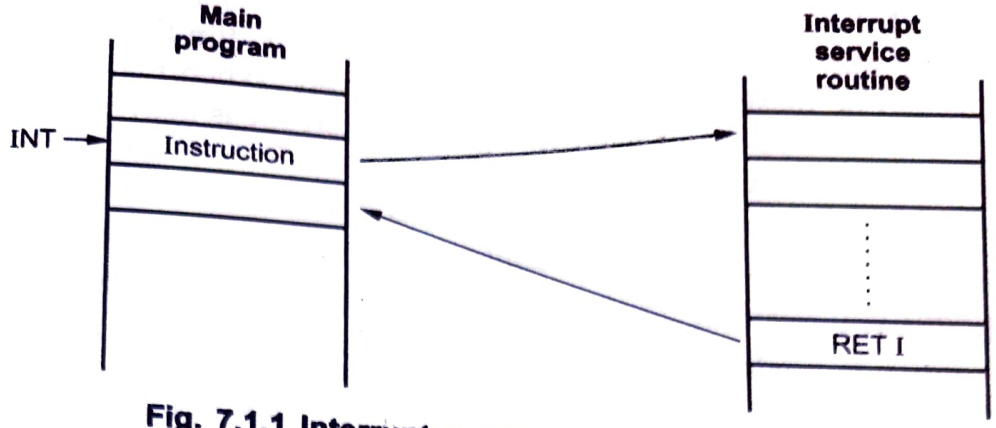



**Fig. 7.1.1 Interrupt service routine processing**

3. It then transfers the program control to a fixed location in memory called the **Interrupt Vector Table (IVT)**. IVT holds the address of the interrupt service routine.

4. The microcontroller gets the address of the Interrupt Service Routine (ISR) from IVT and transfers program control to the ISR. It executes instructions within the interrupt service routine in sequence till it reaches the RETI (Return From The Interrupt) instruction.

5. Upon executing the RETI instruction, it gets the Program Counter (PC) address from the stack and restores the interrupt logic. Since PC is loaded with the address of the next instruction from where the interrupt was initiated, it starts to execute from that address.

**Review Questions**

1. *What is interrupt method ? How it is different from polling method.*
2. *Explain the steps in executing the interrupt.* **VTU : Jan.-19, Marks 4**
3. *What are the steps microcontroller perform upon activation of an interrupt.* **VTU : Aug.-11, Marks 6**
4. *Compare polling and interrupts.* **VTU : Jan.-16, Marks 4**

## 7.1.2 Classification of Interrupts

- An interrupt caused by an external signal is referred as a **hardware interrupt** or **external interrupt**. Conditional interrupts or interrupts caused by special instructions are called **software interrupts or internal interrupts**.
- The interrupts are further classified as maskable interrupts, non-maskable interrupts, vector interrupts and non-vector interrupts.

1. **Maskable Interrupts** : Maskable interrupts are enabled and disabled under program control. By setting or resetting particular flip-flops in the processor, interrupts can be masked or unmasked, respectively. When masked, processor does not respond to the interrupt even though the interrupt is activated. Such interrupts are called Maskable interrupts.
2. **Non-Maskable Interrupts** : The interrupts which can not be masked under software control are called non-maskable interrupts.
3. **Vectored Interrupts** : The interrupt for which the starting address of the interrupt service routine is predefined is called vectored interrupts. Since the starting address of interrupt service routine is predefined the interrupt response time for vector interrupts is less compared to non-vectored interrupts.
4. **Non-vectored Interrupts** : The interrupt for which the starting address of the interrupt service routine is not predefined and it is provided by the external hardware at the time of interrupt is called non-vectored interrupt.

**Review Question**

1. *Give the classification of interrupts.*

## 7.2 Types of Interrupts in 8051

**VTU : Feb.-06, 07, 10, Aug.-06, 07, 09, 10, 11, Dec.-11, 12, June-12, 13, 14, Jan.-15, 16, 18, 19, July-16, 18, 19**

- The 8051 provides 5 interrupt sources. The 8052 provides 6. These are shown in Fig. 7.2.1. The external Interrupts $\overline{INT0}$ and $\overline{INT1}$ can each be either level-activated or transition-activated, depending on bits IT0 and IT1 in Register TCON. 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

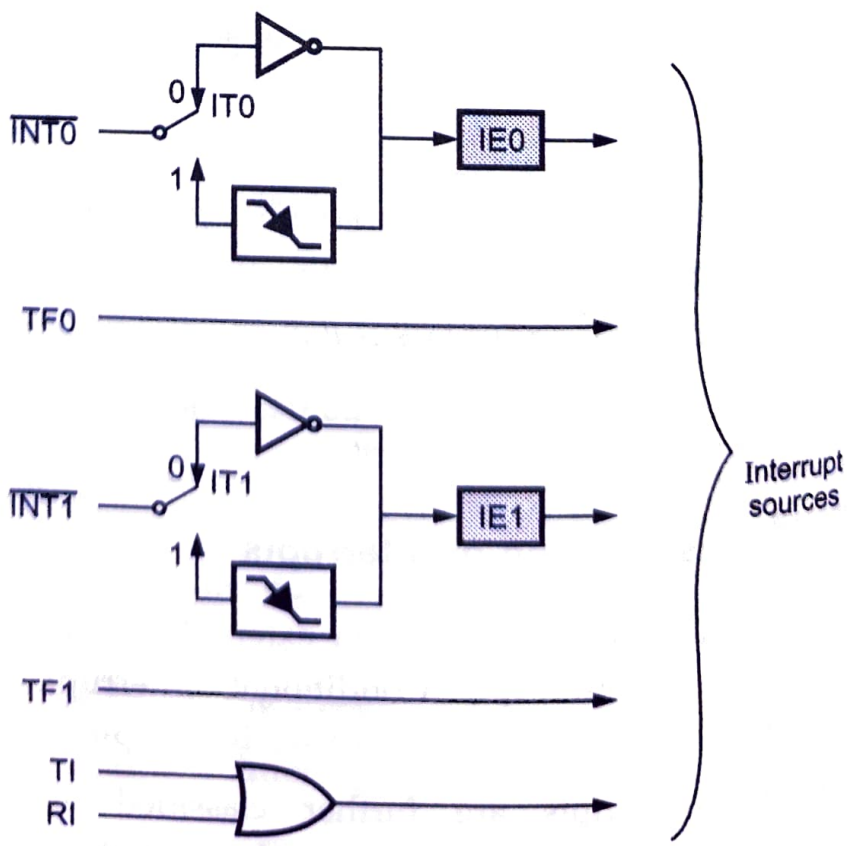



**Fig. 7.2.1 MCS-51 interrupt structure**

the interrupt was transition-activated. If the interrupt was level-activated, then the external requesting source is what controls the request flag, rather that 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 (except see Timer 0 in Mode 3). The timer flag set upon generation of interrupt is cleared by the on-chip hardware when microcontroller starts execution of particular interrupt service routine.
- 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 service routine. 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.

### 7.2.1 Interrupt Control (Enabling and Disabling Interrupts using IE)

- When 8051 is reset, all interrupts are disable. These are enabled by software. All of 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 canceled in software.
- Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE (Fig. 7.2.2). IE contains also a global disable bit, EA, which disables all interrupts at once.
- Note in Fig. 7.2.2 that bit position IE.6 is unimplemented. In the 8051s, bit position IE.5 is also unimplemented. User software should not write 1s to these bit positions, since they may be used in future MCS-51 products.

| (MSB) | | | | | | | (LSB) |
|---|---|---|---|---|---|---|---|
| EA | - | ET2 | ES | ET1 | EX1 | ET0 | EX0 |

| Symbol | Position | Name and Significance |
|---|---|---|
| EA | IE.7 | Enable all control bit.<br>Cleared by software to disable all interrupts, independent of the state of IE.4-IE.0. |
| - | IE.6 | (Reserved) |
| ET2 | IE.5 | (Reserved) |
| ES | IE.4 | Enable Serial port control bit.<br>Set/cleared by software to enable/disable interrupts from TI or RI flags. |

| | | |
|---|---|---|
| ET1 | IE.3 | Enable Timer 1 control bit.<br>Set/cleared by software to enable/disable interrupts from timer/counter 1 |
| EX1 | IE.2 | Enable External interrupt 1 control bit.<br>Set/cleared by software to enable/ disable interrupts from INT1. |
| ET0 | IE.1 | Enable Timer 0 control bit.<br>Set/cleared by software to enable/disable interrupts from timer/counter 0. |
| EX0 | IE.0 | Enable external interrupt 0 control bit.<br>Set/cleared by software to enable/disable interrupts from INT0. |

**Fig. 7.2.2 IE-Interrupt enable register**

**Example 7.2.1** *Write a program to enable serial interrupt, Timer 1 interrupt and external hardware interrupt 0 (EX0).*

**Solution :**

| EA | – | – | ES | ET1 | EX1 | ET0 | EX0 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

```
MOV IE, #1001 1001B        ;    enable serial, Timer 1, EX0
```

**Example 7.2.2** *Write an instruction to disable all interrupts.*

**Solution :** By making EA = 0 we can disable all interrupts.

```
CLR  IE.7        ;    Clear EA bit in the IE register
```

**Example 7.2.3** *Write a program to enable Timer 0 interrupt using bit manipulation instructions.*

**Solution :**

```
SETB    IE.7  ;    Enable interrupts
SETB    IE.1  ;    Enable timer 0 interrupt
```

## 7.2.2 Interrupt Priority and Interrupt Destinations (Vector Locations)

- Each interrupt source can also be individually programmed to one of two priority levels by setting or clearing a bit in Special Function Register IP (Fig. 7.2.3). 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 any other interrupt source.

(MSB) (LSB)

| - | - | - | PS | PT1 | PX1 | PT0 | PX0 |
|---|---|---|---|---|---|---|---|

| Symbol | Position | Name and Significance |
|---|---|---|
| - | IP.7 | (Reserved) |
| - | IP.6 | (Reserved) |
| - | IP.5 | (Reserved) |
| PS | IP.4 | Serial port Priority control bit. Set/cleared by software to specify high/low priority interrupts for Serial port. |
| PT1 | IP.3 | Timer 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 1. |
| PX1 | IP.2 | External interrupt 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT1. |
| PT0 | IP.1 | Timer 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 0. |
| PX0 | IP.0 | External interrupt 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT0. |

**Fig. 7.2.3 IP-interrupt priority control register**

- If two requests of different priority levels are received simultaneously, the request of higher priority level is serviced. If requests of the same priority level are received simultaneously, an internal polling sequence determines which request is serviced. Thus within each priority level there is a second priority structure determined by the polling sequence, as follows :

| Interrupt | Vector location | Priority within level |
|---|---|---|
| External hardware interrupt 0 (INT0) | 0003H | (highest) |
| Timer 0 interrupt (TF0) | 000BH | |
| External hardware interrupt 1 (INT1) | 0013H | |
| Timer 1 interrupt (TF1) | 001BH | |
| Serial communication interrupt (RI and TI) | 0023H | (lowest) |

**Table 7.2.1 Interrupt vector table (IVT) and priority levels**

Note that the "priority within level" structure is only used to resolve simultaneous requests of the same priority level.

1. *Give the classification of interrupts.*
2. *Explain interrupt structure of 8051 and facility available for programming of interrupts.*

**VTU : Feb.-06, Marks 7**

3. *What is the default priority assigned to various interrupts after reset ? Is it possible to alter priority ?*

**VTU : Feb.-06, Marks 5**

4. *What are external interrupts ? How external interrupts are handled in 8051 ? How level triggered interrupts are reset ? Set INT1 interrupt edge triggered.*

**VTU : Aug.-06, Marks 7**

5. *Explain the two hardware interrupt pins $\overline{INT0}$ and $\overline{INT1}$ of 8051 with its concerned registers. Given an application, which uses these pins.*

**VTU : Feb.-07, Marks 10**

6. *Explain the different interrupts of 8051. State how the priority of interrupts in this microcontroller can be altered by the programmer.*

**VTU : Aug.-07, Marks 10**

7. *Explain IE and IP registers with their bit pattern and show how priorities change with example.*

**VTU : Aug.-09, Marks 10**

8. *What are interrupts and interrupt subroutines ? Explain the interrupts that are present in 8051.*

**VTU : Feb.-10, Marks 6**

9. *Explain the different interrupt vector addresses in 8051.*

**VTU : Aug.-10, Dec.-11, June-12, July-18, Marks 4**

10. *How the interrupts in 8051 is classified ? Explain each interrupt.*

**VTU : Aug.-11, Dec.-11, Marks 6**

11. *With regard to the interrupts of 8051,*

**VTU : June-12, Marks 6**

    i) *Briefly explain the procedure of enabling/disabling the entire interrupt system and enabling/disabling of individual interrupts.*

    ii) *Indicate the default priority on reset and procedure to alter this default priority.*

12. *Explain briefly the interrupts of 8051, indicate their vector addresses.* **VTU : June-13, Marks 5**
13. *List out the various types of interrupts and also write the bit pattern of IE SFR with explanation with respect to 8051.*

**VTU : June-14, Marks 6**

14. *What are interrupts ? Specify vector location or interrupts in 8051.* **VTU : Jan.-15, Marks 4**
15. *Explain the interrupts of 8051 clearly mentioning the vector address and priorities.*

**VTU : Dec.-12, Jan.-18, Marks 6**

16. *Explain the six interrupt of 8051, with priority and interrupt vector table.*

**VTU : Jan.-16, July-18, 19, Marks 8**

17. *What are external interrupts? Draw the diagrams for activation of external interrupts. How level triggered interrupts are reset? How to set the two external interrupts as edge triggered interrupts ?*

**VTU : July-16, Marks 8**

18. *Explain the bit status of IP register.*

**VTU : Jan.-19, Makrs 4**

**Review Question**

1. *Write a note on programming external hardware interrupt.*

## 7.4 Programming Timer Interrupts

**VTU : June-12, Jan.-18**

- 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 (except see Timer 0 in Mode 3). When a timer interrupt is generated, the flag that generated it is cleared by the on-chip hardware when the service routine is vectored. Vector address for timer 0 is **000BH** timer 1 is **001BH.**
- Recall the concepts studied in previous section. In that section we have seen the use of timer 0 and timer 1 with the polling method. Here, we are discussing the use of interrupts to program 8051 timers. We know that the Timer Flag (TF) is set (=1) when the timer rolls over. In polling method, the TF is monitored with the instruction 'JNB TF, target address'. We have to wait until the TF is raised. The problem with this polling method is that 8051 can not do anything else until TF is set to high.
- This problem can be solved using interrupt method. If the timer interrupt in the IE register is enabled, TF is set whenever the timer is rolled over and the 8051 is interrupted. Thus the 8051 can perform anything else until it is interrupted. After interruption (timer rolling over) only the 8051 remains busy in executing interrupt service routine.

**Example 7.4.1** *Write an 8051 ALP that continuously reads 8-bit data from port 2 and sends it to port 0. At the same time it should generate square wave of 500 μs period on port 1.0. Assume the crystal frequency = 11.0592 MHz.* **VTU : Jan.-18**

**Solution :** We will use timer 0 in autoreload mode, i.e. mode 2. To generate square wave of 500 μs we have to toggle port 1.0 pin after every 250 μs.

$$\text{Timer clock frequency} = \frac{11.0592 \times 10^6}{12} = 921.6 \text{ kHz}$$

$$\therefore \quad TH0 = 256 - (250\ \mu s \times 921.6 \times 10^3)$$

$$\approx 26 = 1\ AH$$

**Program :**

```
        ORG 0000H
        LJMP MAIN           ; Avoid using memory space
                            ; allocated to interrupt vector table

        ORG 000BH           ; ISR for Timer 0 interrupt
        CPL P1.0            ; Complement P1.0 bit
        RETI                ; return from ISR

        ORG 0030H           ; Start main program after
                            ; interrupt vector table
MAIN :  MOV TMOD, #02H      ; Initialize timer 0 in mode 2
        MOV P2, #0FFH       ; Configure Port 2 as input
        MOV TH0, #AH        ; Load timer count
        MOV IE, #82H        ; Enable timer 0 interrupt
        SETB TR0            ; Start timer 0
BACK :  MOV A, P2           ; Read data from P2
        MOV P0, A           ; Send it on P0
        SJMP BACK           ; Repeat
        END
```

**Example** *... 8051 to generate a square wave of frequency 5 kHz on*

**Review Questions**

1. *Define serial data transfer and parallel data transfer.*
2. *Give comparison between serial and parallel data transfer.*

## 6.2 Basics of Serial Data Communication

**VTU : Feb.-08, 10, Aug.-08, 09, June-10, 12, 13, 14, Dec.-11, Jan.-14, 16, 18**

- In serial data communication, during transmission parallel data is converted into serial bits using a parallel-in serial-out (PISO) shift register. At the receiver, the serial bits are connected into parallel data by another shift register called serial-in parallel-out (SIPO) shift register.

### 6.2.1 Classification

- Serial data transmission can be classified on the basis of how transmission occurs.

1. Simplex 2. Half duplex 3. Full duplex

#### 6.2.1.1 Simplex

- In simplex, the hardware exists such that data transfer takes place only in one direction. There is no possibility of data transfer in the other direction. A typical example is transmission from a computer to the printer.

#### 6.2.1.2 Half Duplex

- The half duplex transmission allows the data transfer in both directions, but not simultaneously. A typical example is a walkie-talkie.

#### 6.2.1.3 Full Duplex

- The full duplex transmission allows the data transfer in both direction simultaneously. The typical example is transmission through telephone lines.

### 6.2.2 Types of Serial Data Communication

- Serial data communication uses two types of communications
  - Asynchronous serial data communication
  - Synchronous serial data communication

#### 6.2.2.1 Asynchronous Serial Data Communication

- Fig. 6.2.1 shows the transmission format for asynchronous transmission. Asynchronous formats are character oriented. In this, the bits of a character or data word are sent at a constant rate, but characters can come at any rate

(asynchronously) as long as they do not overlap. When no characters are being sent, a line stays high at logic 1 called **mark**, logic 0 is called **space**.

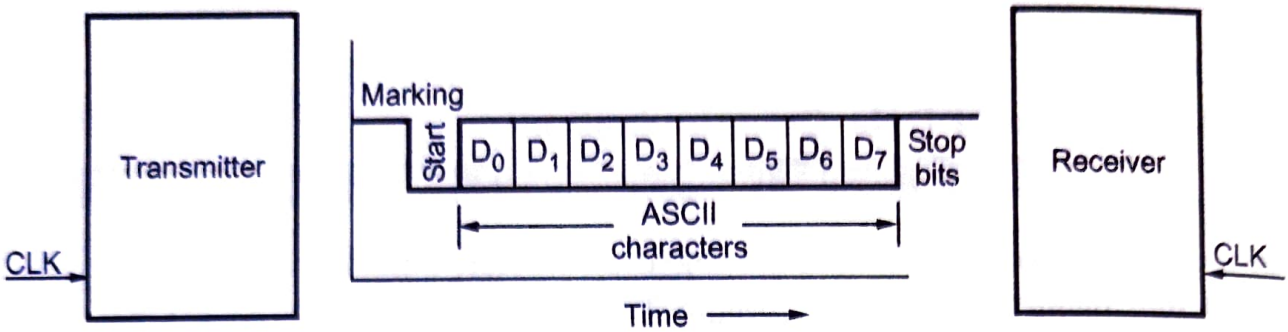



**Fig. 6.2.1 Transmission format for asynchronous transmission**

- The beginning of a character is indicated by a start bit which is always low. This is used to synchronize the transmitter and receiver. After the start bit, the data bits are sent with least significant bit first, followed by one or more stop bits (active high). The stop bits indicate the end of character. Different systems use 1, 1 1/2 or 2 stop bits. The combination of start bit, character and stop bits is known as **frame.**
- The start and stop bits carry no information, but are required because of the asynchronous nature of data. Fig. 6.2.2 illustrates how the data byte CAH would look when transmitted in the asynchronous serial format.

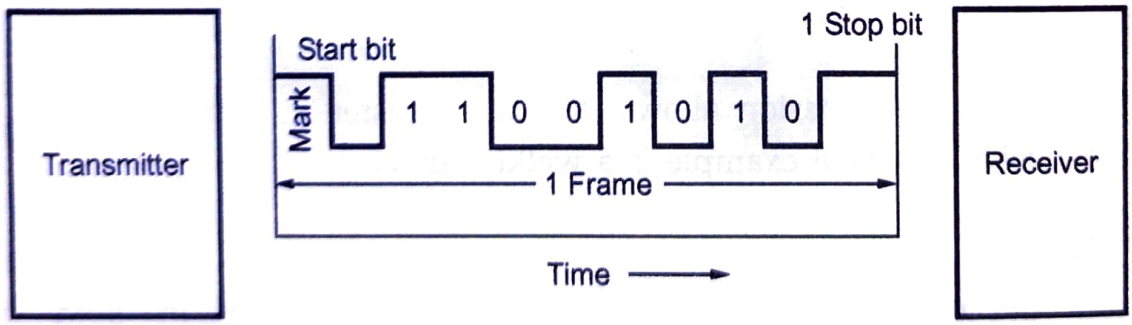



**Fig. 6.2.2 Asynchronous format with data byte CAH**

- The data rate can be expressed as bits/sec or characters/sec. The term bits/sec is also called the **baud rate**. The asynchronous format is generally used in low-speed transmission (less than 20 kbits/sec).

### 6.2.2.2 Synchronous Serial Data Communication

- The start and stop bits in each frame of asynchronous format represents wasted overhead bytes that reduce the overall character rate. These start and stop bits can be eliminated by synchronizing receiver and transmitter.

- They can be synchronized by having a common clock signal. Such a communication is called **synchronous serial communication**. The Fig. 6.2.3 shows the transmission format of synchronous serial communication. In this transmission synchronous bits are inserted instead of start and stop bits.

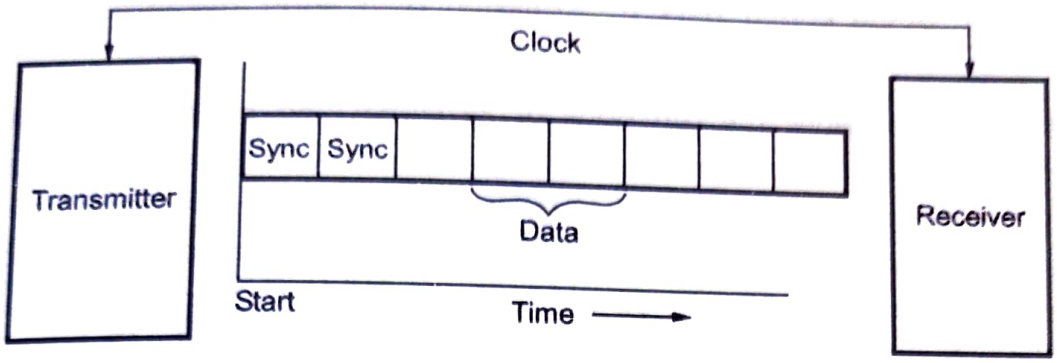



**Fig. 6.2.3 Synchronous transmission format**

| Sr. No. | Asynchronous serial communication | Synchronous serial communication |
|---|---|---|
| 1. | Transmitters and receivers are not synchronized by clock. | Transmitters and receivers are synchronized by clock. |
| 2. | Bits of data are transmitted at constant rate. | Data bits are transmitted with synchronization of clock. |
| 3. | Character may arrive at any rate at receiver. | Character is received at constant rate. |
| 4. | Data transfer is character oriented. | Data transfer takes place in blocks. |
| 5. | Start and stop bits are required to establish communication of each character. | Start and stop bits are not required to establish communication of each character; however, synchronization bits are required to transfer the data block. |
| 6. | Used in low-speed transmissions at about speed less than 20 kbits/sec. | Used in high-speed transmissions. |

**Table 6.2.1 Comparison between asynchronous and synchronous serial data transfer**

## 6.2.3 Baud Rate

- Baud rate is data transmission speed. The rate at which the bits are transmitted (bits/second) is called **baud** or **transfer rate**. The baud rate is the reciprocal of the time to send 1-bit. The common baud rates are multiples of 75 bits per second (bit/s) are typically : 75, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400 bit/s

**Example 8.11.4** *Write an ALP to rotate the stepper motor in clockwise direction, if the status of a switch connected to port pin P1.2 is ON. Otherwise rotate it in counterclockwise direction.*

**Solution :**

```
        ORG  0H                     ; Starting address
MAIN :  SETB P1.2                   ; Make it an input
        MOV DPTR, #ETC              ; Initialize base pointer to excitation code table
        MOV R0, #00H                ;  Initialize pointer to excitation code table
BACK :  JB P1.2, ACW                ; Check switch status
        CJNZ R0, #04, NEXT          ; Roll over the sequence code
        MOV R0, #00H
NEXT :  MOV A, R0                   ; Copy pointer in A
        MOVC A, @A + DPTR           ; Get the code to rotate clockwise
        MOV P1, A                   ; Send code to port 1
        LCALL Delay                 ; Wait for some time
        INC R0                      ; Point to next code
        SJMP BACK                   ; Repeat
ACW :   CJNZ R0, #OFFH, NEXT1       ; Roll over the sequence code
        MOV R0, #03H
NEXT1 : MOV A, R0                   ; Copy pointer in A
        MOVC A, @A + DPTR           ; Get the code to rotate anticlockwise
        MOV P1, A                   ; Send code to port 1
        LCALL Delay                 ; Wait for some time
        DEC R0                      ; Point to previous code
        SJMP BACK                   ; Repeat
Delay : MOV R2, #100
AGAIN1 : MOV R3, #255
AGAIN : DJNZ R3, AGAIN
        DJNZ R2, AGAIN1
        RET
        ORG 0100H
        ETC  DB  05H, 06H, 0AH, 09H
```

**Example 8.11.5** *Show the interfacing of a stepper motor to 8051 and write 8051 assembly/8051 'C' program to rotate stepper motor 2 rotations clockwise and one rotation anticlockwise with appropriate delay.*

## 9.2.2 Control Word Formats

- The ports of the 8255 can then be programmed for any other mode by writing a single control word into the control register, when required.

### 9.2.2.1 For Bit Set/Reset Mode

- Fig. 9.2.3 shows bit set/reset control word format. The eight possible combinations of the states of bits $D_3$ - $D_1$ ( $B_2$ $B_1$ $B_0$) in the Bit Set-Reset format (BSR) determine particular bit in $PC_0$ - $PC_7$ being set or reset as per the status of bit $D_0$.

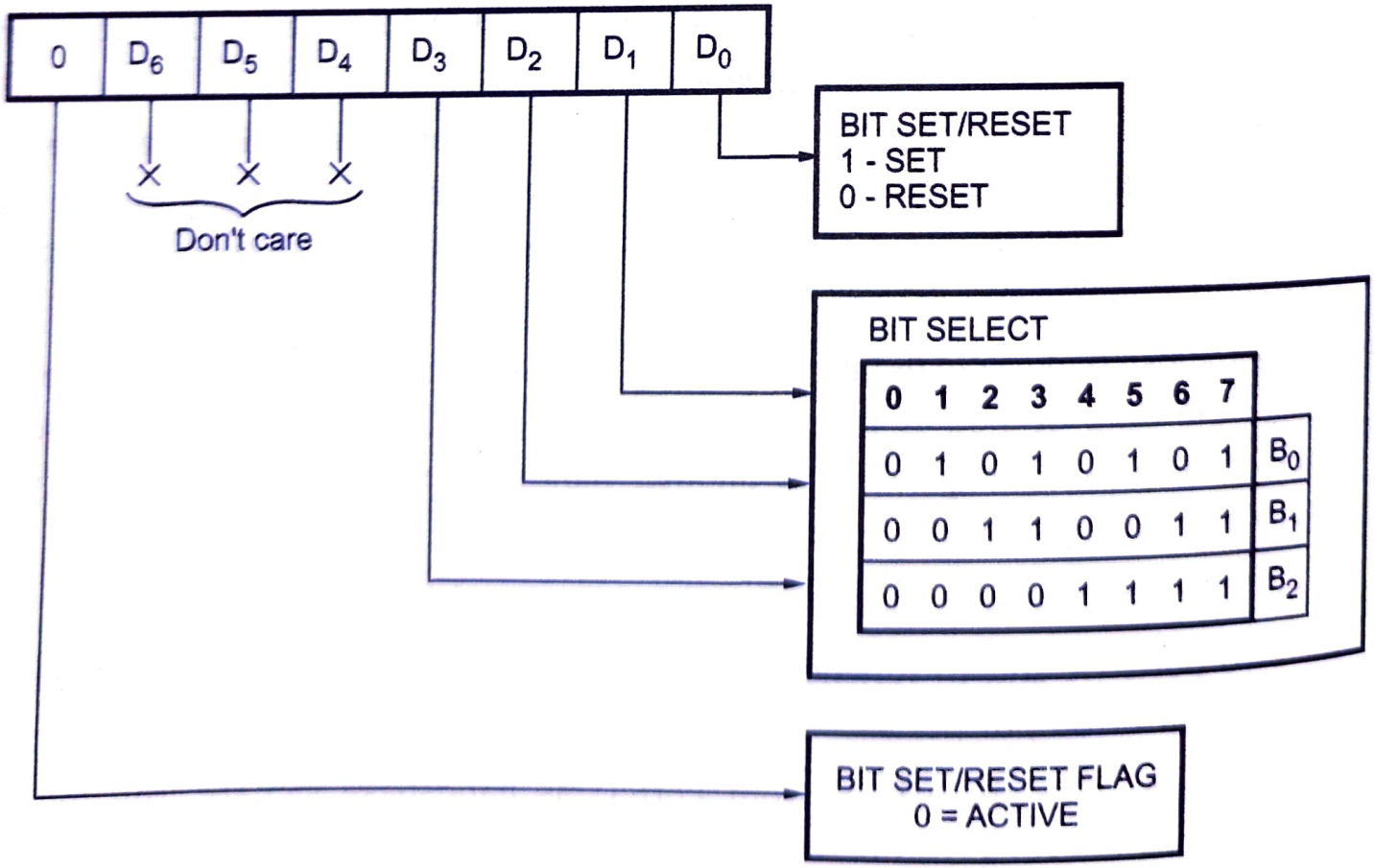



**Fig. 9.2.3 Bit set/reset control word format**

- A BSR word is to be written for each bit that is to be set or reset. For example, if bit $PC_3$ is to be set and bit $PC_4$ is to be reset, the appropriate BSR words that will have to be loaded into the control register will be, 0XXX0111 and 0XXX1000, respectively, where X is don't care.
- The BSR word can also be used for enabling or disabling interrupt signals generated by Port C when the 8255 is programmed for Mode 1 or 2 operation. This is done by setting or resetting the associated bits of the interrupts. This is described in detail in next section.

### 9.2.2.2 For I/O Mode

The mode definition format for I/O mode is shown in Fig. 9.2.4. The control words for both, mode definition and Bit Set-Reset are loaded into the same control register,

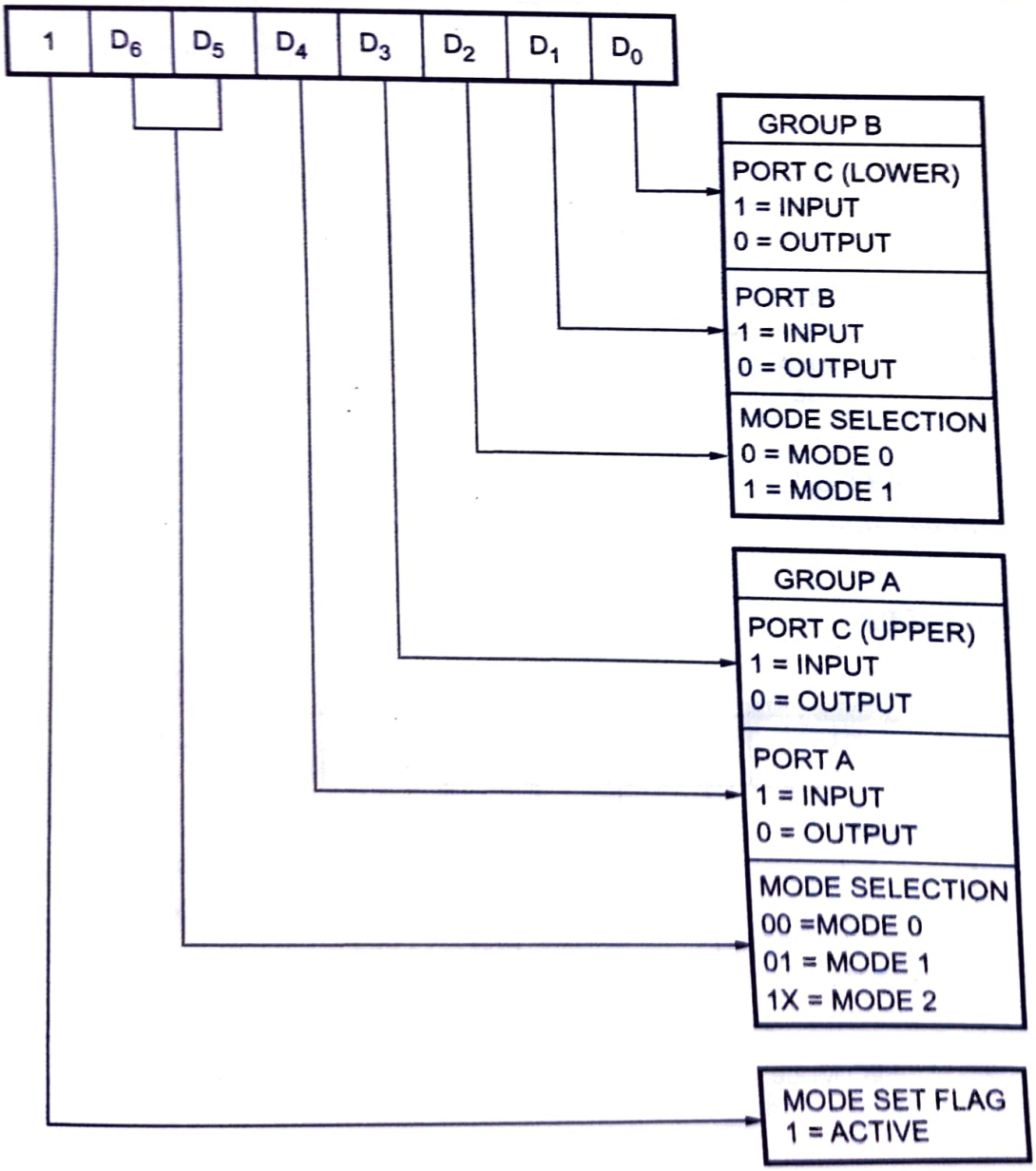



**Fig. 9.2.4 8255 Mode definition format**

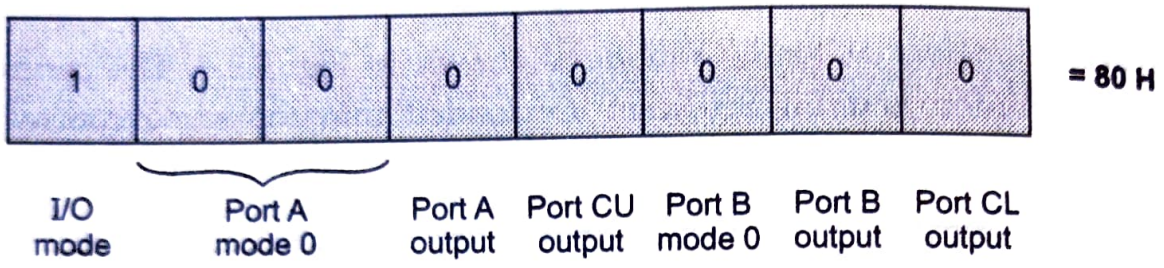
with bit $D_7$ used for specifying whether the word loaded into the control register is a mode definition word or Bit Set-Reset word. If $D_7$ is high, the word is taken as a mode definition word, and if it is low, it is taken as a Bit Set-Reset word. The appropriate bits are set or reset depending on the type of operation desired, and loaded into the control register.

**Example 9.2.1** *Calculate the control word of 8255 for the following cases :*

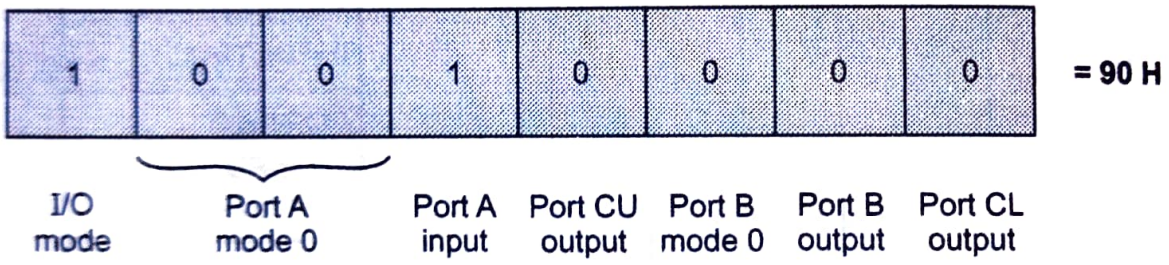
*i) All the ports A, B and C are output ports (mode - 0) ii) PA = in, PB = out, PCL = out = PCH.*

**VTU : June-15, Jan.-18, Marks 5**

**Solution : i)**




**ii)**



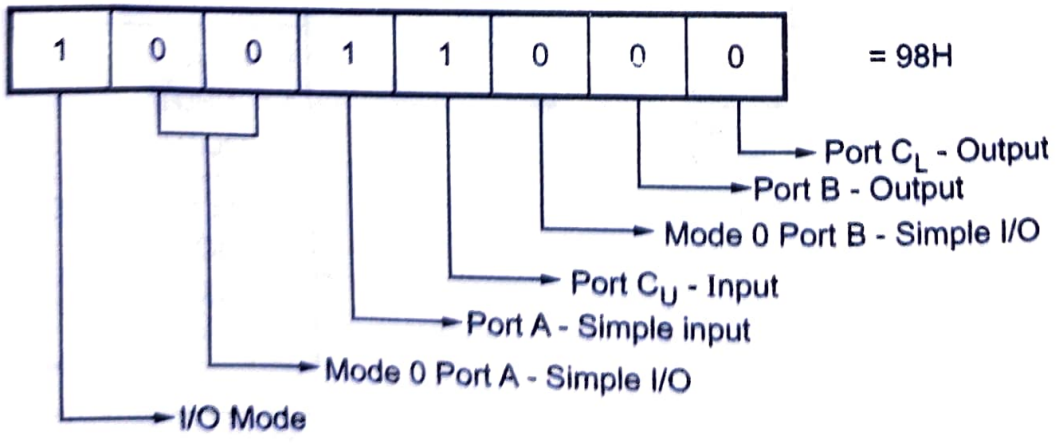

**Example 9.2.2** *Write a program to initialize 8255 in the configuration given below :*

*1. Port A : Simple input 2. Port B : Simple output*

*3. Port $C_L$ : Output 4. Port $C_U$ : Input*

*Assume address of the control word register of 8255 as 8003H.*

**Solution :**

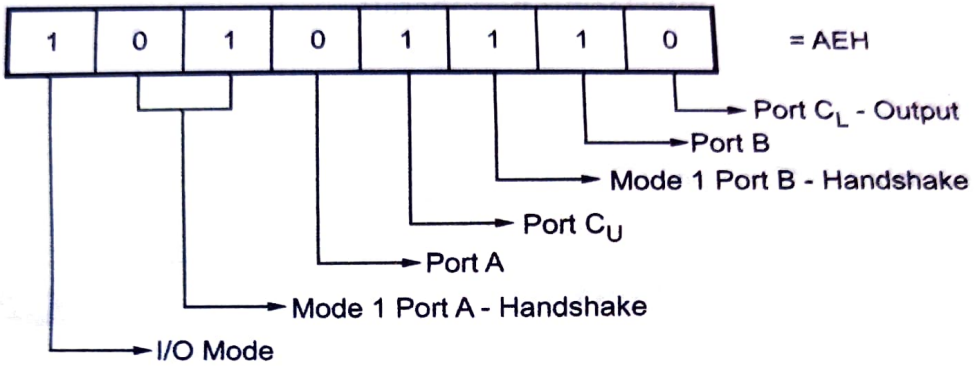


**Example 9.2.3** *Write a program to initialize 8255 in the configuration given below :*

1. *Port A : Output with handshake*
2. *Port B : Input with handshake*
3. *Port $C_L$ : Output*
4. *Port $C_U$ : Input*

*Assume address of the control word register of 8255 as 8003H.*

**Solution :**




**Source program :**

```
MOV A, #0AEH          ; Load control word
MOV DPTR, #8003H
MOVX @DPTR, A         ; Send control word
```

## Review Questions

1. *Explain with a neat diagram, the functional block diagram of 8255.*

   **VTU : Dec.-12, June-13, July-19, Marks 6**

2. *What is meant by BSR mode in 8255 ?*
3. *Explain various I/O modes of 8255.* **VTU : July-16, Marks 6**
4. *Explain the control word for BSR mode in 8255.*
5. *Explain the control word for I/O mode in 8255.*
6. *Briefly show control word of 8255 and specify mode selection.* **VTU : Jan.-15, Marks 6**
7. *Explain the various fields of 8255 control word format. Draw the control word format.*

   **VTU : July-18, Marks 6**

## Review Questions

1. *What is keybounce ? How it is achieved ?*
2. *Explain the matrix keyboard interface of 8051.*
3. *What is key bouncing ? How it is eliminated ?* **VTU : Feb.-09, 4 Marks**
4. *Interface a 4 × 4 matrix keyboard to 8051. Write the required block schematic and the assembly level program.* **VTU : Aug.-06, Feb.-10, 8 Marks**
5. *Interface a 4×4 matrix keyboard to 8051. Write a subroutine to scan the keyboard and send the ASCII code for a valid key closure on Port 0.* **VTU : Aug.-08, 10, 16, Marks**
6. *Show a simple keyboard interface with a port of 8051 and explain its operation.* **VTU : Feb.-09, 6 Marks**
7. *Interface a 2 × 4 keys keyboard to 8051 and write an ALP to send the keycode to port P1, whenever a key is depressed.* **VTU : Dec.-11, Marks 10**

## 8.5 Parallel and Serial ADC Interfacing

**VTU : Feb.-08, 09, Aug.-11, Jan.-18, 19**

### 8.5.1 Interfacing of ADC 0808/0809

- The ADC 0808 and ADC 0809 are monolithic CMOS devices with an 8-channel multiplexer. These devices are also designed to operate from common microprocessor/ microcontroller control buses, with tri-state output latches driving the data bus. The main features of these devices are :

**Features**

- 8-bit successive approximation ADC.
- 8-channel multiplexer with address logic.
- Conversion time 100 μs.
- It eliminates the need for external zero and full-scale adjustments.
- Easy to interface to all microprocessors.
- It operates on single 5 V power supply.
- Output meet TTL voltage level specifications.

**Pin Diagram**

- Fig. 8.5.1 shows pin diagram of 0808/0809 ADC.

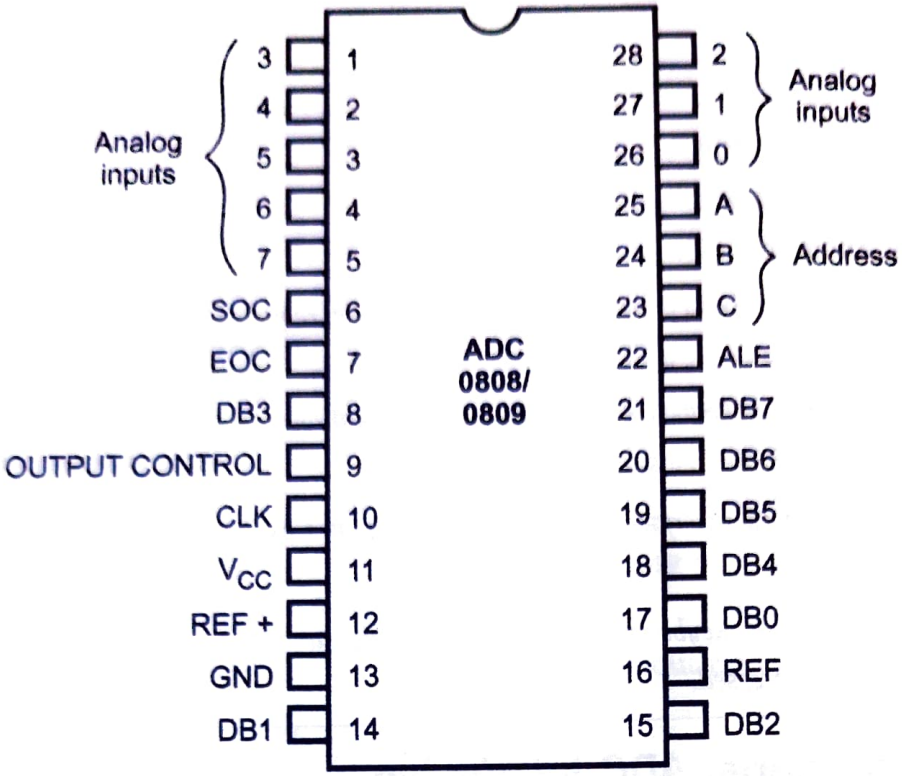



**Fig. 8.5.1 Pin diagram of ADC 0808/0809**

## Operation

- ADC 0808/0809 has eight input channels, so to select desired input channel, it is necessary to send 3-bit address on A, B and C inputs. The address of the desired channel is sent to the multiplexer address inputs through port pins. After at least 50 ns, this address must be latched. This can be achieved by sending ALE signal. After another 2.5 μs, the start of conversion (SOC) signal must be sent high and then low to start the conversion process. To indicate end of conversion ADC 0808/0809 activates EOC signal. The microprocessor system can read converted digital word through data bus by enabling the output enable signal after EOC is activated. This is illustrated in Fig. 8.5.2.

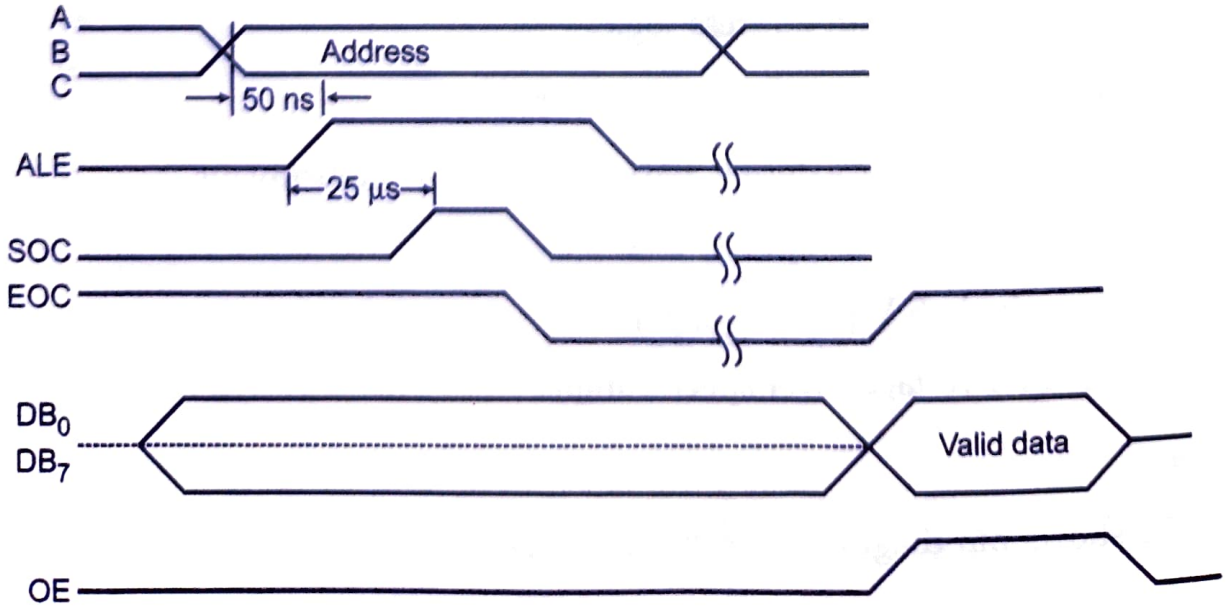



**Fig. 8.5.2 Timing waveforms for ADC 0808**

**Interfacing**

- Fig. 8.5.3 shows typical interfacing circuit for ADC 0808 with microprocessor/ microcontroller system.

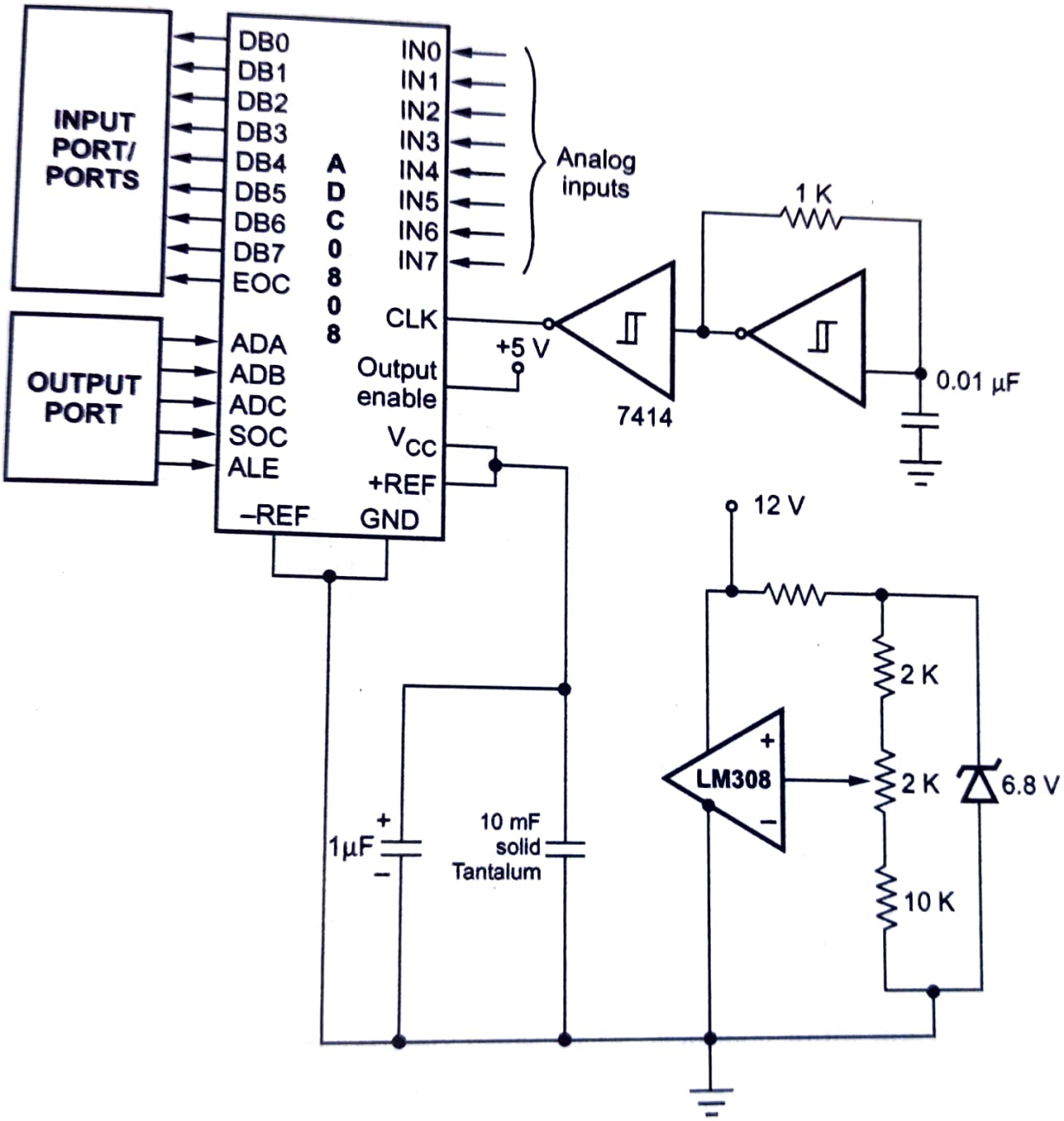



**Fig. 8.5.3 Typical Interface for 0808/0809**

- The zener diode and LM 308 amplifier circuitry is used to produce a $V_{CC}$ and $+V_{REF}$ of 5.12 V for the A/D converter. With this reference voltage the A/D converter will have 256 steps of 20 mV each.