Roll No. | | | | | | | | | | | | **Total No. of Pages : 02**

Total No. of Questions : 09

B.Tech.(ECE)/(ETE) (2011 Onwards)
B.Tech.(Automation & Robotics) (2011 & Onwards)
B.Tech.(Electronics Engg.) (2012 Onwards)
(Sem.–5)

MICROPROCESSORS & MICROCONTROLLERS

Subject Code : BTEC-504
M.Code : 70480

Time : 3 Hrs. **Max. Marks : 60**

INSTRUCTION TO CANDIDATES :

1. **SECTION-A is COMPULSORY consisting of TEN questions carrying TWO marks each.**
2. **SECTION-B contains FIVE questions carrying FIVE marks each and students have to attempt any FOUR questions.**
3. **SECTION-C contains THREE questions carrying TEN marks each and students have to attempt any TWO questions.**

SECTION-A

1. Answer briefly :

(a) What is the function of RST and ALE signals?

(b) What do you understand by Embedded Systems?

(c) Enlist some salient features of 8085.

(d) List the different flags of 8085 microprocessor.

(e) How many ways an 8051 can be interrupted?

(f) Discuss the difference between ADD and ADDC instruction.

(g) If the frequency of the crystal connected to 8085 is 6MHz, calculate the time to fetch and execute NOP instruction.

(h) What is the significance of SWAP instruction?

(i) What is the use of SMOD bit in 8051?

(j) What is the use of DPTR?

SECTION-B

2. Differentiate between memory mapped I/O and peripheral mapped I/O in case of 8085 microprocessor.

3. Draw and explain the timing diagram of memory read cycle.

4. A switch is connected to pin P1.0 and LED to pin P2.7. Write a program to get the status of the switch and send it to the LED. At the same time, generate a waveform of 100 microsecond at pin p3.2.

5. Write a program to generate a square wave of 2 KHz frequency at pin 2.3 of 8051 microcontroller.

6. Discuss the various registers of 8051 microcontroller.

SECTION-C

7. What is the function of SBUF register? Discuss the various steps to transfer data serially.

8. Classify and explain different types of 8085 instructions with examples.

9. Show the connections for Interfacing of DAC with 8051 and write a program to demonstrate its working.

NOTE : Disclosure of Identity by writing Mobile No. or Making of passing request on any page of Answer Sheet will lead to UMC against the Student.

Paper 2 solution MPMC

Section A

(a) What is the function of RST and ALE signals?

ALE (Address Enable Latch) is the control signal which is nothing but a positive going pulse generated when a new operation is started by microprocessor. So when pulse goes high means ALE=1, it makes address bus enable and when ALE=0, means low pulse makes data bus enable. In 8085 Instruction set, RSTn is actually standing for "Restart n". And in this case, n has a value from 0 to 7 only. Thus the eight possible RST instructions are there, e.g. RST 0, RST 1, …, RST 7.

(b) What do you understand by Embedded Systems?

An embedded system can be thought of as a computer hardware system having software embedded in it. An embedded system can be an independent system or it can be a part of a large system. An embedded system is a microcontroller or microprocessor based system which is designed to perform a specific task. For example, a fire alarm is an embedded system; it will sense only smoke.

An embedded system has three components −

- It has hardware.
- It has application software.
- It has Real Time Operating system (RTOS) that supervises the application software and provide mechanism to let the processor run a process as per scheduling by following a plan to control the latencies. RTOS defines the way the system works. It sets the rules during the execution of application program. A small scale embedded system may not have RTOS.

(c) Enlist some salient features of 8085.

- Processor is 8 bit ALU
- 4K Byte ROM
- 128 Byte RAM
- External ROM & RAM both are 64KB
- 2 16 bit Timers
- 4 I/0 PORTS(All Bidirectional & Bit addressable)
- Serial ports(TX & RX)

(d) List the different flags of 8085 microprocessor.

In 8085 microprocessor, the flags register can have a total of eight flags. Thus a flag can be represented by 1 bit of information. But only five flags are implemented in 8085. And they are:

- Carry flag (Cy),
- Auxiliary carry flag (AC),
- Sign flag (S),
- Parity flag (P), and
- Zero flag (Z).

The respective position of these flag bits in flag register has been show the below figure. The positions marked by "x" are to be considered as don't care bits in the flags register. The user is not required to memorize the positions of these flags in the flags register.

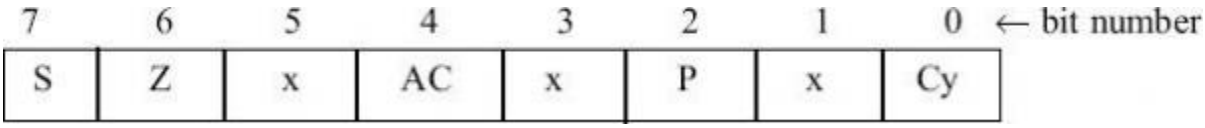


Fig. Flags register

(e) How many ways an 8051 can be interrupted?

8051 has 5 interrupt signals, i.e. INT0, TFO, INT1, TF1, RI/TI. Each interrupt can be enabled or disabled by setting bits of the IE register and the whole interrupt system can be disabled by clearing the EA bit of the same register.

IE (Interrupt Enable) Register

This register is responsible for enabling and disabling the interrupt. EA register is set to one for enabling interrupts and set to 0 for disabling the interrupts. Its bit sequence and their meanings are shown in the following figure.

EA	-	-	ES	ET1	EX1	ET0	EX0

EA	IE.7	It disables all interrupts. When EA = 0 no interrupt will be acknowledged and EA = 1 enables the interrupt individually.
-	IE.6	Reserved for future use.

-	IE.5	Reserved for future use.
ES	IE.4	Enables/disables serial port interrupt.
ET1	IE.3	Enables/disables timer1 overflow interrupt.
EX1	IE.2	Enables/disables external interrupt1.
ET0	IE.1	Enables/disables timer0 overflow interrupt.
EX0	IE.0	Enables/disables external interrupt0.

IP (Interrupt Priority) Register

We can change the priority levels of the interrupts by changing the corresponding bit in the Interrupt Priority (IP) register as shown in the following figure.

- A low priority interrupt can only be interrupted by the high priority interrupt, but not interrupted by another low priority interrupt.
- If two interrupts of different priority levels are received simultaneously, the request of higher priority level is served.
- If the requests of the same priority levels are received simultaneously, then the internal polling sequence determines which request is to be serviced.

-	-	PT2	PS	PT1	PX1	PT0	PX0
bit7	bit6	bit5	bit4	bit3	bit2	bit1	

-	IP.6	Reserved for future use.
-	IP.5	Reserved for future use.
PS	IP.4	It defines the serial port interrupt priority level.

PT1	IP.3	It defines the timer interrupt of 1 priority.
PX1	IP.2	It defines the external interrupt priority level.
PT0	IP.1	It defines the timer0 interrupt priority level.
PX0	IP.0	It defines the external interrupt of 0 priority level.

(f) Discuss the difference between ADD and ADDC instruction.

ADD and ADDC both add the value operand to the value of the Accumulator, leaving the resulting value in the Accumulator. ... ADD and ADDC function identically except that ADDC adds the value of operand as well as the value of the Carry flag whereas ADD does not add the Carry flag to the result.

(g) If the frequency of the crystal connected to 8085 is 6MHz, calculate the time to fetch and execute NOP instruction.

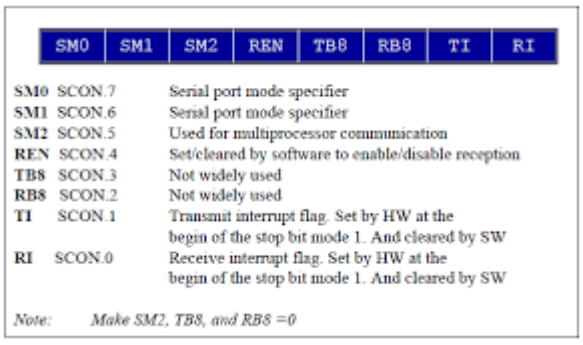
If the 8085 MPU is working on 6MHz clock frequency, then the internal clock frequency is 3MHz. So from that we can easily determine that each clock period is 1/3 of a microsecond. So the NOP will be executed in 1/3 * 4 = 1.333µs

(h) What is the significance of SWAP instruction?

The SWAP instruction exchanges the low-order and high-order nibbles within the accumulator. No flags are affected by this instruction.

(i) What is the use of SMOD bit in 8051?

The SMOD Bit in the PCON Register is used to control the Baud Rate of the Serial Port. There are two general purpose Flag Bits in the PCON Register, which can be used by the programmer during execution. The following table describes the function of each bit in the PCON Register



SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SCON.7	Serial port mode specifier
SM1	SCON.6	Serial port mode specifier
SM2	SCON.5	Used for multiprocessor communication
REN	SCON.4	Set/cleared by software to enable/disable reception
TB8	SCON.3	Not widely used
RB8	SCON.2	Not widely used
TI	SCON.1	Transmit interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW
RI	SCON.0	Receive interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW

Note: Make SM2, TB8, and RB8 =0

(j) What is the use of DPTR?

Data Pointer

¬ The 16-bit Data Pointer (DPTR) register is the concatenation of registers DPH (data pointer's high-order byte) and DPL (data pointer's low-order byte).

¬ The DPTR is used to access the external data memory using register indirect addressing mode.

SECTION B

2. Differentiate between memory mapped I/O and peripheral mapped I/O in case of 8085 microprocessor.

In Memory Mapped Input Output −

- We allocate a memory address to an Input-Output device.
- Any instructions related to memory can be accessed by this Input-Output device.
- The Input-Output device data are also given to the Arithmetic Logical Unit.

Input-Output Mapped Input Output −

- We give an Input-Output address to an Input-Output device
- Only IN and OUT instructions are accessed by such devices.
- The ALU operations are not directly applicable to such Input-Output data.

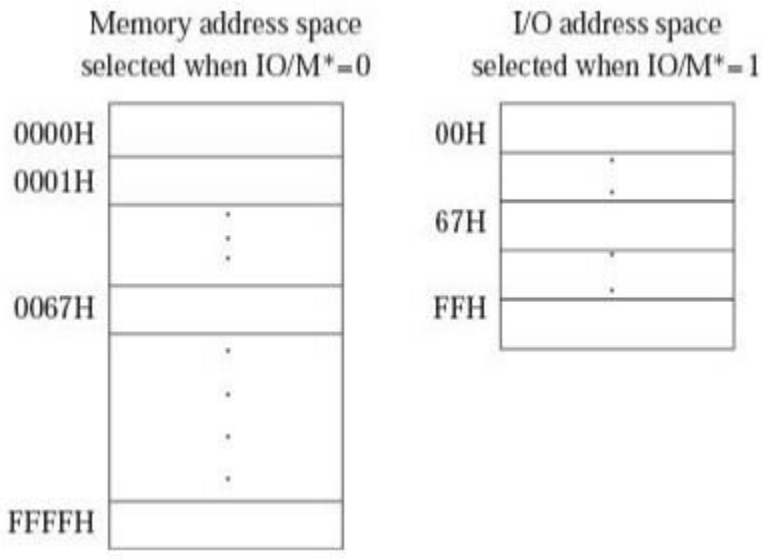
So as a summary we can mention that −

- I/O is any general-purpose port used by processor/controller to handle peripherals connected to it.
- I/O mapped I/Os have a separate address space from the memory. So, total addressed capacity is the number of I/Os connected and a memory connected. Separate I/O-related instructions are used to access I/Os. A separate signal is used for addressing an I/O device.
- Memory-mapped I/Os share the memory space with external memory. So, total addressed capacity is memory connected only. This is underutilisation of resources if your processor supports I/O-mapped I/O. In this case, instructions used to access I/Os are the same as that used for memory.
- Let's take an example of the 8085 processor. It has 16 address lines i.e. addressing capacity of 64 KB memory. It supports I/O-mapped I/Os. It can address up to 256 I/Os.
- If we connect I/Os to it an I/O-mapped I/O then, it can address 256 I/Os + 64 KB memory. And special instructions IN and OUT are used to access the peripherals. Here we fully utilize the addressing capacity of the processor.

- If the peripherals are connected in memory mapped fashion, then total devices it can address is only 64K. This is underutilisation of the resource. And only memory-accessing instructions like MVI, MOV, LOAD, SAVE are used to access the I/O devices.

After the discussion stated earlier, it is not possible for us to conclude to which scheme of addressing the Input Output ports is better. Both of them have their advantages and disadvantages. The Intel family of microprocessors like 8085, 8086, 80386, Pentium, and Zilog family of microprocessors like Z-80, Z-8000, etc. provide I/O-mapped I/O facility, in addition to providing memory-mapped I/O. So some I/O ports can be connected as I/O-mapped I/O ports, and some others as memory-mapped I/O ports in an Intel processor-based system. But Motorola family of microprocessors like 6800, 68000, 68020, etc. provide only memory-mapped I/O. Thus, we can say that an Intel processor is better compared with a Motorola processor, as far as addressing of I/O ports is concerned.

After the discussions stated earlier it is not possible for us to conclude to the extent of which scheme of addressing the Input-Output port is better. Both of them have their advantages and disadvantages. The family of microprocessors which belong to the Intel family-like 8085, 8086, 80386, Pentium, and Zilog family of microprocessors like Z-80, Z-8000, etc provide the facility of Input-Output mapping to Input-Output facility. Where in addition, we provide the facility of memory mapped Input Output also In the processor-based system manufactured by Intel. But the family of microprocessors which belong to Motorolla like 6800, 68000, 68020 provides only Memory mapped Input Output. Hence we can conclude that the processor which belong to Intel family is far better compared with the processor of Motorolla as far the addressing of the Input-Output ports is concerned.



3. Draw and explain the timing diagram of memory read cycle.

1. Instruction cycle: this term is defined as the number of steps required by the cpu to complete the entire process ie. Fetching and execution of one instruction. The fetch and execute cycles are carried out in synchronization with the clock.

2. Machine cycle: It is the time required by the microprocessor to complete the operation of accessing the memory devices or I/O devices. In machine cycle various operations like opcode fetch, memory read, memory write, I/O read, I/O write are performed.

3. T-state: Each clock cycle is called as T-states.

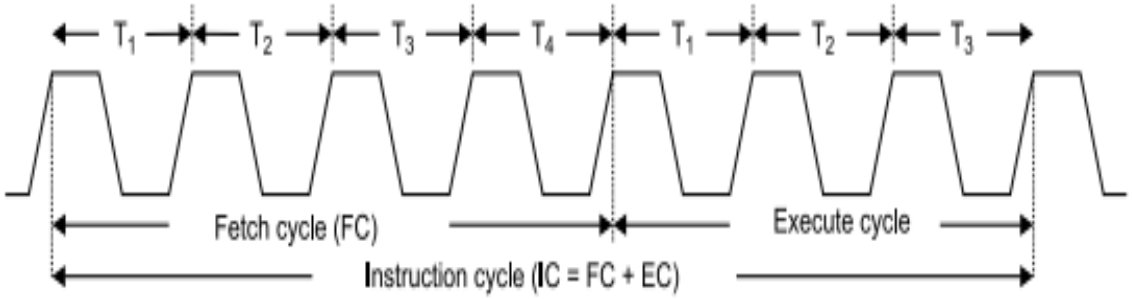
Rules to identify number of machine cycles in an instruction:
1. If an addressing mode is direct, immediate or implicit then No. of machine cycles = No. of bytes.

2. If the addressing mode is indirect then No. of machine cycles = No. of bytes + 1. Add +1 to the No. of machine cycles if it is memory read/write operation.

3. If the operand is 8-bit or 16-bit address then, No. of machine cycles = No. of bytes +1.

4. These rules are applicable to 80% of the instructions of 8085.

Timing Diagram:



Where, Instruction cycle= Fetch Cycle(FC) + Executecycle(EC).

Opcode fetch:

- The microprocessor requires instructions to perform any particular action. In order to perform these actions microprocessor utilizes Opcode which is a part of an instruction which provides detail(ie. Which operation µp needs to perform) to microprocessor.

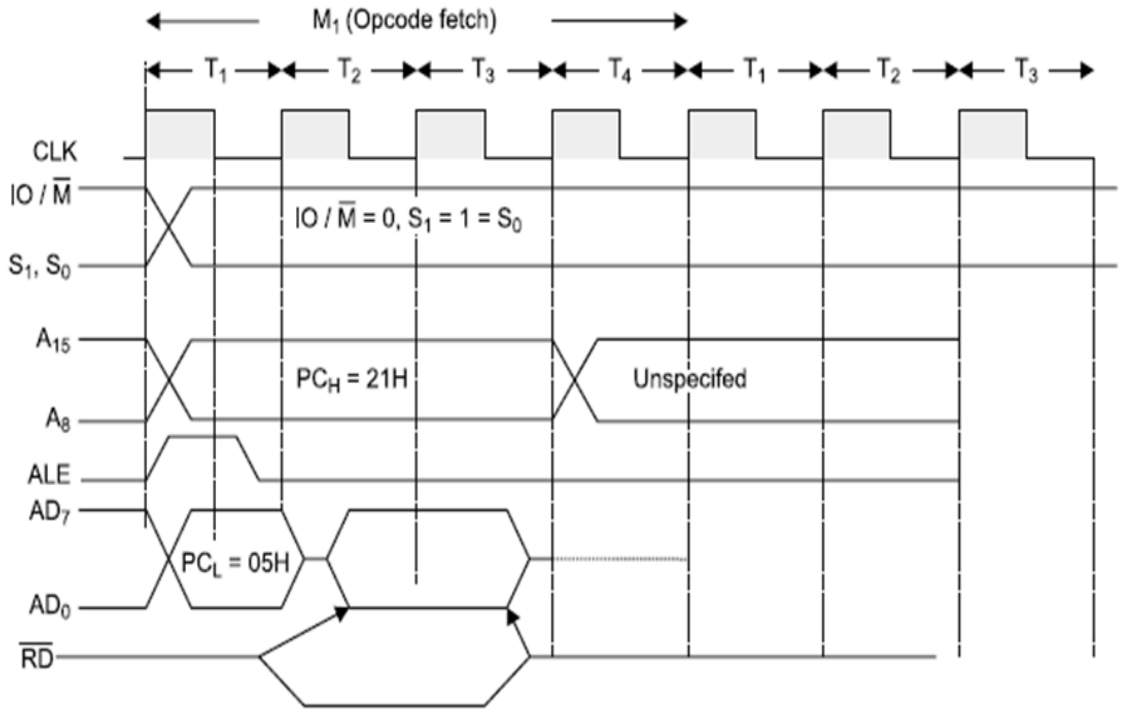


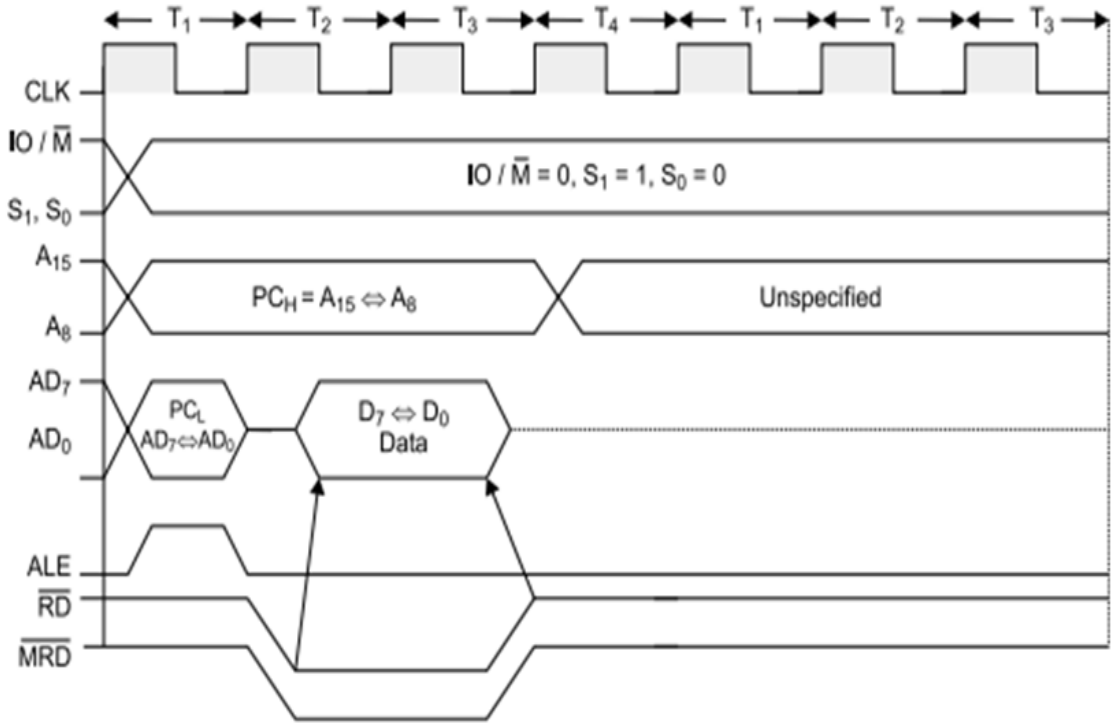
Fig: Opcode fetch timing diagram

Operation:

- During T1 state, microprocessor uses IO/M(bar), S0, S1 signals are used to instruct microprocessor to fetch opcode.
- Thus when IO/M(bar)=0, S0=S1= 1, it indicates opcode fetch operation.
- During this operation 8085 transmits 16-bit address and also uses ALE signal for address latching.
- At T2 state microprocessor uses read signal and make data ready from that memory location to read opcode from memory and at the same time program counter increments by 1 and points next instruction to be fetched.
- In this state microprocessor also checks READY input signal, if this pin is at low logic level ie. '0' then microprocessor adds wait state immediately between T2 and T3.
- At T3, microprocessor reads opcode and store it into instruction register to decode it further.
- During T4 microprocessor performs internal operation like decoding opcode and providing necessary actions.
- The opcode is decoded to know whether T5 or T6 states are required, if they are not required then µp performs next operation.

Read and write timing diagram for memory and I/O Operation

Memory
Read:



<>Figure: Memory read timing diagram

Operation:

- It is used to fetch one byte from the memory.
- It requires 3 T-States.
- It can be used to fetch operand or data from the memory.
- During T1, A8-A15 contains higher byte of address. At the same time ALE is high. Therefore Lower byte of address A0-A7 is selected from AD0-AD7.
- Since it is memory ready operation, IO/M(bar) goes low.

- During T2 ALE goes low, RD(bar) goes low. Address is removed from AD0-AD7 and data D0-D7 appears on AD0-AD7.

- During T3, Data remains on AD0-AD7 till RD(bar) is at low signal.

4. A switch is connected to pin P1.0 and LED to pin P2.7. Write a program to get the status of the switch and send it to the LED. At the same time, generate a waveform of 100 microsecond at pin p3.2.

```
#include<reg51.h>
sbit Led = P2^7; //pin connected to toggle Led
sbit Switch =P1^1; //Pin connected to toggle led
int main()
{
Led = 0; //configuring as output pin
Switch = 1; //Configuring as input pin
while(1) //Continuous monitor the status of the switch.
{
if(Switch == 0)
{
Led =1; //Led On
}
else
{
Led =0; //Led Off
}
}
return 0;
}
```

5. Write a program to generate a square wave of 2 KHz frequency at pin 2.3 of 8051 microcontroller.

Look at the following steps.

1. T — 1 / 50 Hz = 20 ms, the period of the square wave.
2. 1/2 of it for the high and low portions of the pulse = 10 ms
3. 10 ms / 1.085 us = 9216 and 65536 – 9216 = 56320 in decimal, and in hex it is
4. DCOOH.
5. TL = 00 and TH = DC (hex)

The program follows.

```
        MOV   TMOD,#10H       ;Timer 1, mode 1 (16-bit)
AGAIN:  MOV   TL1,#00         ;TL1 = 00, Low byte
        MOV   TH1,#0DCH       ;TH1 = DCH, High byte
        SETB  TR1             ;start Timer 1
BACK:   JNB   TF1,BACK        ;stay until timer rolls over
        CLR   TR1             ;stop Timer 1
        CPL   P2.3            ;comp. P2.3 to get hi, lo
        CLR   TF1             ;clear Timer 1 flag
        SJMP  AGAIN           ;reload timer since mode 1
                              ;is not auto-reload
```

6. Discuss the various registers of 8051 microcontroller.

- Accumulator
- R register
- B register
- Data Pointer (DPTR)
- Program Counter (PC)
- Stack Pointer (SP)

Accumulator

The accumulator, register A, is used for all arithmetic and logic operations. If the accumulator is not present, then every result of each calculation (addition, multiplication, shift, etc.) is to be stored into the main memory. Access to main memory is slower than access to a register like the accumulator because the technology used for the large main memory is slower (but cheaper) than that used for a register.

The "R" Registers

The "R" registers are a set of eight registers, namely, R0, R1 to R7. These registers function as auxiliary or temporary storage registers in many operations. Consider an example of the sum of 10 and 20. Store a variable 10 in an accumulator and another variable 20 in, say, register R4. To process the addition operation, execute the following command −

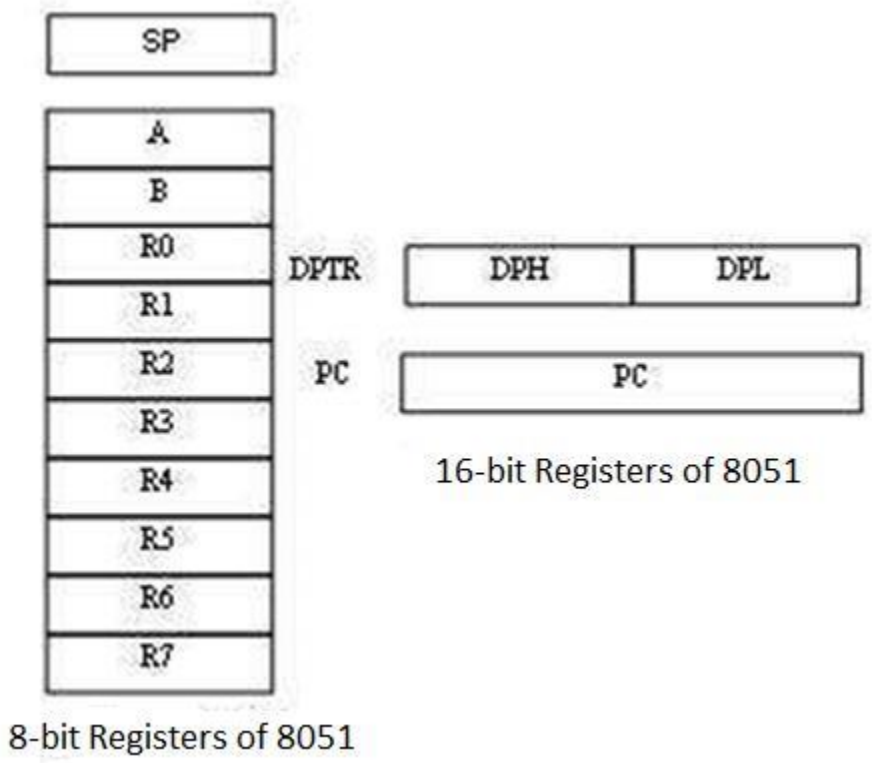
```
ADD A,R4
```

After executing this instruction, the accumulator will contain the value 30. Thus "R" registers are very important auxiliary or helper registers. The Accumulator alone would not be very useful if it were not for these "R" registers. The "R" registers are meant for temporarily storage of values.

Let us take another example. We will add the values in R1 and R2 together and then subtract the values of R3 and R4 from the result.

```
MOV A,R3   ;Move the value of R3 into the accumulator
ADD A,R4   ;Add the value of R4
MOV R5,A   ;Store the resulting value temporarily in R5
MOV A,R1   ;Move the value of R1 into the accumulator
ADD A,R2   ;Add the value of R2
SUBB A,R5  ;Subtract the value of R5 (which now contains R3 + R4)
```

As you can see, we used R5 to temporarily hold the sum of R3 and R4. Of course, this is not the most efficient way to calculate (R1 + R2) – (R3 + R4), but it does illustrate the use of the "R" registers as a way to store values temporarily.



The "B" Register

The "B" register is very similar to the Accumulator in the sense that it may hold an 8-bit (1-byte) value. The "B" register is used only by two 8051 instructions: MUL AB and DIV AB. To quickly and easily multiply or divide A by another number, you may store the other number in "B" and make use of these two instructions. Apart from using MUL and DIV instructions, the "B" register is often used as yet another temporary storage register, much like a ninth R register.

The Data Pointer

The Data Pointer (DPTR) is the 8051's only user-accessible 16-bit (2-byte) register. The Accumulator, R0–R7 registers and B register are 1-byte value registers. DPTR is meant for

pointing to data. It is used by the 8051 to access external memory using the address indicated by DPTR. DPTR is the only 16-bit register available and is often used to store 2-byte values.

The Program Counter

The Program Counter (PC) is a 2-byte address which tells the 8051 where the next instruction to execute can be found in the memory. PC starts at 0000h when the 8051 initializes and is incremented every time after an instruction is executed. PC is not always incremented by 1. Some instructions may require 2 or 3 bytes; in such cases, the PC will be incremented by 2 or 3.

Branch, jump, and interrupt operations load the Program Counter with an address other than the next sequential location. Activating a power-on reset will cause all values in the register to be lost. It means the value of the PC is 0 upon reset, forcing the CPU to fetch the first opcode from the ROM location 0000. It means we must place the first byte of upcode in ROM location 0000 because that is where the CPU expects to find the first instruction.

The Stack Pointer (SP)

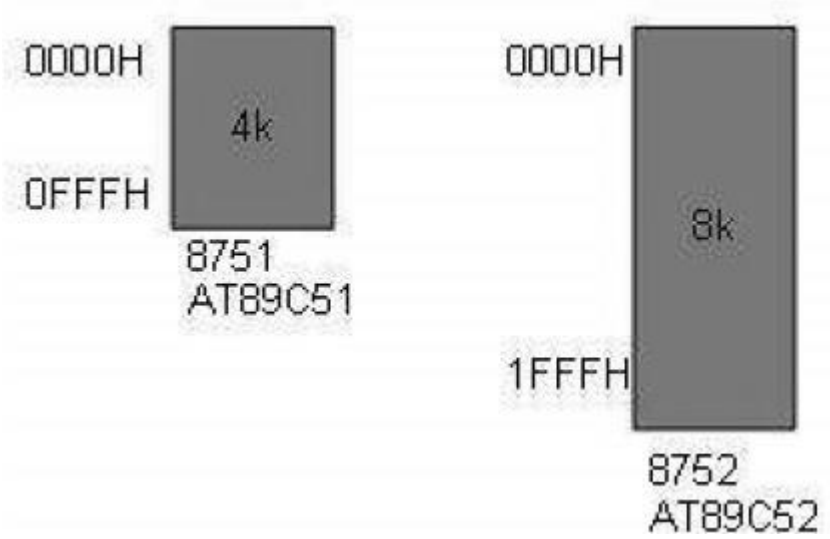
The Stack Pointer, like all registers except DPTR and PC, may hold an 8-bit (1-byte) value. The Stack Pointer tells the location from where the next value is to be removed from the stack. When a value is pushed onto the stack, the value of SP is incremented and then the value is stored at the resulting memory location. When a value is popped off the stack, the value is returned from the memory location indicated by SP, and then the value of SP is decremented.

This order of operation is important. SP will be initialized to 07h when the 8051 is initialized. If a value is pushed onto the stack at the same time, the value will be stored in the internal RAM address 08h because the 8051 will first increment the value of SP (from 07h to 08h) and then will store the pushed value at that memory address (08h). SP is modified directly by the 8051 by six instructions: PUSH, POP, ACALL, LCALL, RET, and RETI.

ROM Space in 8051

Some family members of 8051 have only 4K bytes of on-chip ROM (e.g. 8751, AT8951); some have 8K ROM like AT89C52, and there are some family members with 32K bytes and 64K bytes of on-chip ROM such as Dallas Semiconductor. The point to remember is that no member of the 8051 family can access more than 64K bytes of opcode since the program counter in 8051 is a 16-bit register (0000 to FFFF address).

The first location of the program ROM inside the 8051 has the address of 0000H, whereas the last location can be different depending on the size of the ROM on the chip. Among the 8051 family members, AT8951 has $k bytes of on-chip ROM having a memory address of 0000 (first location) to 0FFFH (last location).



8051 Flag Bits and PSW Register

The program status word (PSW) register is an 8-bit register, also known as flag register. It is of 8-bit wide but only 6-bit of it is used. The two unused bits are user-defined flags. Four of the flags are called conditional flags, which means that they indicate a condition which results after an instruction is executed. These four are CY (Carry), AC (auxiliary carry), P (parity), and OV (overflow). The bits RS0 and RS1 are used to change the bank registers. The following figure shows the program status word register.

The PSW Register contains that status bits that reflect the current status of the CPU.

CY	CA	F0	RS1	RS0	OV	-	P

CY	PSW.7	Carry Flag
AC	PSW.6	Auxiliary Carry Flag
F0	PSW.5	Flag 0 available to user for general purpose.
RS1	PSW.4	Register Bank selector bit 1
RS0	PSW.3	Register Bank selector bit 0

OV	PSW.2	Overflow Flag
-	PSW.1	User definable FLAG
P	PSW.0	Parity FLAG. Set/ cleared by hardware during instruction cycle to indicate even/odd number of 1 bit in accumulator.

We can select the corresponding Register Bank bit using RS0 and RS1 bits.

RS1	RS2	Register Bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

- CY, the carry flag − This carry flag is set (1) whenever there is a carry out from the D7 bit. It is affected after an 8-bit addition or subtraction operation. It can also be reset to 1 or 0 directly by an instruction such as "SETB C" and "CLR C" where "SETB" stands for set bit carry and "CLR" stands for clear carry.
- AC, auxiliary carry flag − If there is a carry from D3 and D4 during an ADD or SUB operation, the AC bit is set; otherwise, it is cleared. It is used for the instruction to perform binary coded decimal arithmetic.
- P, the parity flag − The parity flag represents the number of 1's in the accumulator register only. If the A register contains odd number of 1's, then P = 1; and for even number of 1's, P = 0.
- OV, the overflow 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. It is used only to detect errors in signed arithmetic operations.

SECTION C

7. What is the function of SBUF register? Discuss the various steps to transfer data serially.

8. Its an 8 bit register used solely for serial communication in 8051. For a byte of data to be transferred via TxD line, it muse be placed in SBUF register. Similarly SBUF register holds the serially inputed data received by TxD line of 8051.

Accessed as:

```
MOV SBUF,#'S' ;load ascii

MOV SBUF,A  ;

MOV A, SBUF  ;
```

The moment the byte is written in SBUF, it is framed with start and stop bits and transferred serially through TxD line.

Similarly when bits are received serially via RxD it is deframed by eliminating stop and start bits and a byte of data is received and placed in SBUF.

Programming Serial Data Transmitting

In programming the 8051 to transfer character bytes serially

1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate
2. The TH1 is loaded with one of the values to set baud rate for serial data transfer
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits
4. TR1 is set to 1 to start timer 1
5. TI is cleared by CLR TI instruction
6. The character byte to be transferred serially is written into SBUF register
7. The TI flag bit is monitored with the use of instruction JNB TI,xx to see if the character has been transferred completely
8. To transfer the next byte, go to step 5

Q8. Classify and explain different types of 8085 instructions with examples.

1 Data Transfer Croup

The data transfer instructions move data between registers or between memory and registers.

MOV	Move
MVI	Move Immediate
LDA	Load Accumulator Directly from Memory
STA	Store Accumulator Directly in Memory
LHLD	Load H & L Registers Directly from Memory
SHLD	Store H & L Registers Directly in Memory

An 'X' in the name of a data transfer instruction implies that it deals with a register pair (16-bits);

LXI	Load Register Pair with Immediate data
LDAX	Load Accumulator from Address in Register Pair
STAX	Store Accumulator in Address in Register Pair
XCHG	Exchange H & L with D & E
XTHL	Exchange Top of Stack with H & L

2 Arithmetic Group

The arithmetic instructions add, subtract, increment, or decrement data in registers or memory.

ADD	Add to Accumulator
ADI	Add Immediate Data to Accumulator
ADC	Add to Accumulator Using Carry Flag
ACI	Add immediate data to Accumulator Using Carry

SUB	Subtract from Accumulator
SUI	Subtract Immediate Data from Accumulator
SBB	Subtract from Accumulator Using Borrow (Carry) Flag
SBI	Subtract Immediate from Accumulator Using Borrow (Carry) Flag
INR	Increment Specified Byte by One
DCR	Decrement Specified Byte by One
INX	Increment Register Pair by One
DCX	Decrement Register Pair by One
DAD	Double Register Add; Add Content of Register

Pair to H & L Register Pair

3 Logical Group

This group performs logical (Boolean) operations on data in registers and memory and on condition flags. The logical AND, OR, and Exclusive OR instructions enable you to set specific bits in

the accumulator ON or OFF.

ANA	Logical AND with Accumulator
ANI	Logical AND with Accumulator Using Immediate Data
ORA	Logical OR with Accumulator
OR	Logical OR with Accumulator Using Immediate Data
XRA	Exclusive Logical OR with Accumulator
XRI	Exclusive OR Using Immediate Data

The Compare instructions compare the content of an 8-bit value with the contents of the accumulator;

CMP Compare

CPI Compare Using Immediate Data

The rotate instructions shift the contents of the accumulator one bit position to the left or right:

RLC Rotate Accumulator Left

RRC Rotate Accumulator Right

RAL Rotate Left Through Carry

RAR Rotate Right Through Carry

Complement and carry flag instructions:

CMA Complement Accumulator

CMC Complement Carry Flag

STC Set Carry Flag

4 Branch Group

The branching instructions alter normal sequential program flow, either unconditionally or conditionally. The unconditional branching instructions are as follows:

JMP Jump

CALL Call

RET Return

Conditional branching instructions examine the status of one of four condition flags to determine whether the specified branch is to be executed. The conditions that may be specified are as follows:

NZ	Not Zero (Z = 0)
Z	Zero (Z = 1)
NC	No Carry (C = 0)
C	Carry (C = 1)
PO	Parity Odd (P = 0)
PE	Parity Even (P = 1)
P	Plus (S = 0)
M	Minus (S = 1)

Thus, the conditional branching instructions are specified as follows:

Jumps	Calls	Returns
INC	CNC	RNC (No Carry)
JNZ	CNZ	RNZ (Not Zero)
JM	CM	RM (Minus)
JP0	CPO	RPO (Parity Odd)
JM	CM	RM (Minus)
JPE	CPE	RPE (Parity Even)
JP0	CPO	RPO (Parity Odd)

Two other instructions can affect a branch by replacing the contents or the program counter:

PCHL	Move H & L to Program Counter
RST	Special Restart Instruction Used with Interrupts

5 Stack Instructions

The following instructions affect the Stack and/or Stack Pointer

PUSH	Push Two bytes of Data onto the Stack

POP	Pop Two Bytes of Data off the Stack
XTHL	Exchange Top of Stack with H & L
SPHL	Move content of H & L to Stack Pointer

6 I/0 instructions

IN	Initiate Input Operation
OUT	Initiate Output Operation

7 Machine Control instructions

EI	Enable Interrupt System
DI	Disable Interrupt System
HLT	Halt
NOP	No Operation

9. Show the connections for Interfacing of DAC with 8051 and write a program to demonstrate its working

The Digital to Analog converter (DAC) is a device, that is widely used for converting digital pulses to analog signals. There are two methods of converting digital signals to analog signals. These two methods are binary weighted method and R/2R ladder method. In this article we will use the MC1408 (DAC0808) Digital to Analog Converter. This chip uses R/2R ladder method. This method can achieve a much higher degree of precision. DACs are judged by its resolution. The resolution is a function of the number of binary inputs. The most common input counts are 8, 10, 12 etc. Number of data inputs decides the resolution of DAC. So if there are n digital input pin, there are 2^n analog levels. So 8 input DAC has 256 discrete voltage levels.

The MC1408 DAC (or DAC0808)

In this chip the digital inputs are converted to current. The output current is known as I_{out} by connecting a resistor to the output to convert into voltage. The total current provided by the I_{out} pin is basically a function of the binary numbers at the input pins D_0 - D_7 (D_0 is the LSB

and D_7 is the MSB) of DAC0808 and the reference current I_{ref}. The following formula is showing the function of I_{out}

IOut=Iref{D72+D64+D58+D416+D332+D264+D1128+D0256IOut=Iref{D72+D64+D58+ D416+D332+D264+D1128+D0256

The I_{ref} is the input current. This must be provided into the pin 14. Generally 2.0mA is used as I_{ref}

We connect the I_{out} pin to the resistor to convert the current to voltage. But in real life it may cause inaccuracy since the input resistance of the load will also affect the output voltage. So practically I_{ref} current input is isolated by connecting it to an Op-Amp with $R_f = 5K\Omega$ as feedback resistor. The feedback resistor value can be changed as per requirement.

Generating Sinewave using DAC and 8051 Microcontroller

For generating sinewave, at first we need a look-up table to represent the magnitude of the sine value of angles between 0° to 360°. The sine function varies from -1 to +1. In the table only integer values are applicable for DAC input. In this example we will consider 30° increments and calculate the values from degree to DAC input. We are assuming full-scale voltage of 10V for DAC output. We can follow this formula to get the voltage ranges.

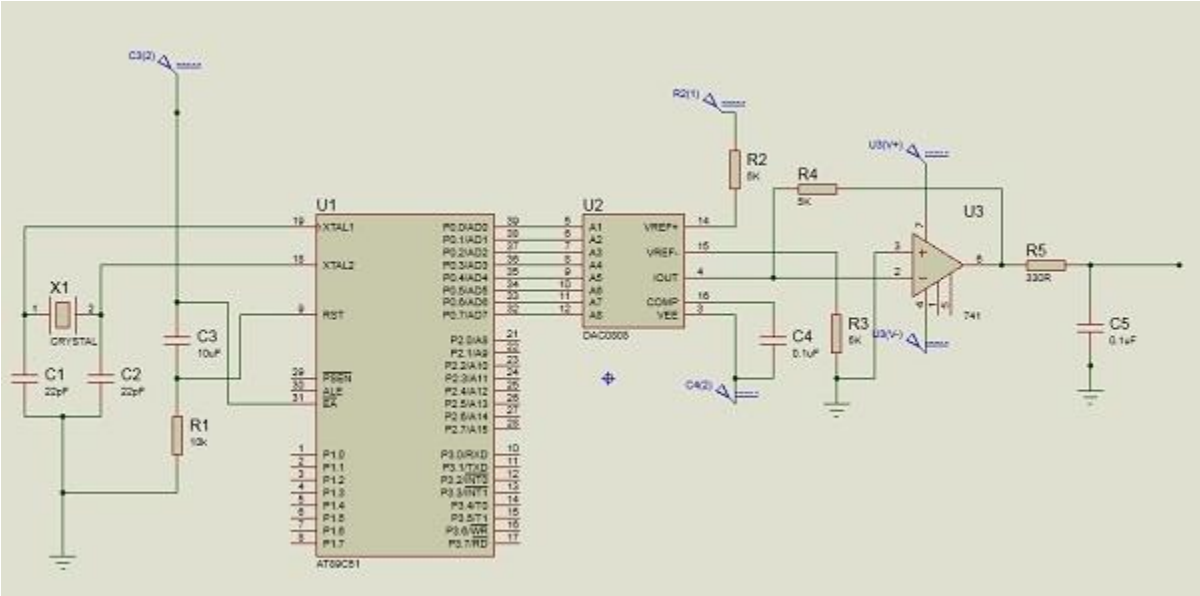
$$V_{out} = 5V + (5 \times \sin\theta)$$

Let us see the lookup table according to the angle and other parameters for DAC.

Angle(in θ)	sinθ	V_{out} (Voltage Magnitude)	Values sent to DAC
0	0	5	128
30	0.5	7.5	192
60	0.866	9.33	238
90	1.0	10	255
120	0.866	9.33	238

Angle(in θ)	sinθ	V_{out} (Voltage Magnitude)	Values sent to DAC
150	0.5	7.5	192
180	0	5	128
210	-0.5	2.5	64
240	-0.866	0.669	17
270	-1.0	0	0
300	-0.866	0.669	17
330	-0.5	2.5	64
360	0	5	128

10. Circuit Diagram −



Source Code

```
#include<reg51.h>

sfr DAC = 0x80; //Port P0 address

void main(){

  int sin_value[12] = {128,192,238,255,238,192,128,64,17,0,17,64};

  int i;

  while(1){

    //infinite loop for LED blinking

    for(i = 0; i<12; i++){

      DAC = sin_value[i];

    }

  }

}
```