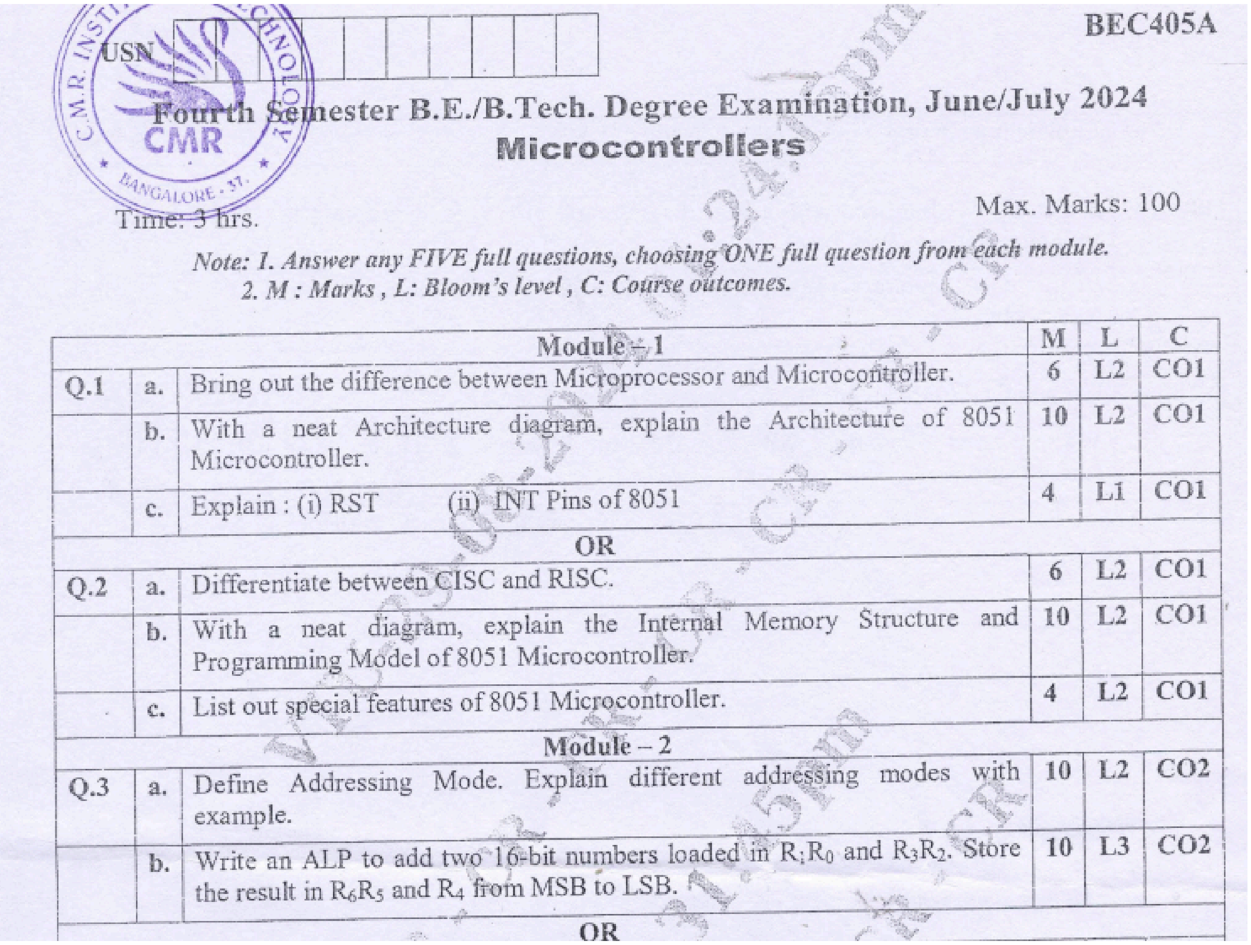
VTU Question Paper Solution



USN □□□□□□□□□□

BEC405A

Fourth Semester B.E./B.Tech. Degree Examination, June/July 2024

Microcontrollers

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	Bring out the difference between Microprocessor and Microcontroller.	6	L2	CO1
	b.	With a neat Architecture diagram, explain the Architecture of 8051 Microcontroller.	10	L2	CO1
	c.	Explain : (i) RST (ii) INT Pins of 8051	4	L1	CO1
		OR			
Q.2	a.	Differentiate between CISC and RISC.	6	L2	CO1
	b.	With a neat diagram, explain the Internal Memory Structure and Programming Model of 8051 Microcontroller.	10	L2	CO1
	c.	List out special features of 8051 Microcontroller.	4	L2	CO1
		Module – 2			
Q.3	a.	Define Addressing Mode. Explain different addressing modes with example.	10	L2	CO2
	b.	Write an ALP to add two 16-bit numbers loaded in R_1R_0 and R_3R_2. Store the result in R_6R_5 and R_4 from MSB to LSB.	10	L3	CO2
		OR			

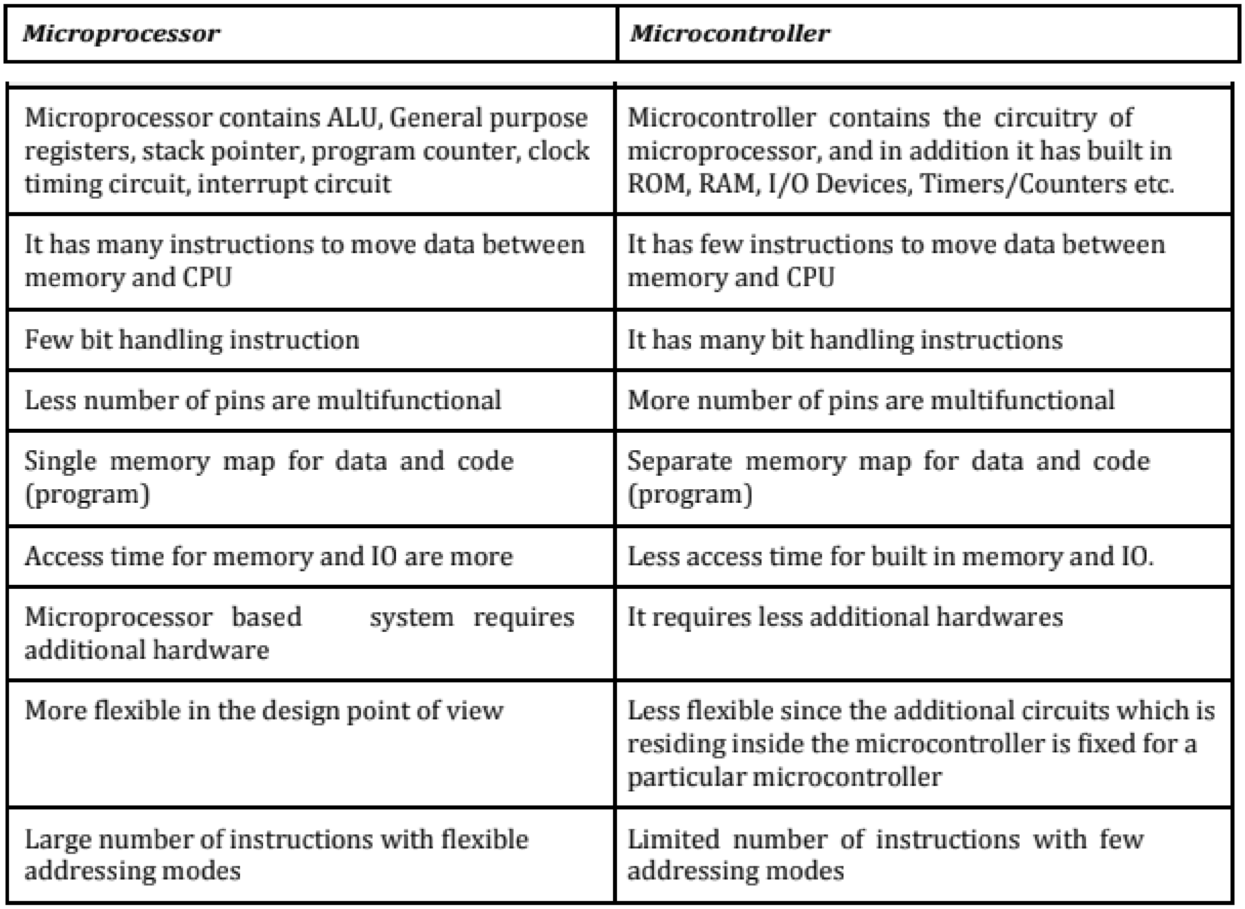
Q.4	a.	Define Stack. Explain the operation of Stack using Stack Pointer, PUSH and POP Instructions.	10	L2	CO2
	b.	Write an ALP to find largest of N numbers.	10	L3	CO2
		Module – 3			
Q.5	a.	Explain : (i) TMOD (ii) TCON register of 8051.	10	L2	CO3
	b.	Assume XTAL = 22 MHz. Write an ALP to generate a square of frequency 1 kHz on Pin P1.2.	10	L2	CO3
		OR			
Q.6	a.	Explain : (i) SCON register (ii) Importance of TI Flag	10	L2	CO3
	b.	Write a C program to transfer "YES" serially at 9600 baud rate, 8 bit data, 1 stop bit, do this continuously.	10	L3	CO3
		Module – 4			
Q.7	a.	Define Interrupt. List the steps involved in Executing an Interrupt.	10	L2	CO4
	b.	Explain Interrupt Vector table of 8051 Microcontroller.	5	L2	CO4
	c.	Explain Interrupt enable register.	5	L2	CO4
		OR			
Q.8	a.	Explain Interrupt Control used in 8051.	10	L2	CO4
	b.	Explain the steps involved in programming serial communication Interrupt.	5	L2	CO4

	c.	Explain how multiple Interrupts are handled in 8051.	5	L2	CO4
		Module – 5			
Q.9	a.	Explain DAC Interface with a neat diagram and also write a program to generate staircase waveform.	10	L3	CO5
	b.	With a neat diagram, write a program to Interface Stepper Motor to 8051 Microcontroller.	10	L3	CO5
		OR			
Q.10	a.	Explain the Interfacing of DC motor using C programming.	10	L3	CO5
	b.	With a neat diagram, write a ALP to Interface LCD to 8051 Microcontroller.	10	L3	CO5

* * * * *

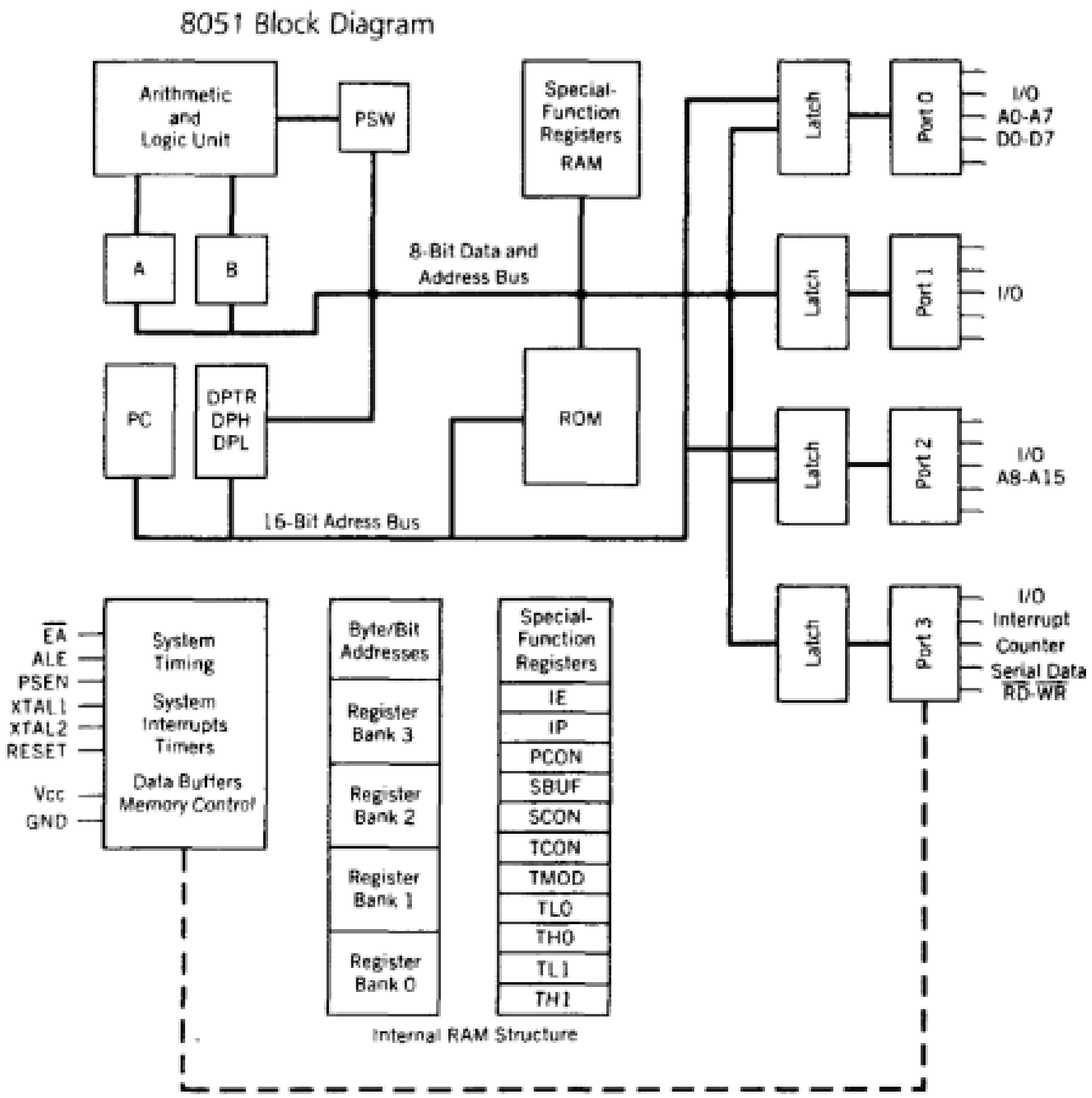
Solution

1. a.Bring out the difference between Microprocessor and Microcontroller.



Microprocessor	*Microcontroller*
Microprocessor contains ALU, General purpose registers, stack pointer, program counter, clock timing circuit, interrupt circuit	Microcontroller contains the circuitry of microprocessor, and in addition it has built in ROM, RAM, I/O Devices, Timers/Counters etc.
It has many instructions to move data between memory and CPU	It has few instructions to move data between memory and CPU
Few bit handling instruction	It has many bit handling instructions
Less number of pins are multifunctional	More number of pins are multifunctional
Single memory map for data and code (program)	Separate memory map for data and code (program)
Access time for memory and IO are more	Less access time for built in memory and IO.
Microprocessor based system requires additional hardware	It requires less additional hardwares
More flexible in the design point of view	Less flexible since the additional circuits which is residing inside the microcontroller is fixed for a particular microcontroller
Large number of instructions with flexible addressing modes	Limited number of instructions with few addressing modes

b. With a neat Architecture diagram explain the architecture of 8051 Microcontrollers.


8051 Block Diagram

Introduction

Salient features of 8051 microcontroller are given below.

- Eight bit CPU

- On chip clock oscillator

- 4Kbytes of internal program memory (code memory) *[ROM]*

- 128 bytes of internal data memory *[RAM]*

- 64 Kbytes of external program memory address space.

- 64 Kbytes of external data memory address space.
- 32 bi directional I/O lines (can be used as four 8 bit ports or 32 individually addressable I/O lines)
- Two 16 Bit Timer/Counter :T0, T1
- Full Duplex serial data receiver/transmitter
- Four Register banks with 8 registers in each bank.
- Sixteen bit Program counter (PC) and a data pointer (DPTR)
- 8 Bit Program Status Word (PSW)
- 8 Bit Stack Pointer
- Five vector interrupt structure (RESET not considered as an interrupt.)
- 8051 CPU consists of 8 bit ALU with associated registers like accumulator ‘A’ , B register, PSW, SP, 16 bit program counter, stack pointer.
- ALU can perform arithmetic and logic functions on 8 bit variables.
- 8051 has 128 bytes of internal RAM which is divided into
    - Working registers [00 – 1F]
    - Bit addressable memory area [20 – 2F]
    - General purpose memory area (Scratch pad memory) [30-7F]

8051 has 4 K Bytes of internal ROM. The address space is from 0000 to 0FFFh. If the program size is more than 4 K Bytes 8051 will fetch the code automatically from external memory.

- Accumulator is an 8 bit register widely used for all arithmetic and logical operations.

  Accumulator is also used to transfer data between external memory. B register is used along with Accumulator for multiplication and division. A and B registers together is also called MATH registers.
- PSW (Program Status Word). This is an 8 bit register which contains the arithmetic status of ALU and the bank select bits of register banks.

CY	AC	F0	RS1	RS0	OV		-P

CY - carry flag
AC - auxiliary carry flag
F0 - available to the user for general purpose

RS1,RS0 - register bank select bits
OV - overflow
P - parity

- Stack Pointer (SP) – it contains the address of the data item on the top of the stack.

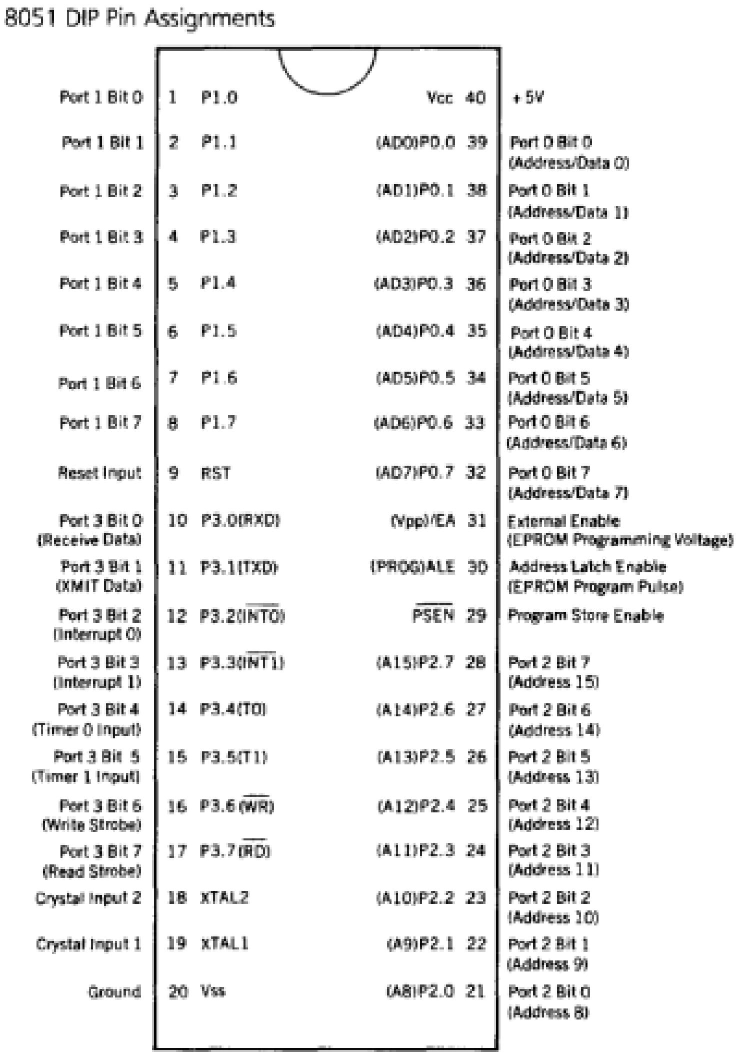
  Stack may reside anywhere on the internal RAM. On reset, SP is initialized to 07 so that the default stack will start from address 08 onwards.

- Data Pointer (DPTR) – DPH (Data pointer higher byte), DPL (Data pointer lower byte).

  This is a 16 bit register which is used to furnish address information for internal and external program memory and for external data memory.

- Program Counter (PC) – 16 bit PC contains the address of next instruction to be executed. On reset PC will set to 0000. After fetching every instruction PC will increment by one.

C. Explain (i) RST (ii) INT Pins of 8051



8051 DIP Pin Assignments

Port 1 Bit 0 1 P1.0 Vcc 40 +5V
Port 1 Bit 1 2 P1.1 (AD0)P0.0 39 Port 0 Bit 0 (Address/Data 0)
Port 1 Bit 2 3 P1.2 (AD1)P0.1 38 Port 0 Bit 1 (Address/Data 1)
Port 1 Bit 3 4 P1.3 (AD2)P0.2 37 Port 0 Bit 2 (Address/Data 2)
Port 1 Bit 4 5 P1.4 (AD3)P0.3 36 Port 0 Bit 3 (Address/Data 3)
Port 1 Bit 5 6 P1.5 (AD4)P0.4 35 Port 0 Bit 4 (Address/Data 4)
Port 1 Bit 6 7 P1.6 (AD5)P0.5 34 Port 0 Bit 5 (Address/Data 5)
Port 1 Bit 7 8 P1.7 (AD6)P0.6 33 Port 0 Bit 6 (Address/Data 6)
Reset Input 9 RST (AD7)P0.7 32 Port 0 Bit 7 (Address/Data 7)
Port 3 Bit 0 (Receive Data) 10 P3.0(RXD) (Vpp)/EA 31 External Enable (EPROM Programming Voltage)
Port 3 Bit 1 (XMIT Data) 11 P3.1(TXD) (PROG)ALE 30 Address Latch Enable (EPROM Program Pulse)
Port 3 Bit 2 (Interrupt 0) 12 P3.2(INT0) PSEN 29 Program Store Enable
Port 3 Bit 3 (Interrupt 1) 13 P3.3(INT1) (A15)P2.7 28 Port 2 Bit 7 (Address 15)
Port 3 Bit 4 (Timer 0 Input) 14 P3.4(T0) (A14)P2.6 27 Port 2 Bit 6 (Address 14)
Port 3 Bit 5 (Timer 1 Input) 15 P3.5(T1) (A13)P2.5 26 Port 2 Bit 5 (Address 13)
Port 3 Bit 6 (Write Strobe) 16 P3.6(WR) (A12)P2.4 25 Port 2 Bit 4 (Address 12)
Port 3 Bit 7 (Read Strobe) 17 P3.7(RD) (A11)P2.3 24 Port 2 Bit 3 (Address 11)
Crystal Input 2 18 XTAL2 (A10)P2.2 23 Port 2 Bit 2 (Address 10)
Crystal Input 1 19 XTAL1 (A9)P2.1 22 Port 2 Bit 1 (Address 9)
Ground 20 Vss (A8)P2.0 21 Port 2 Bit 0 (Address 8)

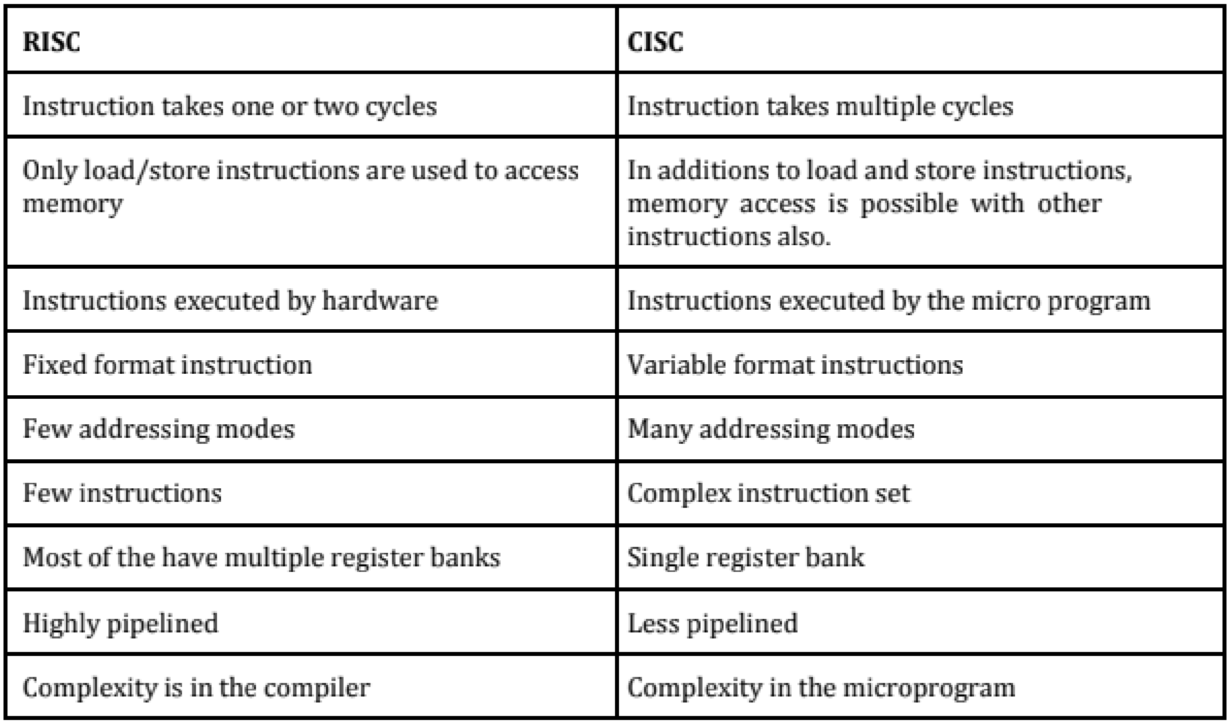
1. RST (Reset) Pin:

- **Pin Number**: 9
- **Function**: Used to reset the microcontroller and set it to a known initial state.
- **Operation**: When a high logic level (typically 1) is applied to the RST pin for at least two machine cycles, the 8051's internal registers and Special Function Registers (SFRs) are set to their default values.
- **Common Uses**: Initializing the microcontroller before starting a new operation, resetting the device in the event of a malfunction, or during power-on.

2. INT (Interrupt) Pins:

- **Pins**: There are two external interrupt pins, **INT0** (Pin 12) and **INT1** (Pin 13).
- **Function**: Used to handle external interrupt requests, allowing the microcontroller to stop its current task to service an urgent event.
- **Interrupt Vector Addresses**:
  - **INT0**: Address 0003H
  - **INT1**: Address 0013H
- **Types of Interrupts**:
  - **Level Triggered**: Activated when held low.
  - **Edge Triggered**: Activated on the falling edge of a pulse.
- **Common Uses**: Handling time-sensitive events, monitoring sensors, and other peripheral devices that need quick responses.

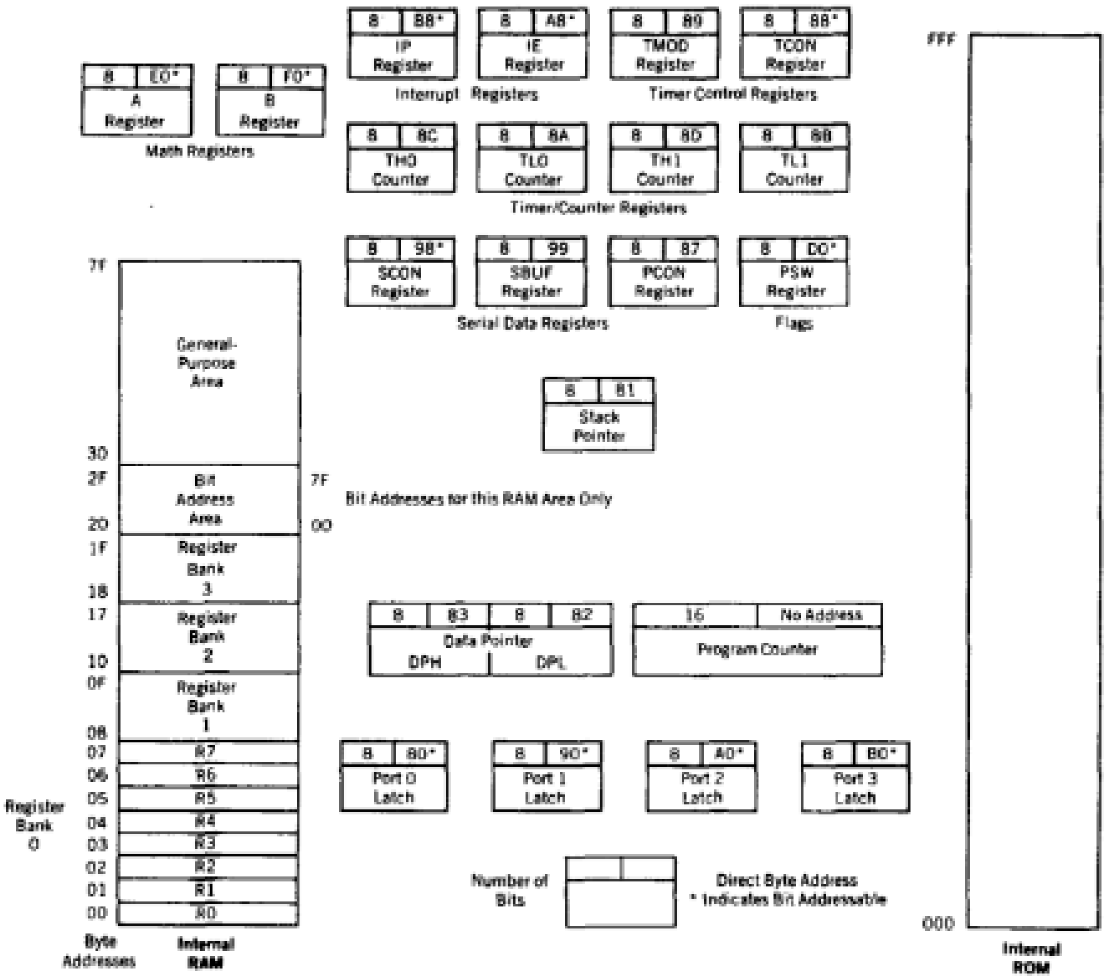
Q2. a. Differentiate between CISC and RISC



RISC	**CISC**
Instruction takes one or two cycles	Instruction takes multiple cycles
Only load/store instructions are used to access memory	In additions to load and store instructions, memory access is possible with other instructions also.
Instructions executed by hardware	Instructions executed by the micro program
Fixed format instruction	Variable format instructions
Few addressing modes	Many addressing modes
Few instructions	Complex instruction set
Most of the have multiple register banks	Single register bank
Highly pipelined	Less pipelined
Complexity is in the compiler	Complexity in the microprogram

b. With a neat diagram, explain the internal Memory Structure and programming model of 8051.

8051 Programming Model



Register Banks: 00h to 1Fh. The 8051 uses 8 general-purpose registers R0 through R7 (R0, R1, R2, R3, R4, R5, R6, and R7). There are four such register banks. Selection of register bank can be done through RS1,RS0 bits of PSW. On reset, the default Register Bank 0 will be selected.

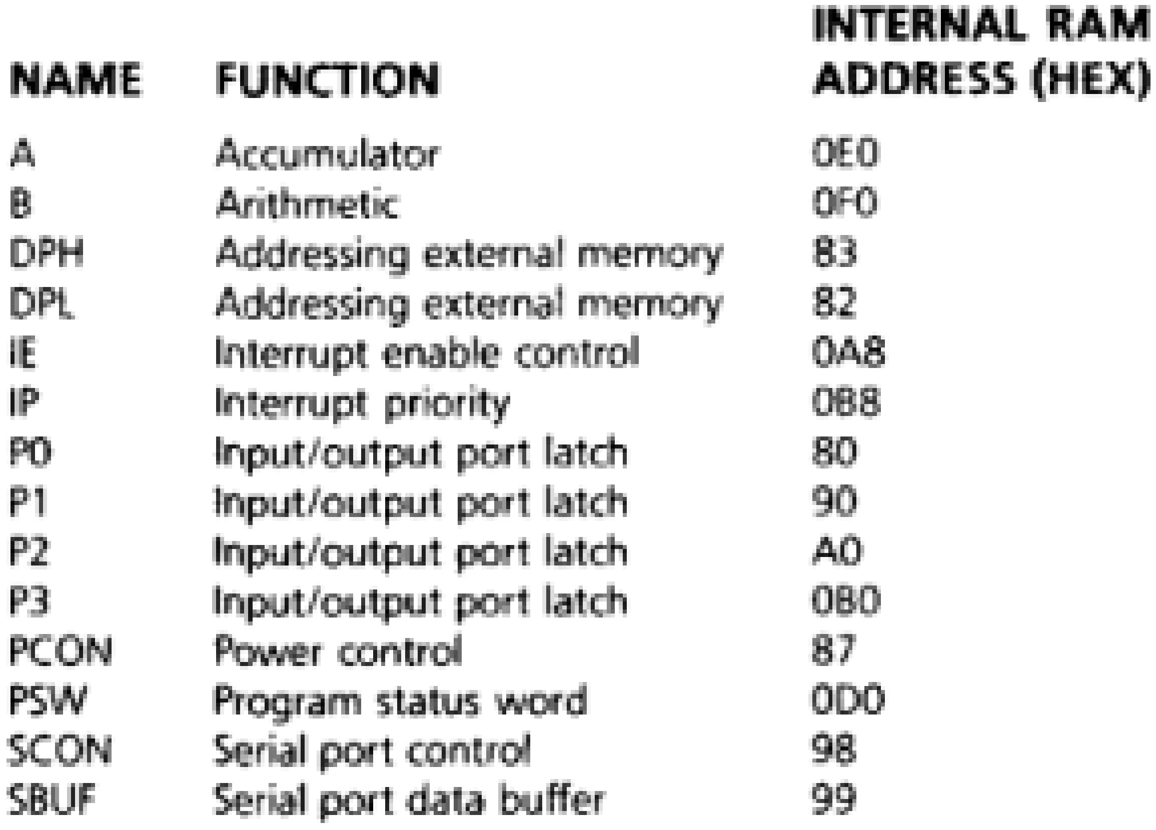
Bit Addressable RAM: 20h to 2Fh . The 8051 supports a special feature which allows access to bit variables. This is where individual memory bits in Internal RAM can be set or cleared. In all there are 128 bits numbered 00h to 7Fh. Being bit variables any one variable can have a value 0 or 1. A bit variable can be set with a command such as SETB and cleared with a command such as CLR. Example instructions are:
SETB 25h ; sets the bit 25h (becomes 1)
CLR 25h ; clears bit 25h (becomes 0)
Note, bit 25h is actually bit 5 of Internal RAM location 24h.
The Bit Addressable area of the RAM is just 16 bytes of Internal RAM located between 20h and

2Fh.

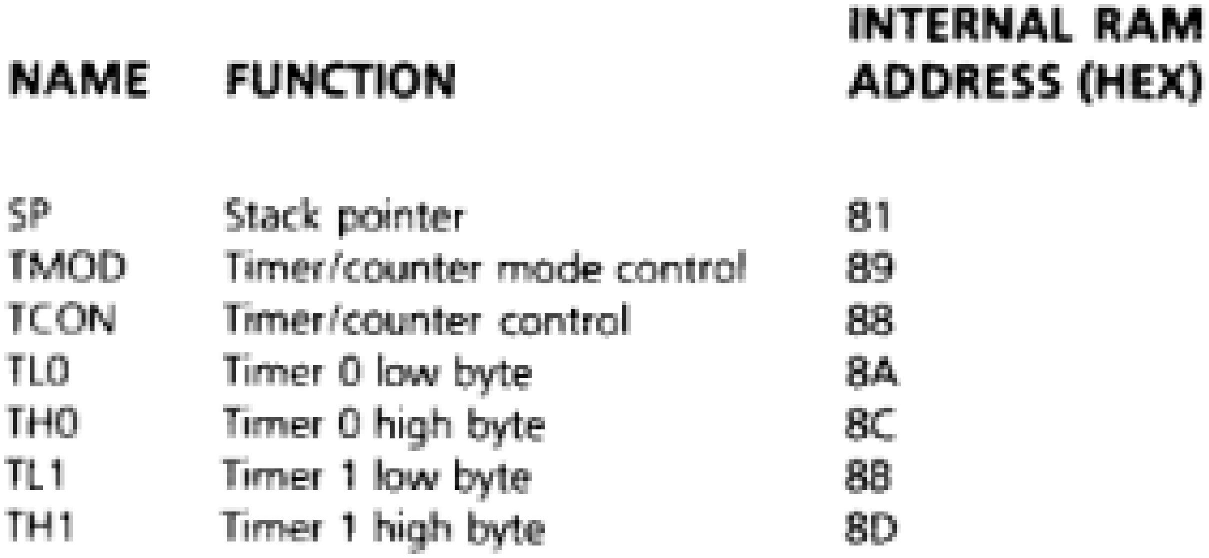
General Purpose RAM: 30h to 7Fh. Even if 80 bytes of Internal RAM memory are available for general-purpose data storage, user should take care while using the memory location from 00 -2Fh
since these locations are also the default register space, stack space, and bit addressable space. It is a good practice to use general purpose memory from 30 – 7Fh. The general purpose RAM can be accessed using direct or indirect addressing modes.

C. List out the special features of 8051 Microcontoller.

Special Function Registers



NAME	FUNCTION	INTERNAL RAM ADDRESS (HEX)
A	Accumulator	0E0
B	Arithmetic	0F0
DPH	Addressing external memory	83
DPL	Addressing external memory	82
IE	Interrupt enable control	0A8
IP	Interrupt priority	0B8
P0	Input/output port latch	80
P1	Input/output port latch	90
P2	Input/output port latch	A0
P3	Input/output port latch	0B0
PCON	Power control	87
PSW	Program status word	0D0
SCON	Serial port control	98
SBUF	Serial port data buffer	99



NAME	FUNCTION	INTERNAL RAM ADDRESS (HEX)
SP	Stack pointer	81
TMOD	Timer/counter mode control	89
TCON	Timer/counter control	88
TL0	Timer 0 low byte	8A
TH0	Timer 0 high byte	8C
TL1	Timer 1 low byte	8B
TH1	Timer 1 high byte	8D

Q.3. a. Define Addressing Mode. Explain different addressing modes with example.

2.2 ADDRESSING MODES

Various methods of accessing the data are called addressing modes.

8051 addressing modes are classified as follows.

1. Immediate addressing.
2. Register addressing.
3. Direct addressing.
4. Indirect addressing.
5. Relative addressing.
6. Absolute addressing.
7. Long addressing.
8. Indexed addressing.
9. Bit inherent addressing.
10. Bit direct addressing.

1. Immediate addressing.

In this addressing mode the data is provided as a part of instruction itself. In other words data immediately follows the instruction.

Eg. MOV A,#30H

ADD A, #83 # Symbol indicates the data is immediate.

2. Register addressing.

In this addressing mode the register will hold the data. One of the eight general registers (R0 to R7) can be used and specified as the operand.

```
Eg. MOV A,R0
    ADD A,R6
```

R0 – R7 will be selected from the current selection of register bank. The default register bank will be bank 0. ***3. Direct addressing***

There are two ways to access the internal memory. Using direct address and indirect address. Using direct addressing mode we can not only address the internal memory but SFRs also. In direct addressing, an 8 bit internal data memory address is specified as part of the instruction and hence, it can specify the address only in the range of 00H to FFH. In this addressing mode, data is obtained directly from the memory. Eg. MOV A,60h

```
    ADD A,30h
```

4. Indirect addressing

The indirect addressing mode uses a register to hold the actual address that will be used in data movement. Registers R0 and R1 and DPTR are the only registers that can be used as data pointers. Indirect addressing cannot be used to refer to SFR registers. Both R0 and R1 can hold 8 bit address and DPTR can hold 16 bit address.

```
Eg. MOV A,@R0
    ADD A,@R1
    MOVX A,@DPTR
```

5. Indexed addressing.

In indexed addressing, either the program counter (PC), or the data pointer (DTPR)—is used to hold the base address, and the A is used to hold the offset address. Adding the value of the base address to the value of the offset address forms the effective address. Indexed addressing is used with JMP or MOVC instructions. Look up tables are easily implemented with the help of index addressing.

Eg. MOVC A, @A+DPTR // *copies the contents of memory location pointed by the sum of the accumulator A and the DPTR into accumulator A.*

MOVC A, @A+PC // *copies the contents of memory location pointed by the sum of the accumulator A and the program counter into accumulator A.*

6. Relative Addressing.

Relative addressing is used only with conditional jump instructions. The relative address, (offset), is an 8 bit signed number, which is automatically added to the PC to make the address of the next instruction. The 8 bit signed offset value gives an address range of +127 to —128 locations. The jump destination is usually specified using a label and the assembler calculates the jump offset accordingly. The advantage of relative addressing is that the program code is easy to relocate and the address is relative to position in the memory.

```
Eg. SJMP LOOP1
    JC BACK
```

7. Absolute addressing

Absolute addressing is used only by the AJMP (Absolute Jump) and ACALL (Absolute Call) instructions. These are 2 bytes instructions. The absolute addressing mode specifies the lowest 11 bit of the memory address as part of the instruction. The upper 5 bit of the destination address are

the upper 5 bit of the current program counter. Hence, absolute addressing allows branching only within the current 2 Kbyte page of the program memory.

Eg. AJMP LOOP1
ACALL LOOP2

8. Long Addressing

The long addressing mode is used with the instructions LJMP and LCALL. These are 3 byte instructions. The address specifies a full 16 bit destination address so that a jump or a call can be made to a location within a 64 Kbyte code memory space.

Eg. LJMP FINISH
LCALL DELAY

9. Bit Inherent Addressing

In this addressing, the address of the flag which contains the operand, is implied in the opcode of the instruction.

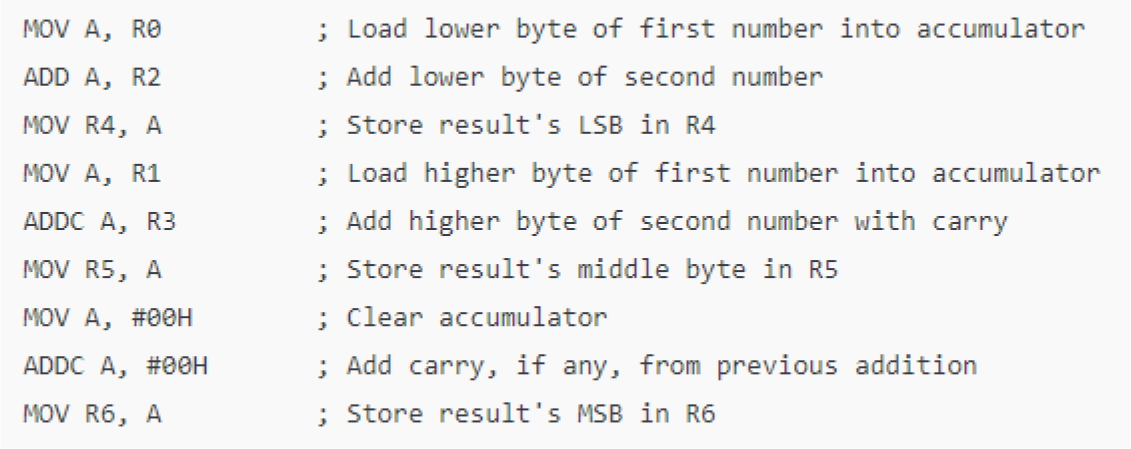
Eg. CLR C *; Clears the carry flag to 0*

10. Bit Direct Addressing

In this addressing mode the direct address of the bit is specified in the instruction. The RAM space 20H to 2FH and most of the special function registers are bit addressable. Bit address values are between 00H to 7FH.

Eg. CLR 07h ;
;
SETB 07H

b. Write an ALP to add 16-bit numbers loaded in R1 R0 and R3R2 . Store the result in R6R5 and R4 from MSB to LSB

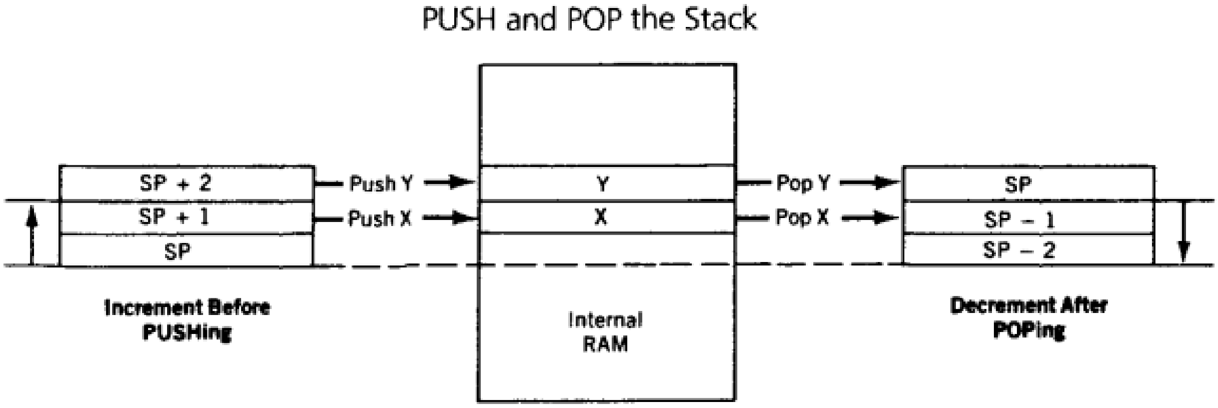


```
MOV A, R0          ; Load lower byte of first number into accumulator
ADD A, R2          ; Add lower byte of second number
MOV R4, A          ; Store result's LSB in R4
MOV A, R1          ; Load higher byte of first number into accumulator
ADDC A, R3         ; Add higher byte of second number with carry
MOV R5, A          ; Store result's middle byte in R5
MOV A, #00H        ; Clear accumulator
ADDC A, #00H       ; Add carry, if any, from previous addition
MOV R6, A          ; Store result's MSB in R6
```

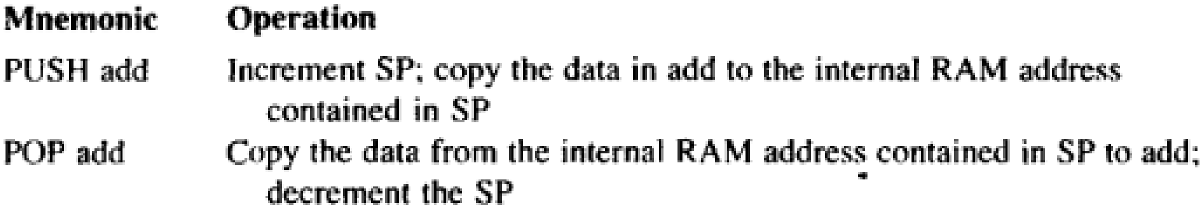
Q.4.a. Define Stack. Explain the operation of Stack using Stack Pointer, PUSH and POP Instructions.

A **stack** is a specialized data structure in computer science that operates on a **Last In, First Out (LIFO)** principle. This means that the last item added to the stack is the first one to be removed. Stacks are commonly used in programming and computer organization to manage function calls, interrupt handling, and memory storage for temporary data.

Push and Pop Opcodes

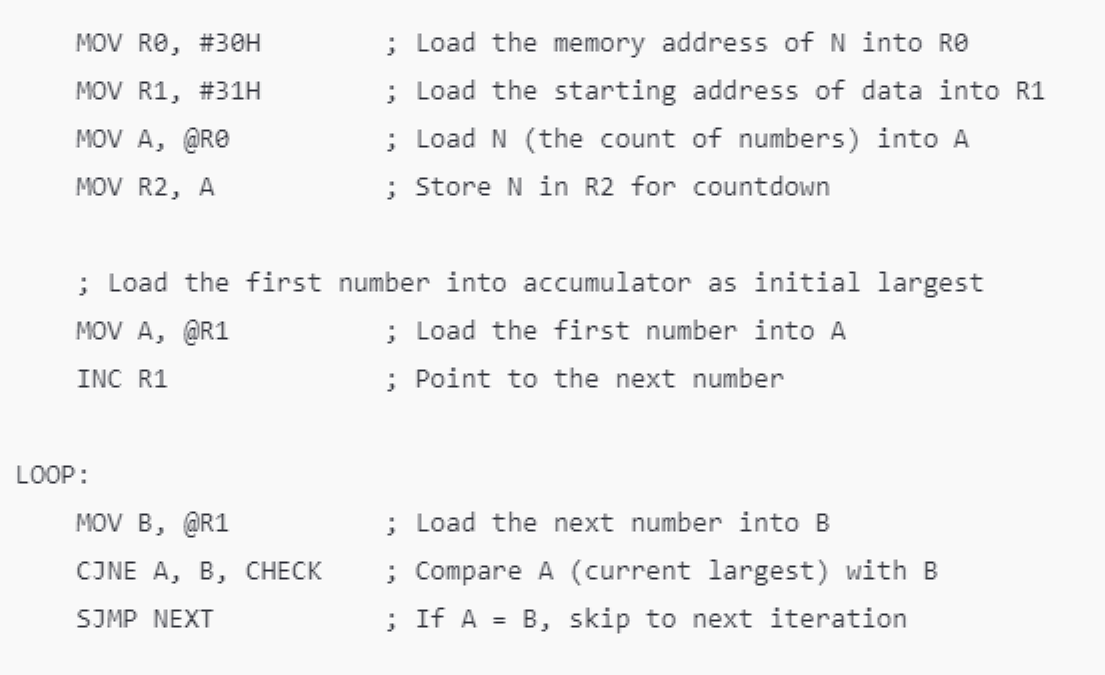


PUSH and POP the Stack

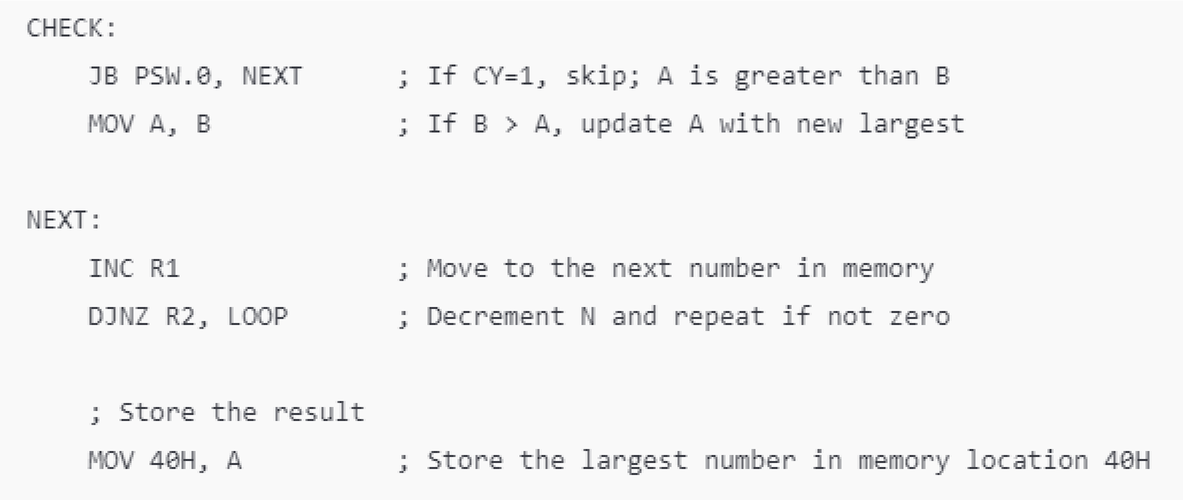


Mnemonic	Operation
PUSH add	Increment SP; copy the data in add to the internal RAM address contained in SP
POP add	Copy the data from the internal RAM address contained in SP to add; decrement the SP

b. Write an ALP to find the largest of N numbers.

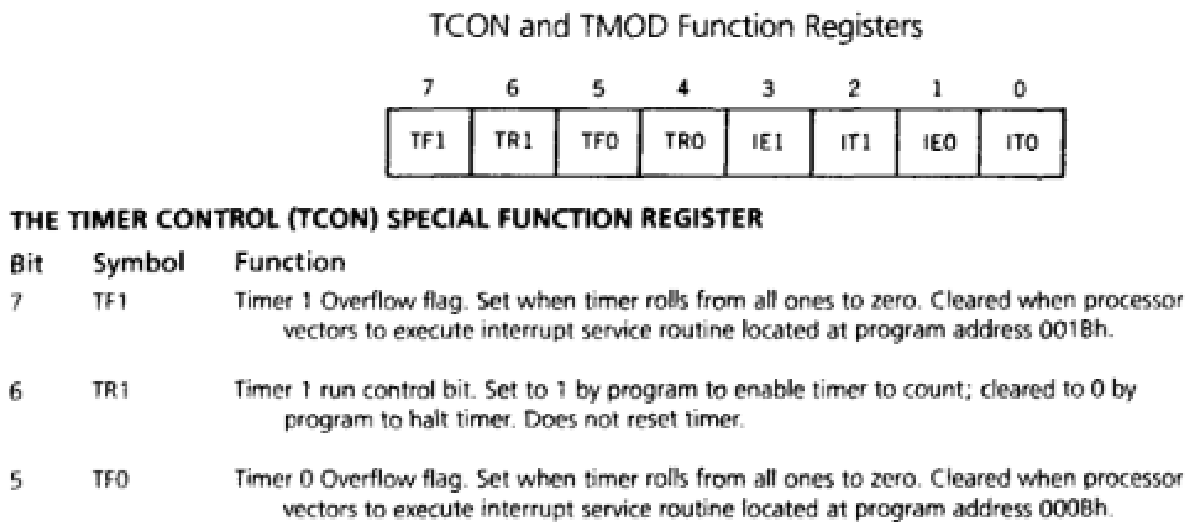


```
    MOV R0, #30H        ; Load the memory address of N into R0
    MOV R1, #31H        ; Load the starting address of data into R1
    MOV A, @R0          ; Load N (the count of numbers) into A
    MOV R2, A           ; Store N in R2 for countdown

    ; Load the first number into accumulator as initial largest
    MOV A, @R1          ; Load the first number into A
    INC R1              ; Point to the next number

LOOP:
    MOV B, @R1          ; Load the next number into B
    CJNE A, B, CHECK    ; Compare A (current largest) with B
    SJMP NEXT           ; If A = B, skip to next iteration
```



```
CHECK:
    JB PSW.0, NEXT      ; If CY=1, skip; A is greater than B
    MOV A, B            ; If B > A, update A with new largest

NEXT:
    INC R1              ; Move to the next number in memory
    DJNZ R2, LOOP       ; Decrement N and repeat if not zero

    ; Store the result
    MOV 40H, A          ; Store the largest number in memory location 40H
```

Q.5. a. Explain (i) TMOD (ii) TCON register of 8051



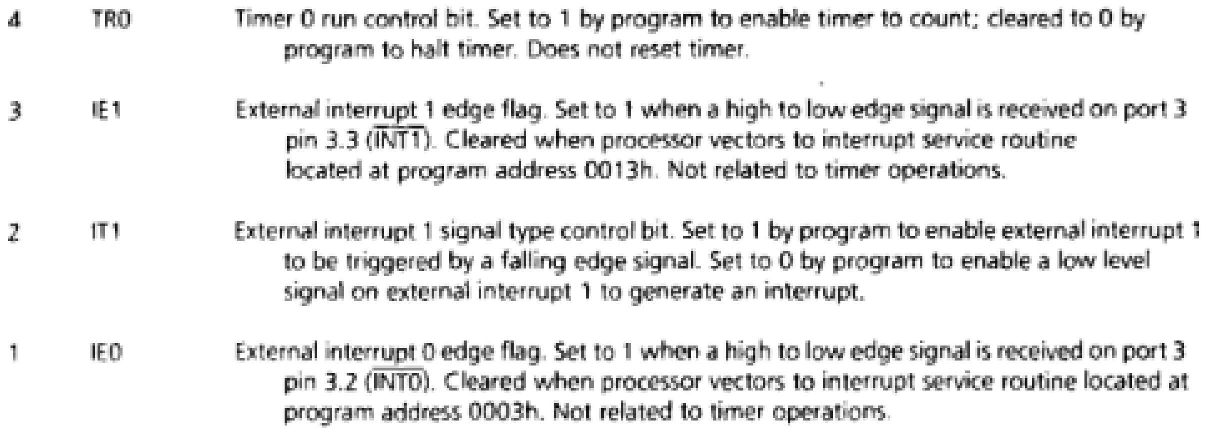
TCON and TMOD Function Registers

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

THE TIMER CONTROL (TCON) SPECIAL FUNCTION REGISTER

Bit	Symbol	Function
7	TF1	Timer 1 Overflow flag. Set when timer rolls from all ones to zero. Cleared when processor vectors to execute interrupt service routine located at program address 001Bh.
6	TR1	Timer 1 run control bit. Set to 1 by program to enable timer to count; cleared to 0 by program to halt timer. Does not reset timer.
5	TF0	Timer 0 Overflow flag. Set when timer rolls from all ones to zero. Cleared when processor vectors to execute interrupt service routine located at program address 000Bh.

TCON & TMOD Function Registers



4	TR0	Timer 0 run control bit. Set to 1 by program to enable timer to count; cleared to 0 by program to halt timer. Does not reset timer.
3	IE1	External interrupt 1 edge flag. Set to 1 when a high to low edge signal is received on port 3 pin 3.3 ($\overline{INT1}$). Cleared when processor vectors to interrupt service routine located at program address 0013h. Not related to timer operations.
2	IT1	External interrupt 1 signal type control bit. Set to 1 by program to enable external interrupt 1 to be triggered by a falling edge signal. Set to 0 by program to enable a low level signal on external interrupt 1 to generate an interrupt.
1	IE0	External interrupt 0 edge flag. Set to 1 when a high to low edge signal is received on port 3 pin 3.2 ($\overline{INT0}$). Cleared when processor vectors to interrupt service routine located at program address 0003h. Not related to timer operations.

b. Assume XTAL = 22 MHz. Write an ALP to generate a square wave of frequency 1KHz on Pin P1.2.

Calculations:

1. **Clock Frequency (XTAL)** = 22 MHz.
2. **8051 Machine Cycle Frequency** = XTAL / 12 = 22 MHz / 12 = 1.833 MHz.
3. **Timer Clock Period** = 1 / 1.833 MHz ≈ 0.546 µs.

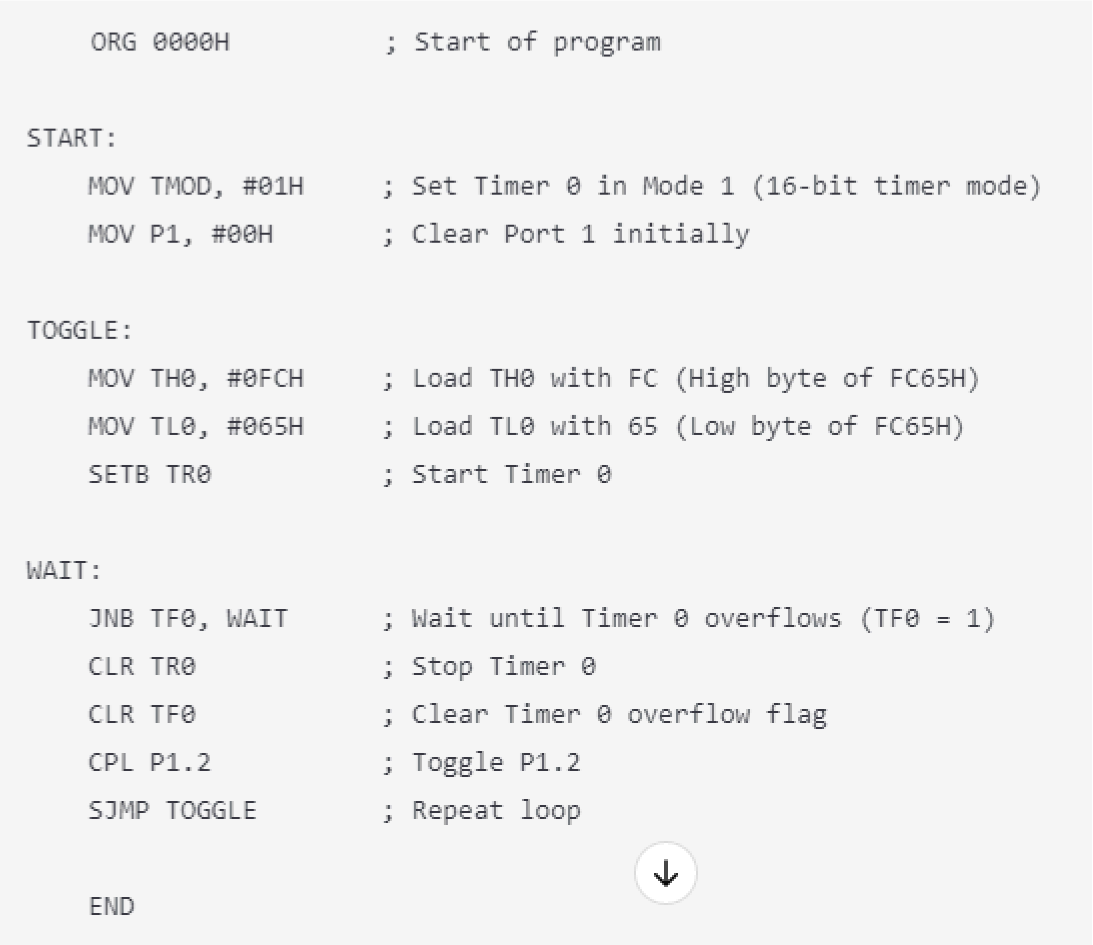
To generate a 1 kHz square wave, the period TTT should be **1 ms** (since Frequency=1Period\text{Frequency} = \frac{1}{\text{Period}}Frequency=Period1). For a square wave, the **high** and **low** times are equal, so each state (high or low) should last **0.5 ms**.

4. **Timer Delay Required** = 0.5 ms = 500 µs.
5. **Timer Counts Required** = 500 µs0.546 µs≈916\frac{500 \, \text{µs}}{0.546 \, \text{µs}} \approx 9160.546µs500µs≈916 counts.

Since Timer 0 in mode 1 (16-bit timer) counts from **65536** down to **0**, the timer must be loaded with 65536−916=6462065536 - 916 = 6462065536−916=64620 (which in hexadecimal is **FC65H**).

ALP to Generate a 1 kHz Square Wave on Pin P1.2:

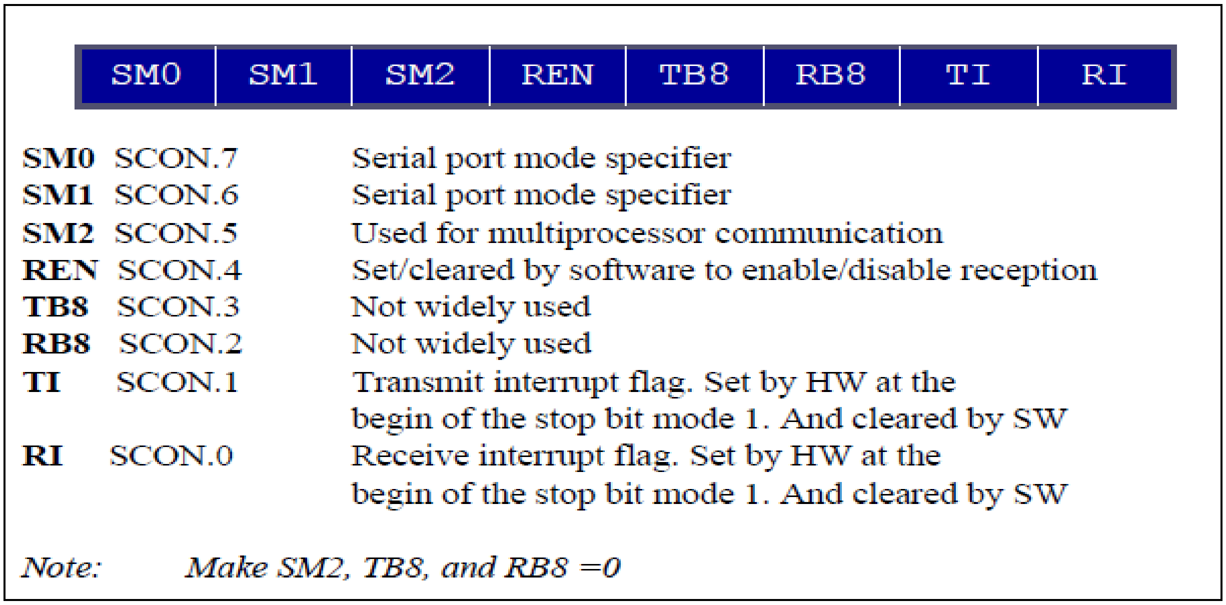
Here's the program to toggle **P1.2** every 0.5 ms, creating a 1 kHz square wave.



```
    ORG 0000H          ; Start of program

START:
    MOV TMOD, #01H     ; Set Timer 0 in Mode 1 (16-bit timer mode)
    MOV P1, #00H       ; Clear Port 1 initially

TOGGLE:
    MOV TH0, #0FCH     ; Load TH0 with FC (High byte of FC65H)
    MOV TL0, #065H     ; Load TL0 with 65 (Low byte of FC65H)
    SETB TR0           ; Start Timer 0

WAIT:
    JNB TF0, WAIT      ; Wait until Timer 0 overflows (TF0 = 1)
    CLR TR0            ; Stop Timer 0
    CLR TF0            ; Clear Timer 0 overflow flag
    CPL P1.2           ; Toggle P1.2
    SJMP TOGGLE        ; Repeat loop

    END
```

6. a. Explain (i) SCON Register (ii) Importance of TI Flag

- SCON is an 8-bit register used to program the start bit, stop bit, and data bits of data framing, among other things

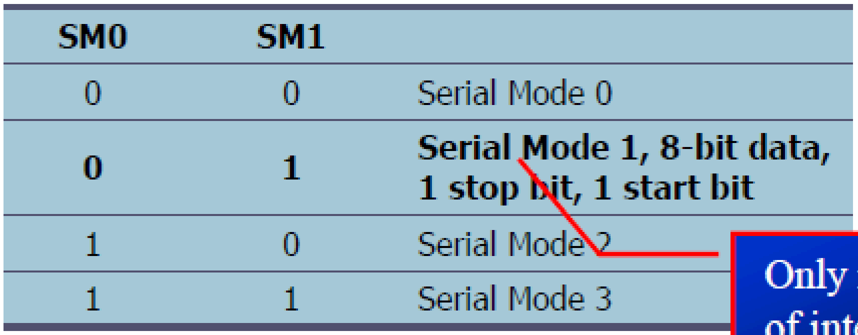


SM0	SM1	SM2	REN	TB8	RB8	TI	RI

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

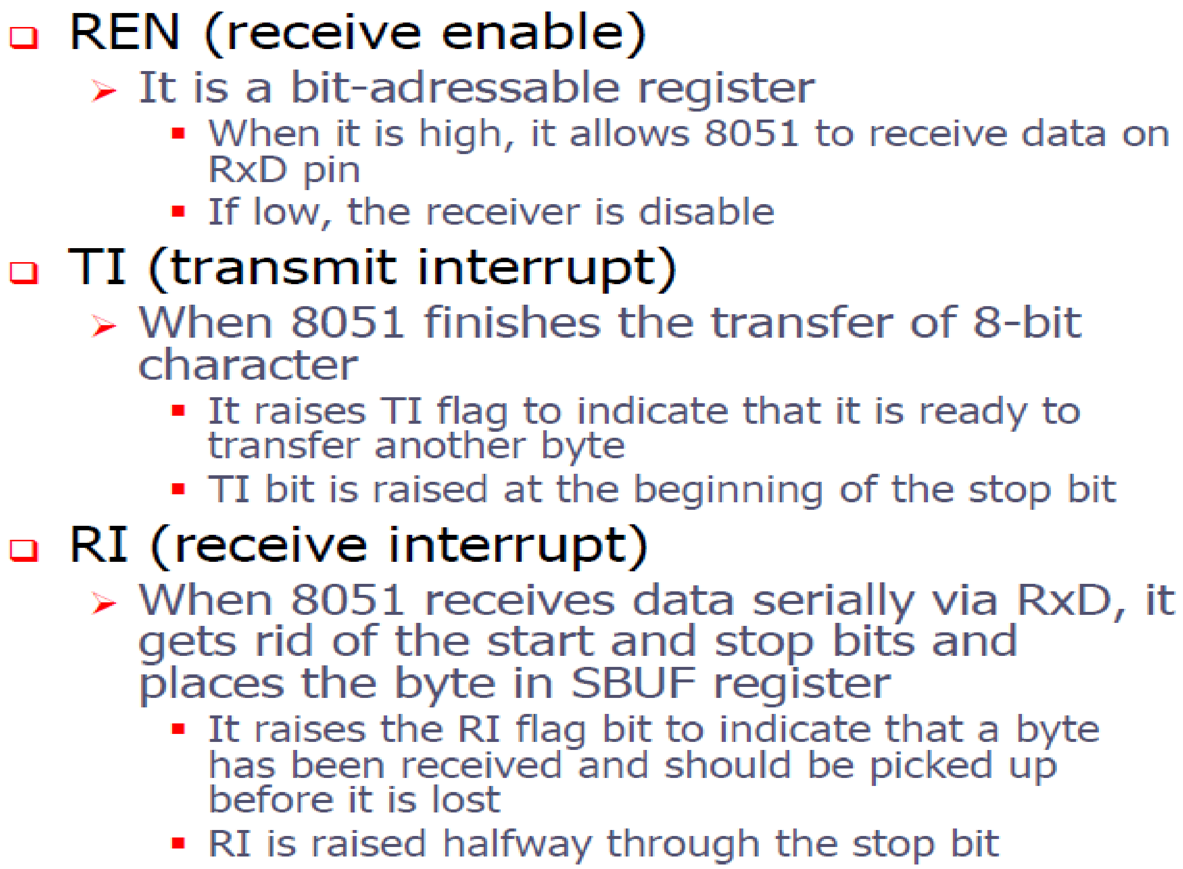
- SM0, SM1
  - They determine the framing of data by specifying the number of bits per character, and the start and stop bits



SM0	SM1	
0	0	Serial Mode 0
0	**1**	**Serial Mode 1, 8-bit data, 1 stop bit, 1 start bit**
1	0	Serial Mode 2
1	1	Serial Mode 3

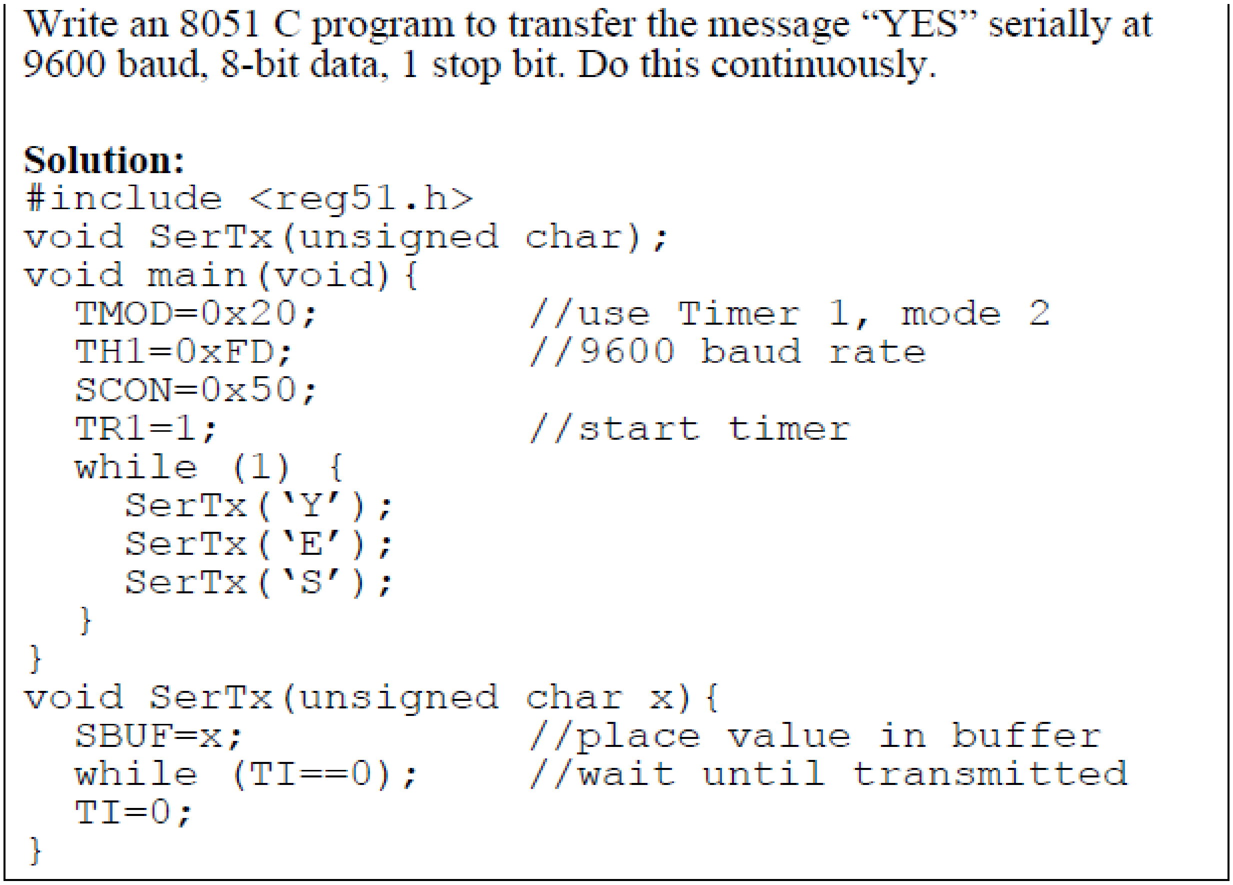
Only mode 1 is of interest to us

- SM2
  - This enables the multiprocessing capability of the 8051



- REN (receive enable)
  - It is a bit-adressable register
    - When it is high, it allows 8051 to receive data on RxD pin
    - If low, the receiver is disable
- TI (transmit interrupt)
  - When 8051 finishes the transfer of 8-bit character
    - It raises TI flag to indicate that it is ready to transfer another byte
    - TI bit is raised at the beginning of the stop bit
- RI (receive interrupt)
  - When 8051 receives data serially via RxD, it gets rid of the start and stop bits and places the byte in SBUF register
    - It raises the RI flag bit to indicate that a byte has been received and should be picked up before it is lost
    - RI is raised halfway through the stop bit

B. Write a C program to transfer “Yes” serially at 9600 baud rate, 8 bit data, 1 stop bit do it continuously.



Write an 8051 C program to transfer the message “YES” serially at 9600 baud, 8-bit data, 1 stop bit. Do this continuously.

Solution:

```
#include <reg51.h>
void SerTx(unsigned char);
void main(void){
  TMOD=0x20;          //use Timer 1, mode 2
  TH1=0xFD;           //9600 baud rate
  SCON=0x50;
  TR1=1;              //start timer
  while (1) {
    SerTx(‘Y’);
    SerTx(‘E’);
    SerTx(‘S’);
  }
}
void SerTx(unsigned char x){
  SBUF=x;             //place value in buffer
  while (TI==0);      //wait until transmitted
  TI=0;
}
```

7. a. Define Interrupt. List the steps involved in executing an Interrupt.

Interrupts

- An interrupt is an external or internal event that interrupts the microcontroller to inform it that a device needs its service

  A single microcontroller can serve several devices by two ways

  1. Interrupts 2. Polling

  Interrupt

  a). Whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal

  b). Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device

c). The program which is associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler

d). It is request based and priority based service with out wasting processing time.

Interrupts

- **An interrupt is an external or internal event that interrupts the microcontroller to inform it that a device needs its service**

  A single microcontroller can serve several devices by two ways

1. Interrupts 2. Polling

Interrupt

a). Whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal

b). Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device

c). The program which is associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler

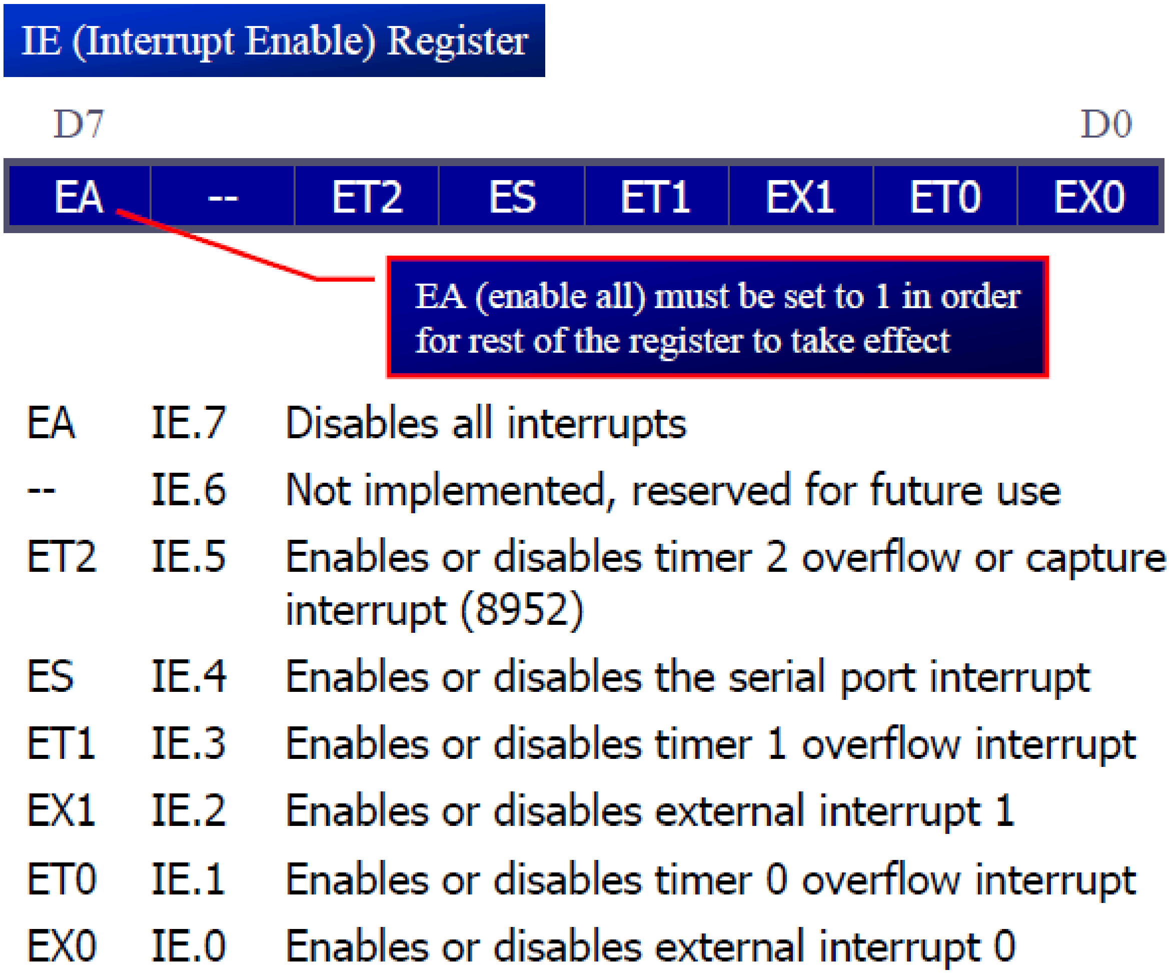
d). It is request based and priority based service with out wasting processing time.

b. Explain Interrupt Vector Table of 8051 Microcontroller.

Each interrupt has a specific place in code memory where program execution (interrupt service routine) begins. There are only eight memory locations available for each interrupt. If ISR is bigger use LJMP instruction)

Interrupt	Vector address	Type	Priority
Reset	0000H	External / Hardware	1 (Highest)
External Interrupt 0	0003H	External / Hardware	2
Timer Interrupt 0	000BH	Internal / Software	3
External Interrupt 01	0013H	External / Hardware	4
Timer Interrupt 1	001BH	Internal / Software	5
Serial Interrupt	0023H	Internal / software	6

C. Explain Interrupt Enable Register.



IE (Interrupt Enable) Register

D7							D0
EA	--	ET2	ES	ET1	EX1	ET0	EX0

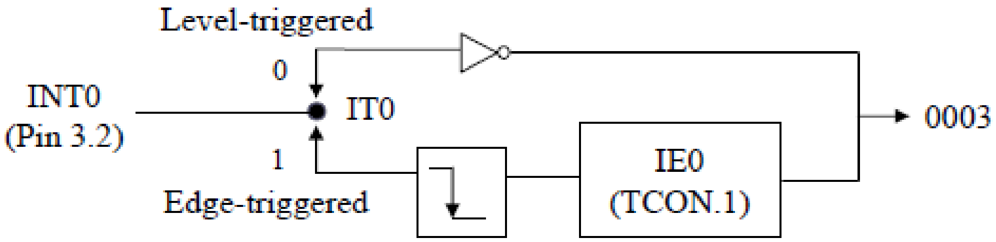
EA (enable all) must be set to 1 in order for rest of the register to take effect

EA	IE.7	Disables all interrupts
--	IE.6	Not implemented, reserved for future use
ET2	IE.5	Enables or disables timer 2 overflow or capture interrupt (8952)
ES	IE.4	Enables or disables the serial port interrupt
ET1	IE.3	Enables or disables timer 1 overflow interrupt
EX1	IE.2	Enables or disables external interrupt 1
ET0	IE.1	Enables or disables timer 0 overflow interrupt
EX0	IE.0	Enables or disables external interrupt 0

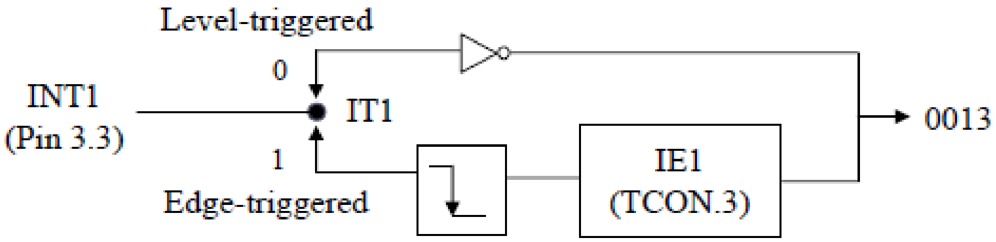
8. a. Explain Interrupt control used in 8051.

- The 8051 has two external hardware interrupts
  - Pin 12 (P3.2) and pin 13 (P3.3) of the 8051, designated as INT0 and INT1, are used as external hardware interrupts
    - The interrupt vector table locations 0003H and 0013H are set aside for INT0 and INT1
  - There are two activation levels for the external hardware interrupts
    - Level trigged
    - Edge trigged

Activation of INT0



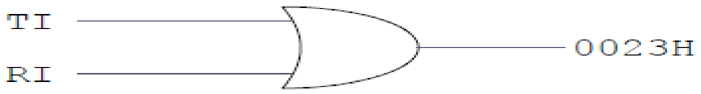
Activation of INT1



b. Explain the steps involved in programming serial communication Interrupt.

- TI (transfer interrupt) is raised when the last bit of the framed data, the stop bit, is transferred, indicating that the SBUF register is ready to transfer the next byte
- RI (received interrupt) is raised when the entire frame of data, including the stop bit, is received
  - In other words, when the SBUF register has a byte, RI is raised to indicate that the received byte needs to be picked up before it is lost (overrun) by new incoming serial data

- In the 8051 there is only one interrupt set aside for serial communication
  - This interrupt is used to both send and receive data
  - If the interrupt bit in the IE register (IE.4) is enabled, when RI or TI is raised the 8051 gets interrupted and jumps to memory location 0023H to execute the ISR
  - In that ISR we must examine the TI and RI flags to see which one caused the interrupt and respond accordingly



Serial interrupt is invoked by TI or RI flags

- The serial interrupt is used mainly for receiving data and is never used for sending data serially

C. Explain how multiple interrupts were handled in 8051.

Interrupt Handling Workflow:

1. An interrupt is triggered (based on priority and enabled settings).
2. The 8051 checks the priority and, if permitted, halts the main program.
3. The 8051 jumps to the vector address of the interrupt and executes the ISR.
4. The ISR completes and executes the `RETI` instruction to return to the main program.

By using the **IE** and **IP registers**, the 8051 microcontroller can handle multiple interrupt sources efficiently, allowing high-priority tasks to interrupt lower-priority ones and ensuring smooth execution of critical tasks.

9. a. Explain DAC Interface with a neat diagram and also write a program to generate a staircase waveform.

The **Digital-to-Analog Converter (DAC)** interface allows a digital system like the **8051 microcontroller** to output analog voltages. DACs convert digital values (binary numbers) into proportional analog voltage or current outputs. For generating waveforms (such as sine, square, and staircase), the DAC interface is commonly used with 8051.

Basic Diagram of DAC Interface with 8051

The 8051's **Port 1** (P1) is often used to interface with an **8-bit DAC** (e.g., **DAC0808**). The digital output from the microcontroller's port is sent to the DAC, which outputs a corresponding analog voltage.

Diagram

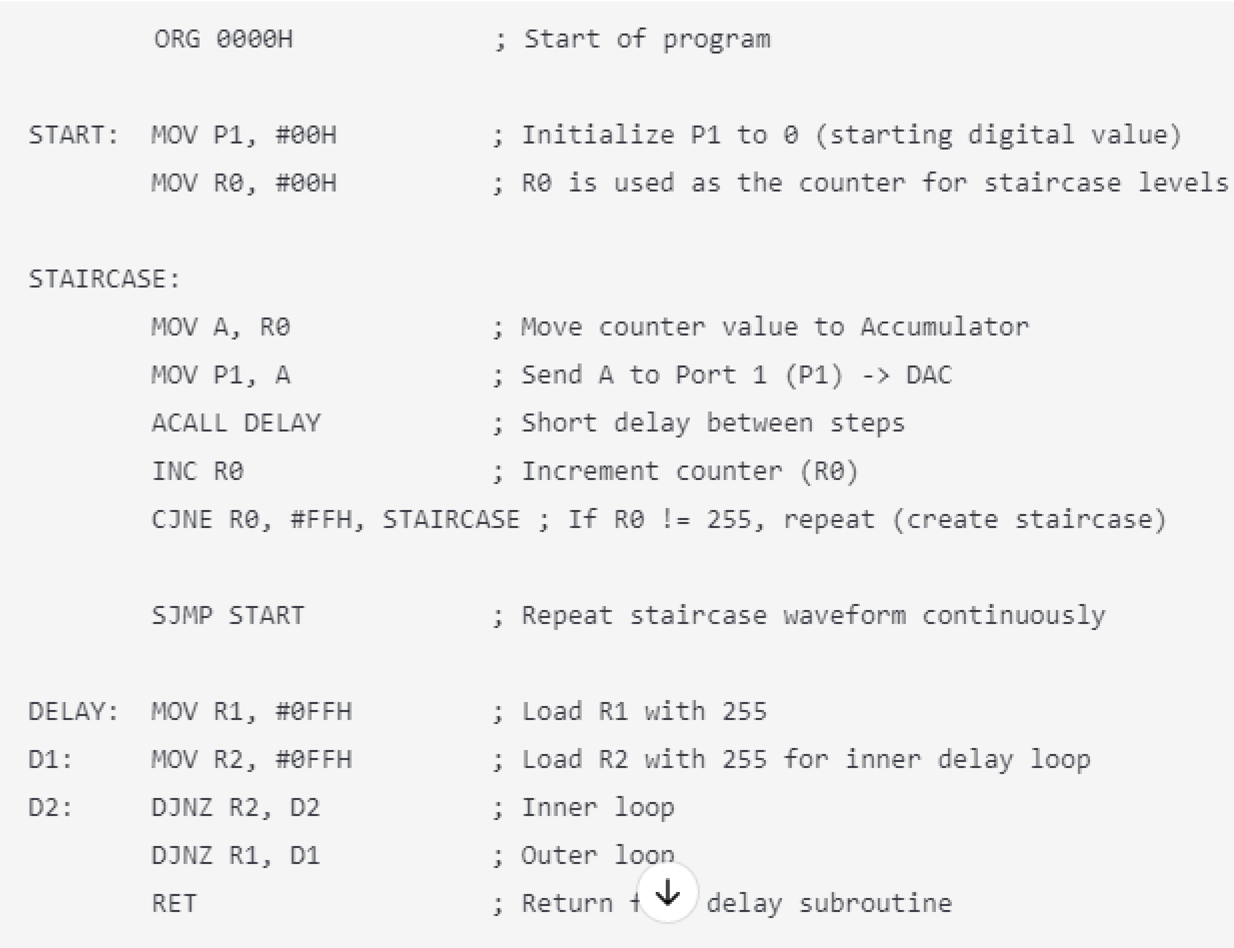
Here's a description to help visualize it:

1. **8051 Microcontroller** with its **P1.0 to P1.7** pins connected to the **D0-D7** input pins of the **DAC**.
2. **DAC0808** is connected to **V_ref** for reference voltage and **V_out** for analog output.
3. The DAC's analog output (**V_out**) is connected to a **load resistor** or connected to an oscilloscope to view the waveform.

Staircase Waveform Generation Program

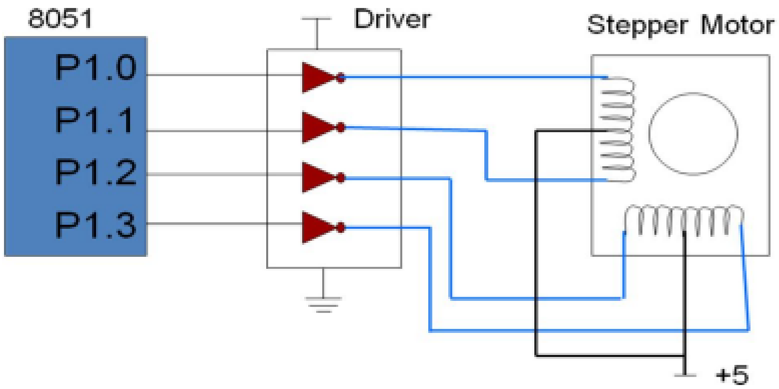
To generate a **staircase waveform** using 8051 and DAC, we increment the digital value sent to the DAC in steps. Each increment increases the output analog voltage by a small amount, creating a "staircase" effect.

Here’s an Assembly Language Program (ALP) for generating a staircase waveform:



```
        ORG 0000H            ; Start of program

START:  MOV P1, #00H         ; Initialize P1 to 0 (starting digital value)
        MOV R0, #00H         ; R0 is used as the counter for staircase levels

STAIRCASE:
        MOV A, R0            ; Move counter value to Accumulator
        MOV P1, A            ; Send A to Port 1 (P1) -> DAC
        ACALL DELAY          ; Short delay between steps
        INC R0               ; Increment counter (R0)
        CJNE R0, #FFH, STAIRCASE ; If R0 != 255, repeat (create staircase)

        SJMP START           ; Repeat staircase waveform continuously

DELAY:  MOV R1, #0FFH        ; Load R1 with 255
D1:     MOV R2, #0FFH        ; Load R2 with 255 for inner delay loop
D2:     DJNZ R2, D2          ; Inner loop
        DJNZ R1, D1          ; Outer loop
        RET                  ; Return f    delay subroutine
```

b. With a neat diagram, write a program to interface Stepper Motor to 8051 microcontroller.



8051 interfaces to stepper motor

Example 1: Write an ALP to rotate the stepper motor clockwise / anticlockwise
continuously with full step sequence.

Program:

```
MOV A,#66H
BACK: MOV P1,A
RR A
ACALL DELAY
SJMP BACK
DELAY: MOV R1,#100
UP1: MOV R2,#50
UP: DJNZ R2,UP
DJNZ R1,UP1
RET
```

Note: motor to rotate in anticlockwise use instruction RL A instead of RR A

10. a. Explain the Interfacing of DC Motor using C programming.

A switch is connected to pin P2.7. Write a C to monitor the status of the SW.

If SW = 0, DC motor moves clockwise and if SW = 1, DC motor moves anticlockwise.

Program:

```
# include <reg51.h>
sbit SW =P2^7;
sbit Enable = P1^0;
sbit MTR_1 = P1^1;
sbit MTR_2 = P1^2;
void main ( )
{
SW=1;
Enable = 0;
MTR_1=0;
MTR_2=0;
while( )
{
Enable =1;
if( SW==1)
{ MTR_1=1;
MTR_2=0;
}
else
{ MTR_1=0;
MTR_2=1;
}}}
```

b. With a neat diagram, write an ALP to interface the LCD to 8051 microcontroller.

```
ORG 0H
MOV A,#38H ;INIT. LCD 2 LINES, 5X7 MATRIX
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time
```

```
MOV A,#0EH ;display on, cursor on
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time
MOV A,#01 ;clear LCD
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time
MOV A,#06H ;shift cursor right
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time
MOV A,#86H ;cursor at line 1, pos. 6
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time
ACALL DATAWRT ;call display subroutine
ACALL DELAY ;give LCD some time
E ;display letter E
ACALL DATAWRT ;call display subroutine
ACALL DELAY ;give LCD some time
S ;display letter S
ACALL DATAWRT ;call display subroutine
AGAIN: SJMP AGAIN ;stay here
COMNWRT: ;send command to LCD
MOV P1,A ;copy reg A to port 1
CLR P2.0 ;RS=0 for command
CLR P2.1 ;R/W=0 for write
SETB P2.2 ;E=1 for high pulse
ACALL DELAY ;give LCD some time
CLR P2.2 ;E=0 for H-to-L pulse
RET
DATAWRT: ;write data to LCD
MOV P1,A ;copy reg A to port 1
SETB P2.0 ;RS=1 for data
CLR P2.1 ;R/W=0 for write
SETB P2.2 ;E=1 for high pulse
ACALL DELAY ;give LCD some time
CLR P2.2 ;E=0 for H-to-L pulse
```

```
RET
DELAY:
MOV R3,#50 ;50 or higher for fast CPUs
HERE2: MOV R4,#255 ;R4 = 255
HERE: DJNZ R4,HERE ;stay until R4 becomes 0
DJNZ R3,HERE2
RET
END
```