BEC405A_Microcontrollers solution- June July 2025

CBCS SCHEME

USN | | | | | | | | | | BEC405A

Fourth Semester B.E./B.Tech. Degree Examination, June/July 2025
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.	With diagrams, explain the RAM structure of 8051 microcontroller.	8	L2	CO1
	b.	With necessary sketches, explain (i) Flags and program status word (ii) Stack operation.	8	L2	CO1
	c.	Write a note on Embedded Microcontrollers.	4	L1	CO1
		OR			
Q.2	a.	With a neat diagram, explain the block diagram of 8051 microcontroller.	8	L2	CO1
	b.	Write an interfacing diagram of 8051 microcontroller interfaced to 16K bytes of RAM.	8	L2	CO1
	c.	Compare CISC and RISC architecture.	4	L2	CO1
		Module – 2			
Q.3	a.	Write a program segment to copy the value 55 H into RAM memory locations 40 H to 44 H using, (i) Direct addressing mode, (ii) Register indirect addressing mode without a loop (iii) and with a loop	6	L2	CO2
	b.	Explain the following instructions with examples: (i) Move A, @A + DPTR (ii) RRC A (iii) DA A	6	L2	CO2
	c.	Briefly explain the arithmetics instructions of 8051 microcontroller.	8	L2	CO2
		OR			
Q.4	a.	Write an assembly language program to multiply the number present in external memory location 800 AH and 8050 H. Store the lower byte of result obtained in R0 and higher byte in R1.	8	L3	CO2
	b.	Explain the role of CALL and subroutines in 8051 microcontroller programming. Give an example.	4	L2	CO2
	c.	If the number A6H is placed in external RAM between locations 0100H and 0200H. Write an assembly language program to find the address of that location and place that address in R6 and R7 registers.	8	L3	CO2
		Module – 3			
Q.5	a.	Explain the functions of each bit in the TMOD register.	6	L2	CO3
	b.	Explain MODE-1 programming of timers in 8051.	6	L2	CO3
	c.	Write a 8051 C program to transmit the message 'ECE' using serial communication port of 8051. Use baud rate 4800.	8	L2	CO3

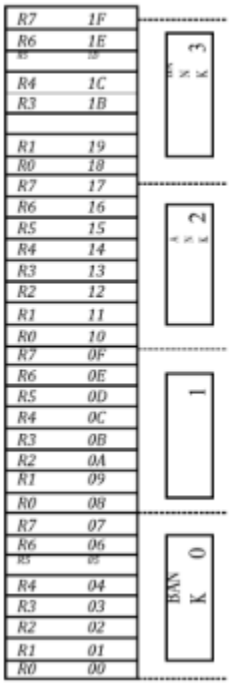
		OR			
Q.6	a.	Explain the importace of TI flag and RI flag.	6	L2	CO3
	b.	Write the steps required for programming 8051 to transmit and receive the data serially.	6	L2	CO3
	c.	Explain how timers are used as counters and also explain the counters operation using a code snippet.	8	L2	CO3
		Module – 4			
Q.7	a.	Explain the following : (i) Interrupt (ii) Interrupt Service Routine (ISR) (iii) Interrupt Vector Table (IVT)	8	L2	CO4
	b.	Write the instructions to : (i) Enable the serial interrupt, timer 0 interrupt and external hardware interrupt. (ii) Disable the timer 0 interrupt. (iii) Disable all interrupts with a single instruction. Use bit manipulation instructions for all the cases.	6	L2	CO4
	c.	Explain the bit contents of IE register.	6	L2	CO4
		OR			
Q.8	a.	List the steps involved in executing interrupts in 8051 microcontroller.	6	L2	CO4
	b.	Assume XTAL = 11.0592 MHz. Use timer 0 to create the square wave. Write a assembly program that continuously gets a 8 bit of data from P(0) and sends it to P(1). While simultaneously creating square wave of 200 µs period on P2.5.	8	L3	CO4
	c.	Write the interrupt priority upon reset in 8051. Also explain how the priority of the interrupts can be set using IP register.	6	L2	CO4
		Module – 5			
Q.9	a.	With neat diagram, write an assembly language program to interface stepper motor to 8051 microcontroller.	10	L3	CO5
	b.	Explain DAC interface with diagram and also write program to generate triangular waveform.	10	L3	CO5
		OR			
Q.10	a.	With neat diagram, write an assembly language program to interface LCD to 8051 microcontroller.	10	L3	CO5
	b.	A door sensor is connected to the P1.1 pin and a buzzer is connected to P1.7. Write 8051 C program to monitor the door sensor and when it opens, sound the buzzer. The buzzer can be sound by sending a square wave of a few hundred Hz.	10	L2	CO5

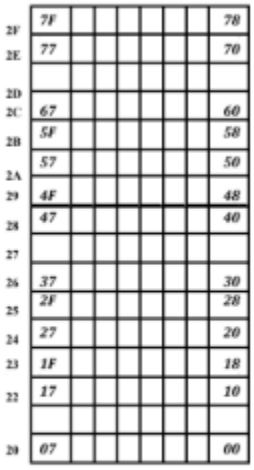
* * * * *

		Module – 1	M	L	C
Q.1	a.	With diagrams, explain the RAM structure of 8051 microcontroller.	8	L2	CO1
	b.	With necessary sketches, explain (i) Flags and program status word (ii) Stack operation.	8	L2	CO1
	c.	Write a note on Embedded Microcontrollers.	4	L1	CO1

1a)

Internal RAM organization





Bit addressable memory

General purpose memory

Working Registers

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.

1b)

PSW Register in 8051 Microcontroller | Program Status Word

- Flags are single bit register and used to store the result of certain function after executing instruction. Flags are grouped inside PSW and PCON registers.
- PSW register in 8051 microcontroller contains math flags and PCON contains general user flags.
- Math flags are grouped in PSW of microcontroller 8051 and they are Carry(C), Auxiliary Carry (AC), Over flow (OV), and Parity (P). The general user flags are GF0 and GF1 which are grouped in PCON register.

PSW Register in 8051

- The PSW is accessible fully as an 8-bit register, with the address D0H.
- The bit pattern of this flag register is

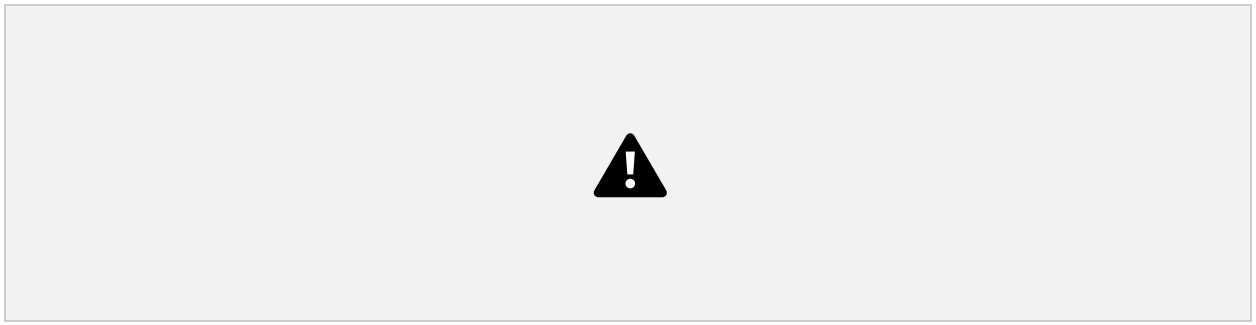


Figure 1: Format of PSW register in 8051 Microcontroller

Parity bit (P)

- This parity flag bit is used to show the number of 1s in the accumulator only. If the accumulator register contains an odd number of 1s, then this flag set to 1.
- If accumulator contains even number of 1s, then this flag cleared to 0.

Overflow flag (OV)

- This flag is set during ALU operations, to indicate overflow in the result. It is set to 1 if there is a carry out of either the D7 bit or the D6 bit of the accumulator.
- Overflow flag is set when arithmetic operations such as add and subtract result in sign conflict.
-
- **The conditions under which the OV flag is set are as follows:**
  - Positive + Positive = Negative
  - Negative + Negative = Positive
  - Positive – Negative = Negative
  - Negative – Positive = Positive

Register bank select bits (RS1 and RS0)

- These two bits are used to select one of four register banks of RAM. By setting and clearing these bits, registers R0-R7 are stored in one of four banks of RAM as follows.

RS1	RS0	Bank Selected	Address of Registers
0	0	Bank 0	00h-07h

0	1	Bank 1	08h-0Fh
1	0	Bank 2	10h-17h
1	1	Bank 3	18h-1Fh

-
  - These bits are user-programmable. They can be set by the programmer to point to the correct register banks.
  - The register bank selection in the programs can be changed using these two bits.

General-purpose flag (F0)

-
  - This is a user-programmable flag; the user can program and store any bit of his/her choice in this flag, using the bit address.

Auxiliary carry flag (AC)

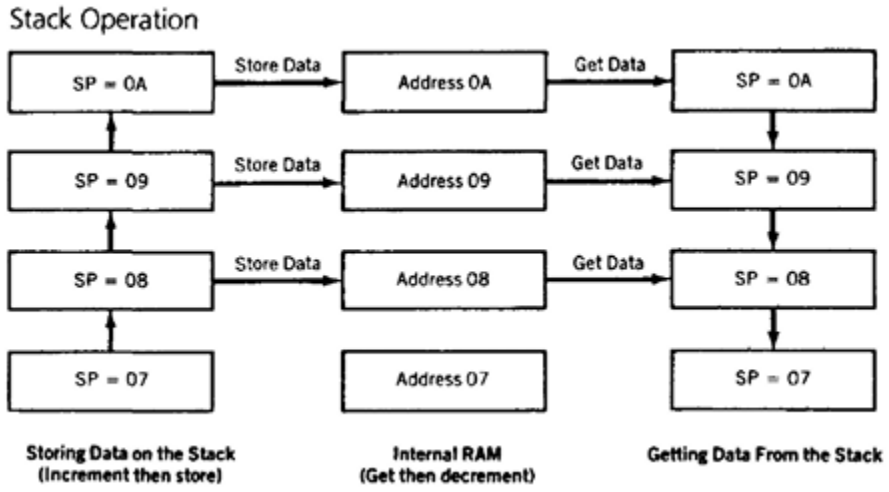
-
  - It is used in association with BCD arithmetic. This flag is set when there is a carry out of the D3 bit of the accumulator.

Carry flag (CY)

-
  - This flag is used to indicate the carry generated after arithmetic operations. It can also be used as an accumulator, to store one of the data bits for bit-related Boolean instructions.
- The 8051 supports bit manipulation instructions.
- This means that in addition to the byte operations, bit operations can also be done using bit data.
- For this purpose, the contents of the PSW are bit-addressable.

The Stack and the Stack Pointer • The stack refers to an area of internal RAM that is used in conjunction with certain opcodes to store and retrieve data quickly. • The 8-bit stack pointer (SP) register is used by the 8051 to hold an internal RAM address that is called the "top of the stack." • The top of the stack is the location in internal RAM where the last byte of data was stored by a stack operation. • The SP increments before storing data on the stack so that the stack grows up as data is stored. • As data is retrieved from the stack, the byte is read from the stack, and then

the SP decrements to point to the next available byte of stored data. • The SP is set to 07h when the 8051 is reset and can be changed to any internal RAM address by the programmer.



1c) Embedded microcontrollers are small, powerful devices that control and interact with the physical world. They're essentially single-chip computers that contain a processing core, memory, and input/output peripherals, making them ideal for specific tasks within larger systems.

Key Features:

- Compact Design: Small size fits into tight spaces
- Low Power Consumption: Perfect for battery-powered devices
- Real-Time Processing: Handles urgent tasks, like in medical devices or car safety features
- Versatile Communication: Works with various protocols like UART, SPI, and I2C

Applications:

- Consumer Electronics: Smartphones, smart TVs, gaming consoles, and home automation systems
- Automotive Systems: Engine control, safety systems, and infotainment systems

- Industrial Automation: Control machinery, monitor equipment, and provide real-time feedback

- IoT Devices: Connects and regulates smart devices, enabling automation and data exchange

Popular Microcontrollers:

- ATmega328P (Arduino Uno): Great for beginners, hobbyists, and DIY projects

- STM32 (Series of ARM Cortex-M chips): Robust, flexible, and user-friendly for newcomers

- ESP32: Excellent for IoT projects with built-in Wi-Fi and Bluetooth

- RP2040 (Raspberry Pi Pico): Powerful, affordable, and easy to use for real-time control and DIY electronics

Benefits:

- Increased Efficiency: Automate tasks, process data quickly, and work with sensors and actuators

- Improved Accuracy: Reduce errors and improve performance in various applications

- Cost-Effective: Available at a range of prices, suitable for mass-produced devices

Future Trends:

- Artificial Intelligence (AI): Integration of AI algorithms for intelligent decision-making and control

- Edge Computing: Processing data locally, reducing dependence on cloud servers

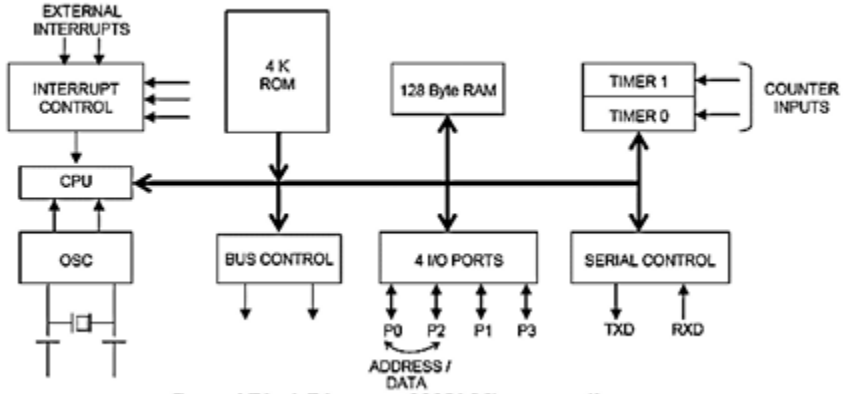
- Increased Connectivity: Seamless communication with other devices and systems for advanced data collection and analysis

2.

		OR			
Q.2	a.	With a neat diagram, explain the block diagram of 8051 microcontroller.	8	L2	CO1
	b.	Write an interfacing diagram of 8051 microcontroller interfaced to 16K bytes of RAM.	8	L2	CO1
	c.	Compare CISC and RISC architecture.	4	L2	CO1

2a)

8051 Architecture



General Block Diagram of 8051 Microcontroller

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

o Working registers [00 – 1F]

o Bit addressable memory area [20 – 2F]

o General purpose memory area (Scratch pad memory) [30-7F]

2b)

Solution: Given, Memory size: 16k
that means we require 2^n=16k :: n address lines
here n=14 :: A_0 to A_{13} address lines are required.
A_{14} and A_{15} are connected through OR gate to CS pin of external RAM.
when A_{14} and A_{15} both are low (logic '0'), external data memory(RAM) is selected.
Address Decoding(Memory Map)for 16k x 8 RAM.

Address	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	HEX adrs.
starting	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H
end	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFFH

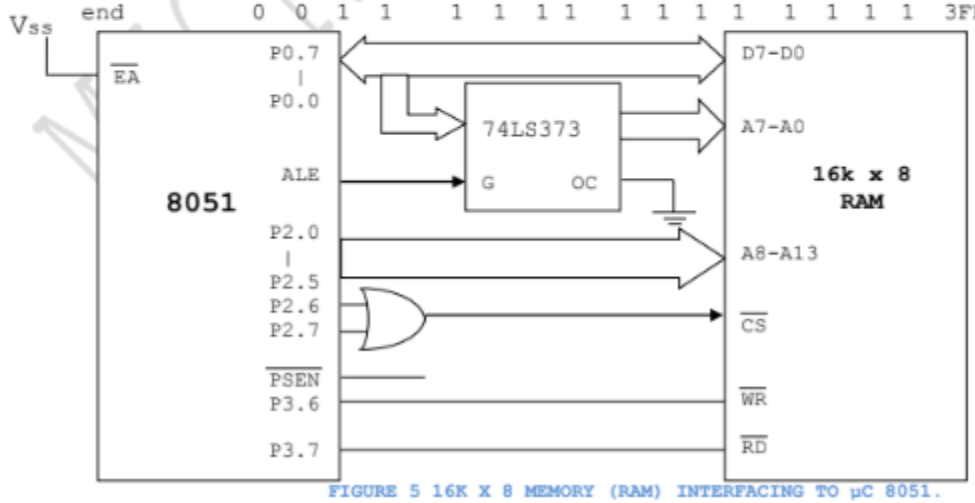


FIGURE 5 16K X 8 MEMORY (RAM) INTERFACING TO µC 8051.

2c)

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

Module – 2

Q.3	a.	Write a program segment to copy the value 55 H into RAM memory locations 40 H to 44 H using, (i) Direct addressing mode, (ii) Register indirect addressing mode without a loop (iii) and with a loop	6	L2	CO2
	b.	Explain the following instructions with examples: (i) Move A, @A + DPTR (ii) RRC A (iii) DA A	6	L2	CO2
	c.	Briefly explain the arithmetics instructions of 8051 microcontroller.	8	L2	CO2

3a)

Direct addressing mode:

```
MOV A, #55H      ; Load accumulator with 55h

    MOV 40H, A      ; Copy 55h into RAM location 40h
    MOV 41H, A      ; Copy 55h into RAM location 41h
    MOV 42H, A      ; Copy 55h into RAM location 42h
```

```
    MOV 43H, A      ; Copy 55h into RAM location 43h
    MOV 44H, A      ; Copy 55h into RAM location 44h

    SJMP $          ; Stay here (infinite loop)
```

Register indirect addressing mode without loop

```
MOV A, #55H      ; Load immediate data 55H into Accumulator
    MOV R0, #40H     ; Load starting address 40H into R0

    MOV @R0, A       ; Store 55H at 40H
    INC R0           ; Point to 41H
    MOV @R0, A       ; Store 55H at 41H
    INC R0           ; Point to 42H
    MOV @R0, A       ; Store 55H at 42H
    INC R0           ; Point to 43H
    MOV @R0, A       ; Store 55H at 43H
    INC R0           ; Point to 44H
    MOV @R0, A       ; Store 55H at 44H

    SJMP $           ; Stay here forever (stop program)
```

Register indirect addressing mode with loop

```
MOV A, #55H      ; Load accumulator with 55H
    MOV R0, #40H     ; Initialize R0 with starting address (40H)
    MOV R1, #05H     ; Counter = 5 (since 40H to 44H → 5 locations)

LOOP:   MOV @R0, A       ; Store 55H into address pointed by R0
    INC R0           ; Point to next memory location
    DJNZ R1, LOOP    ; Decrement R1, repeat until 0
```

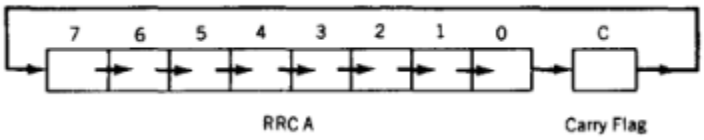
SJMP $; Stop program (infinite loop)

3b)

b.	Explain the following instructions with examples: (i) Move A, @A + DPTR (ii) RRC A (iii) DA A	6	L2	CO2

3b) RRC A

RRC A Rotate the A register and the carry flag, as a ninth bit, one bit position to the right; bit A0 to the carry flag, carry flag to A7, A7 to A6, A6 to A5, A5 to A4, A4 to A3, A3 to A2, A2 to A1, and A1 to A0




```asm
        MOV A, #96H     ; A = 1001 0110b
        SETB C          ; Set carry = 1
        RRC A           ; Rotate right through carry
        SJMP $
```

inal result:

- `A = 0CBh`
- `CY = 0`

DA A

`DA A` = **Decimal Adjust Accumulator**

- It is used **after adding two BCD (Binary-Coded Decimal) numbers**.
- Adjusts the result in `A` so that it becomes a valid **BCD number** (two decimal digits, 00–99).
- Works based on **Auxiliary Carry (AC)** a**Carry (CY)** flags.

The 8051 rules for `DA A` are:

1. If the **lower nibble (A0–A3) > 9** OR **AC = 1**, then `+06` is added to the low nibble.
2. If the **upper nibble (A4–A7) > 9** OR **CY = 1**, then `+60h` is added to the accumulator.

This ensures each nibble of `A` holds a valid **BCD digit (0–9)**.

```
MOV A, #25H      ; A = 25 (BCD 25)
ADD A, #37H      ; A = 25 + 37 = 5Ch
DA A             ; Adjust to valid BCD
```

- After `ADD`: `A = 5Ch = 0101 1100b`
  (upper nibble = 5, lower nibble = 12 → not valid BCD)
- `DA A` fixes it: `A = 62h` (which is valid BCD = decimal 62).

```
MOV A, #48H      ; A = 48
ADD A, #75H      ; A = BDh (not valid BCD)
DA A
```

- After `ADD`: `A = BDh`, CY = 0
- `DA A` adds 60h → `A = 1 1 3h`, CY = 1 (since > 99 BCD)

So final result: BCD = **123**, with CY carrying the hundred's digit.

c. | Briefly explain the arithmetics instructions of 8051 microcontroller.

Arithmetic Instruction

Mnemonic	Operation
INC destination	Increment destination by 1
DEC destination	Decrement destination by 1
ADD/ADDC destination,source	Add source to destination without/with carry (C) flag
SUBB destination,source	Subtract, with carry, source from destination
MUL AB	Multiply the contents of registers A and B
DIV AB	Divide the contents of register A by the contents of register B
DA A	Decimal Adjust the A register

Incrementing and Decrementing

Mnemonic	Operation
INC A	Add a one to the A register
INC Rr	Add a one to register Rr
INC add	Add a one to the direct address
INC @ Rp	Add a one to the contents of the address in Rp
INC DPTR	Add a one to the 16-bit DPTR
DEC A	Subtract a one from register A
DEC Rr	Subtract a one from register Rr
DEC add	Subtract a one from the contents of the direct address
DEC @ Rp	Subtract a one from the contents of the address in register Rp

Addition

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

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

Subtraction

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

Multiplication and Division

Mnemonic	Operation
MUL AB	Multiply A by B; put the low-order byte of the product in A, put the high-order byte in B

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

Decimal Arithmetic

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

4

Q.4	a.	Write an assembly language program to multiply the number present in external memory location 800 AH and 8050 H. Store the lower byte of result obtained in R0 and higher byte in R1.	8	L3	CO2
	b.	Explain the role of CALL and subroutines in 8051 microcontroller programming. Give an example.	4	L2	CO2
	c.	If the number A6H is placed in external RAM between locations 0100H and 0200H. Write an assembly language program to find the address of that location and place that address in R6 and R7 registers.	8	L3	CO2

4A

	a.	Write an assembly language program to multiply the number present in external memory location 800 AH and 8050 H. Store the lower byte of result obtained in R0 and higher byte in R1.

```
MOV DPTR, #8000H      ; Point DPTR to 8000H
    MOVX A, @DPTR         ; A = data from external memory [8000H]
    MOV B, A              ; Save first number in B

    MOV DPTR, #8050H      ; Point DPTR to 8050H
```

```
MOVX A, @DPTR        ; A = data from external memory [8050H]
; Now: A = second number, B = first number

MUL AB               ; Multiply A * B
                     ; Result: 16-bit → A = low byte, B = high byte

MOV R0, A            ; Store lower byte in R0
MOV R1, B            ; Store higher byte in R1

SJMP $               ; Stop program (infinite loop)
```

b. Explain the role of CALL and subroutines in 8051 microcontroller programming. Give an example.

Calls and Subroutines

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

A program that does not have to deal unexpectedly with the world outside of the microcontroller could be written using jumps to alter program flow as external conditions require. This sort of program can determine external conditions by moving data from the port pins to a location and jumping on the conditions of the port pin data. This technique is called "polling" and requires that the program does not have to respond to external conditions quickly. (Quickly means in microseconds; slowly means in milliseconds.)

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

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

Subroutines

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

Common practice when writing a large program is to divide the total task among many programmers in order to speed completion. The entire program can be broken into smaller parts and each programmer given a part to write and debug. The main program

can then call each of the parts, or subroutines, that have been developed and tested by each individual of the team.

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

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

Calls and the Stack

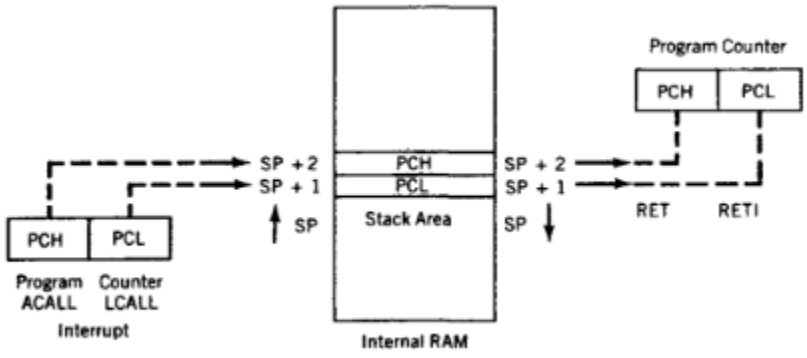
A call, whether hardware or software initiated, causes a jump to the address where the called subroutine is located. At the end of the subroutine the program resumes operation at the opcode address immediately *following* the call. As calls can be located anywhere in the program address space and used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

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

Figure 6.2 diagrams the following sequence of events:

1. A call opcode occurs in the program software, or an interrupt is generated in the hardware circuitry.
2. The return address of the next instruction after the call instruction or interrupt is found in the program counter.
3. The return address bytes are pushed on the stack, *low* byte *first*.
4. The stack pointer is incremented for each push on the stack.
5. The subroutine address is placed in the program counter.
6. The subroutine is executed.
7. A RET (return) opcode is encountered at the end of the subroutine.

Storing and Retrieving the Return Address



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

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

Calls and Returns

Calls use short- or long-range addressing; returns have no addressing mode specified but are always long range. The following table shows examples of call opcodes:

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

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

4C)

c. If the number A6H is placed in external RAM between locations 0100H and 0200H. Write an assembly language program to find the address of that location and place that address in R6 and R7 registers.

```
MOV DPTR, #0100H      ; Start address of search
    MOV R2, #01H      ; High-byte of end address (0200H → 0x02)
    MOV R3, #00H      ; Low-byte of end address (0200H → 0x00)
    MOV R4, #0        ; Found flag = 0

SEARCH: MOVX A, @DPTR     ; Read external RAM[DPTR]
    CJNE A, #0A6H, NEXT   ; Compare with A6H
    ; If equal, store DPTR into R6:R7
    MOV R7, DPH       ; High byte of address
    MOV R6, DPL       ; Low byte of address
    MOV R4, #1        ; Mark as found
    SJMP DONE
```

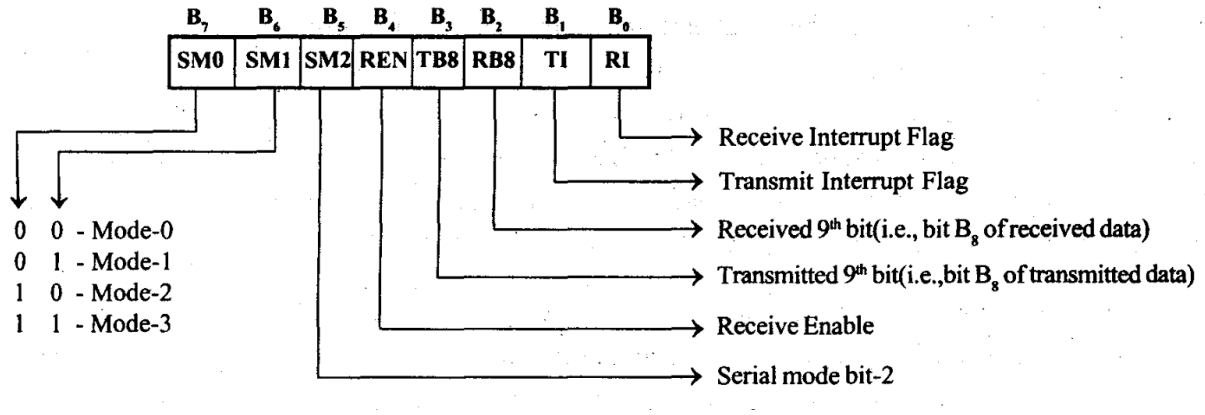
```
NEXT:   INC DPTR            ; Increment address
        MOV A, DPH
        CJNE A, #02H, SEARCH  ; If not reached 0200H high byte
        MOV A, DPL
        CJNE A, #00H, SEARCH  ; If not reached 0200H low byte

DONE:   SJMP $              ; Stop program
```

Module – 3

Q.5	a.	Explain the functions of each bit in the TMOD register.	6	L2	CO3
	b.	Explain MODE-1 programming of timers in 8051.	6	L2	CO3
	c.	Write a 8051 C program to transmit the message 'ECE' using serial communication port of 8051. Use baud rate 4800.	8	L2	CO3

5a)

The TMOD register is used to select the operating mode and the timer/counter operation of the timers. • The format of TMOD register is,



The lower four bits of TMOD register is used to control timer-0 and the upper four bits are used to control timer-1. • The two timers can be independently program to operate in various modes. • The TMOD register has two separate two bit field M0 and Ml to program the operating mode of timers. The operating modes of timers are mode-0, mode-1, mode-2 and mode-3. In all these operating modes the oscillator clock is divided by 12 and applied as input clock to timer.

MODE-0 In mode-0 the timer register is configured as 13-bit register. o For timer-1 the 8 bits of TH1 and lower 5 bits of TL1 are used to form 13-bit register. o For timer-0 the 8-bit of TH0 and

lower 5 bits of TL0 are used to form 13-bit register. o The upper three bits of TL registers are ignored. o For every clock input to timer the 13-bit timer register is incremented by one When the timer count rolls over from all 1's to all 0's, (i.e., 1 1111 1111 1111 to 0 0000 0000 0000) the timer interrupt flag in TCON register is set to one.

Mode-1

The mode-1 is same as mode-0 except the size of the timer register. In mode-1 the TH and TL registers are cascaded to form 16-bit timer register.

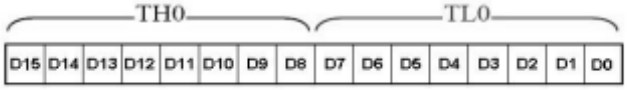
MODE-2 In mode-2, the timers function as 8-bit timer with automatic reload feature. The TL register will function as 8-bit timer count register and the TH register will hold an initial count value. o When the timer is started, the initial value in TH is loaded to TL and for each clock input to timer the 8-bit timer count register is incremented by one. o When the timer count rolls over from all 1's to all 0's (i.e., 1111 1111 to 0000 0000), the timer interrupt flag in TCON register is set to one and the content of TH register is reloaded in TL register and the count process starts again from from this initial value.

Mode-3 In mode-3, the timer-0 is configured as two separate 8-bit timers and the timer-1 is stopped. o In mode-3 the TL0 will function as 8-bit timer controlled by standard timer-0 control bits and the TH0 will function as 8-bit timer controlled by timer-1 control bits. o While timer-0 is programmed in mode-3, the timer-0 can be programmed in mode-0, 1 or 2 and can be used for an application that does not require an interrupt. o The C/T(Low) bit of TMOD register is used to program the counter or timer operation of the timer. When C/T bit is set to one, the timer will function as event counter. The C/T(Low) bit is programmed to zero for timer operation. o The timer will run only if clock input is allowed. o When GATE = 1, the clock input to timer is allowed only if the signal at pin is high and when GATE =0 the signal at INT (low) pin is ignored.

The 8051 has two timers; timer 0, timer 1. They can be used either as timers or as event counters. Both timer 0 and timer 1 are 16 bits wide. Since the 8051 has an 8-bit architecture , each 16 bit timer is accessed as two separate registers of low byte and high byte.

TIMER 0 registers

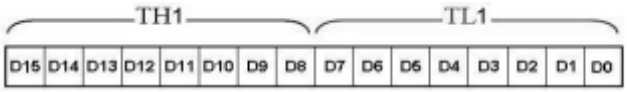
The 16 bit register of timer 0 is accessed as low byte and high byte. The low byte register is called TL0 (timer 0 low byte) and the high byte register is referred to as TH0 (timer 0 high byte). These registers can be accessed like any other register, such as A, B, R0, R1, R2 etc. For example, the instruction " MOV TL0,#25H" loads the value 25H into TL0.



Fig(1): Timer 0 Registers

TIMER 1 registers

Timer 1 is also 16 bits, and its 16 bit register is split into two bytes, referred to as TL1(timer 1 low byte) and TH1 (timer 1 high byte). These registers are accessible in the same way as the registers of timer 0.



F ig(2): Timer 1 Registers

TMOD (Timer Mode) Register

Both timers 0 and 1 use the same register, called TMOD, to set the various timer operation modes. TMOD is an 8-bit register in which the lower 4 bits are set aside for timer 0 and the upper4 bits are set aside for timer 1. In each case, the lower 2 bits are used to set the timer mode and the upper 2 bits to specify the operation. TMOD register is shown in fig(3).

5b

(MSB)							(LSB)
GATE	C/T	M1	M0	GATE	C/T	M1	M0
TIMER 1				TIMER 0			

Fig(3): TMOD Register

GATE: The T MOD register of Fig(3) that both timers 0 and 1 have the GATE bit. Every timer has means of starting and stopping. Some timers do this by software, some by hardware, and some both software and hardware controls. The timers in the 8051 have both. The start and stop of the timer are controlled by way of software by the TR (timer start) bits TR0 and TR1. This is achieved by the instructions "SETB TR1" and " CLR T R1" for timer 1 and " SETB TR0" and " CLR TR0" for time 0. The SETB instruction starts it, and it is stopped by the CLR instruction. These instructions start and stop the timers as long as GATE=0 in the TMOD register.

M1, M0: M0 and M1 select the timer mode. As show in the below Table, there are three modes; 0 ,1, and 2. Mode 0 is a 13 bit timer, mode 1 is a 16 bit timer and mode 2 is an 8-bit timer.

M1	M2	MODE
0	0	0
0	1	1
1	0	2
1	1	3

C / T (Clock / Timer): This bit in the TMOD register is used to decide whether the timer is used as a delay generator or an event counter. If C/T =0, it is used as a timer for time delay generation. The clock source for the time delay is the crystal frequency of the 8051.

Timer Programming

Mode 1 Programming

The following are the characteristics and operations of mode 1:

1. It is a 16-bit timer, therefore it allows values of 0000 to FFFFH to be loaded into the timer's registers TL and TH.
2. After TH and TL are loaded with a 16-bit initial value, the timer must be started. This is done by "SETB TR0" for Timer 0 and "SETB TR1" for Timer 1.
3. After the timer is started, it starts to count up. It counts up until it reaches its limit of FFFFH. When it rolls over from FFFFH to 0000, it sets high flag bit called TF (timer flag). This timer flag can be monitered. When this timer flag is raised, one option would be to stop the timer with the instructions "CLR TR0" for Timer 0 and "CLR TR1" for Timer 1. Notice that each timer has its own timer flag: TF0 for Timer 0 and TF1 for Timer 1.

4. After the timer reaches its limit and rolls over, in order to repeat the process the registers TH and TL must be reloaded with the original value, and TF must be reset to '0'.

Steps to Program in Mode 1

1. Load the TMOD value register indicating which timer (Timer 0 or Timer 1) is to be used and which timer mode (0 or 1) is selected.
2. Load registers TL and TH with initial count values
3. Start the timer.
4. Keep monitoring the timer flag (TF). Get out of the loop when TF becomes high
5. Stop the timer.
6. Clear the TF flag for the next round.
7. Go back to Step 2 to load TH and TL again.

Example

	MOV TMOD, #01	Time 0, mode 1 (16-bit mode)
HERE:	MOV TL0, #F2H	TL0 = F2H, the Low byte
	MOV TH0, #FFH	TH0 = FFH, the High byte
	CPL P1.5	
	ACALL DELAY	
	SJMP HERE	

Delay using Timer 0

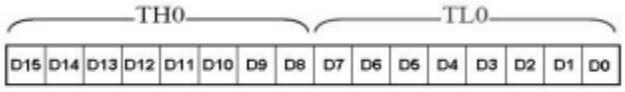
DELAY:

	SETB TR0	Start Timer 0
AGAIN:	JNB TF0, AGAIN	Monitor Timer 0 flag until ir rolls over
	CLR TR0	Stop Timer 0
	CLR TF0	Clear Timer 0 flag
	RET	

The 8051 has two timers; timer 0, timer 1. They can be used either as timers or as event counters. Both timer 0 and timer 1 are 16 bits wide. Since the 8051 has an 8-bit architecture , each 16 bit timer is accessed as two separate registers of low byte and high byte.

TIMER 0 registers

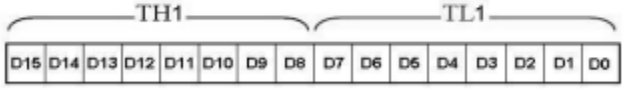
The 16 bit register of timer 0 is accessed as low byte and high byte. The low byte register is called TL0 (timer 0 low byte) and the high byte register is referred to as TH0 (timer 0 high byte). These registers can be accessed like any other register, such as A, B, R0, R1, R2 etc. For example, the instruction " MOV TL0,#25H" loads the value 25H into TL0.



Fig(1): Timer 0 Registers

TIMER 1 registers

Timer 1 is also 16 bits, and its 16 bit register is split into two bytes, referred to as TL1(timer 1 low byte) and TH1 (timer 1 high byte). These registers are accessible in the same way as the registers of timer 0.



F ig(2): Timer 1 Registers

TMOD (Timer Mode) Register

Both timers 0 and 1 use the same register, called TMOD, to set the various timer operation modes. TMOD is an 8-bit register in which the lower 4 bits are set aside for timer 0 and the upper4 bits are set aside for timer 1. In each case, the lower 2 bits are used to set the timer mode and the upper 2 bits to specify the operation. TMOD register is shown in fig(3).

e.	Write a 8051 C program to transmit the message 'ECE' using serial communication port of 8051. Use baud rate 4800.

5c)

1. Setting up the serial port

- Select Serial Mode 1: This is the preferred mode for communication with PCs, as it supports variable baud rates and 8-bit data with 1 start and 1 stop bit. This is achieved by setting the SM0 and SM1 bits of the SCON register (Serial Control Register) to 0 and 1, respectively, resulting in a value of 0x50 for SCON.
- Baud Rate Generation (using Timer 1):

- Configure Timer 1: To generate a 4800 baud rate, Timer 1 should be set in Mode 2 (8-bit auto-reload mode). This can be achieved by loading the TMOD register (Timer Mode Register) with the value `0x20`.
- Load TH1: For a 4800 baud rate with an 11.0592 MHz crystal, the TH1 (Timer 1 High Byte Register) should be loaded with a specific value, which is `0xFA` (-6 in decimal).
- Start Timer 1: The TR1 bit (Timer 1 Run Control Bit) in the TCON register (Timer Control Register) should be set to 1 to initiate Timer 1's operation.

2. Transmitting data

- Load SBUF: The character to be transmitted (in this case, 'E', 'C', or 'E') needs to be placed into the SBUF register (Serial Data Buffer Register).
- Wait for Transmission Complete: Monitor the TI flag bit (Transmit Interrupt Flag) in the SCON register. This bit will be set when the transmission of the current byte is finished.
- Clear TI Flag: Once transmission is complete, the TI flag needs to be cleared by the programmer to allow for the transmission of the next character.

3. Example code (for 8051)

c

```c
#include <reg51.h>

void main(void){
  TMOD=0x20; // Use Timer 1, Mode 2 (8-bit auto-reload)
  TH1=0xFA; // Set baud rate to 4800 for 11.0592 MHz crystal
  SCON=0x50; // Configure Serial Port Mode 1 (8-bit data, 1 start, 1 stop bit)
  TR1=1; // Start Timer 1
```

```
  // Transmit "ECE"
  SBUF='E'; // Place 'E' in the buffer
  while (TI==0); // Wait for transmission to complete
  TI=0; // Clear the TI flag

  SBUF='C'; // Place 'C' in the buffer
  while (TI==0); // Wait for transmission to complete
  TI=0; // Clear the TI flag

  SBUF='E'; // Place 'E' in the buffer
  while (TI==0); // Wait for transmission to complete
  TI=0; // Clear the TI flag

  while(1); // Infinite loop to prevent the program from ending
}
```

Use code with caution.

6. a. Explain the importance of TI flag and RI flag.TI – Transmit Interrupt Flag Meaning: TI indicates that t

he 8051 has finished transmitting (sending) one byte of data from the SBUF (Serial Buffer Register). Set automatically by hardware when the last bit (stop bit) of a frame is transmitted. Must be cleared by software after being serviced.

Importance: Tells the programmer that the UART is ready to send the next byte. Prevents overwriting of data in SBUF before the previous transmission is completed. Used in polling or interrupt-driven serial communication.

Example: After sending a byte to SBUF, you wait until TI = 1 before writing the next byte. 2. RI – Receive Interrupt Flag Meaning: RI indicates that the 8051 has received one complete byte of data into SBUF from the Rx line. Set automatically by hardware when a full frame (start, data, stop bits) is received. Must be cleared by software after reading the data.

Importance: Tells the programmer that new data is available in SBUF for reading. Ensures that the program does not attempt to read SBUF before data is valid. Used in polling or interrupt-driven reception. 👉 Example: When RI = 1, you know a new character has arrived, so you read it from SBUF. Summary (Side-by-Side): Flag Set by Meaning Importance TI Hardware Transmission complete Ready for next byte to send RI Hardware One byte received Data available to read Without TI, you wouldn't know when it's safe to send the next byte. Without RI, you wouldn't know when a byte has arrived and is ready to be read.

b. Write the steps required for programming 8051 to transmit and receive the data serially.

- **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

- **Programming the 8051 to receive 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. TH1 is loaded to set baud rate

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. RI is cleared by CLR RI instruction

6. The RI flag bit is monitored with the use of instruction JNB RI,xx to see if an entire character has been received yet

7. When RI is raised, SBUF has the byte, its contents are moved into a safe place

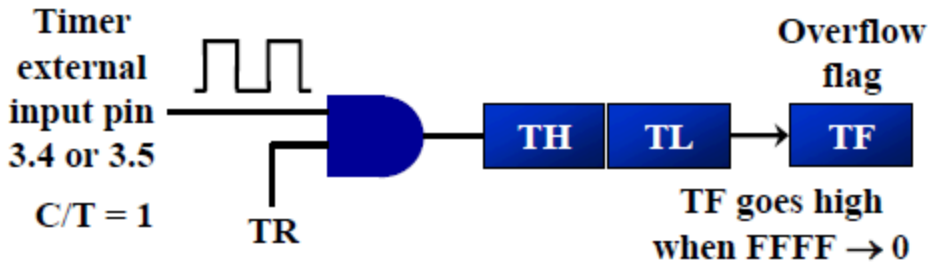
8. To receive the next character, go to step 5

C. Explain how timers are used as counters and also explain the counters operation using code snippet.

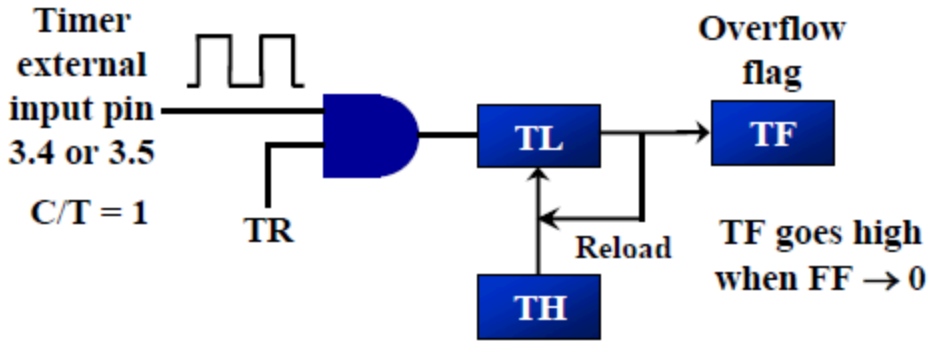
Timer as Counter

Bit C / T = 1 for counter mode.
Counter also as all modes as in Timer

Timer with external input (Mode 1)



Timer with external input (Mode 2)



Assume that a 1-Hz external clock is being fed into pin T1 (P3.5). 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 P1. Start the count at 0H.

Solution:

```
#include <reg51.h>
sbit T1=P3^5;
void main(void){
  T1=1;
  TMOD=0x60;
  TH1=0;
  while (1) {
    do {
       TR1=1;
       P1=TL1;
    }
    while (TF1==0);
    TR1=0;
    TF1=0;

  }
}
```

Module – 4

7.a. Explian the following

(i) Interrupt

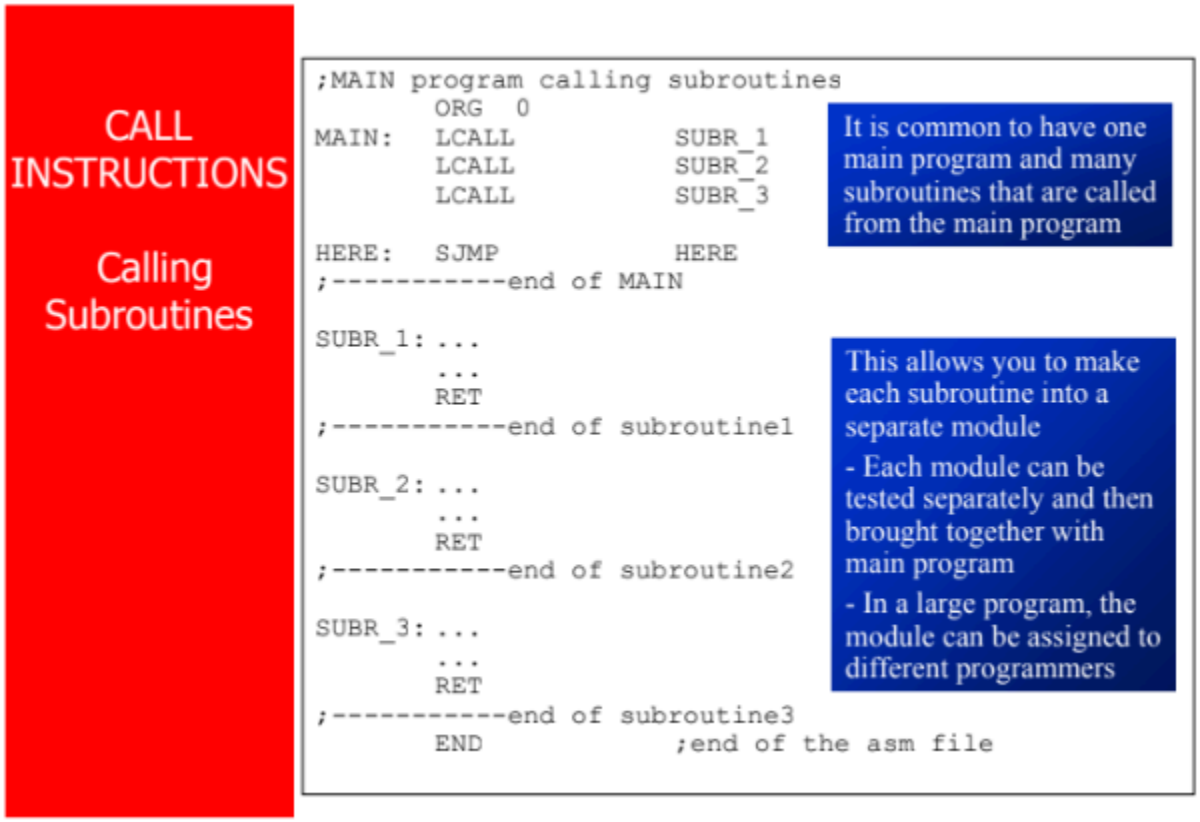
Interrupts and Returns

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

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

(ii) Interrupt Service Routine

Assembly language program examples on subroutine



(iii) Interrupt Vector Table

In the 8051 microcontroller, whenever an interrupt occurs, the CPU pauses the main program and jumps to a fixed memory location. That fixed memory address is called the Interrupt Vector Address. At each vector location, you usually place either: The Interrupt Service Routine (ISR) directly, or A LJMP instruction that points to the ISR stored elsewhere in memory. So, the Interrupt Vector Table is basically a table of fixed memory locations reserved for each interrupt.

Interrupt Vectors

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

b. Write the Instructions to

(i) Enable the serial interrupt, timer 0 interrupt and external hardware interrupt

1. IE (Interrupt Enable) Register Format

```arduino
EA   -   ET2   ES   ET1   EX1   ET0   EX0
7    6    5    4    3     2     1     0   (bit positions)
```

- **EA** – Global interrupt enable (must be set = 1 to allow any interrupt).
- **ES** – Enable Serial interrupt.
- **ET1** – Enable Timer1 interrupt.
- **EX1** – Enable External interrupt 1.
- **ET0** – Enable Timer0 interrupt.
- **EX0** – Enable External interrupt 0.
- (ET2 is for 8052 only, ignore for 8051).

2. Instructions

(i) Enable the Serial interrupt, Timer 0 interrupt, and External hardware interrupt (say External 0)

We must set **EA**, **ES**, **ET0**, **EX0** bits.

```assembly
SETB EA    ; Enable global interrupts
SETB ES    ; Enable Serial interrupt
SETB ET0   ; Enable Timer0 interrupt
SETB EX0   ; Enable External interrupt 0
```

(ii) Disable Timer 0 interrupt

(iii) Disable all interrupts with single instruction

Use bit manipulation instruction for all these cases.

(ii) Disable Timer 0 interrupt

We must clear ET0 bit.

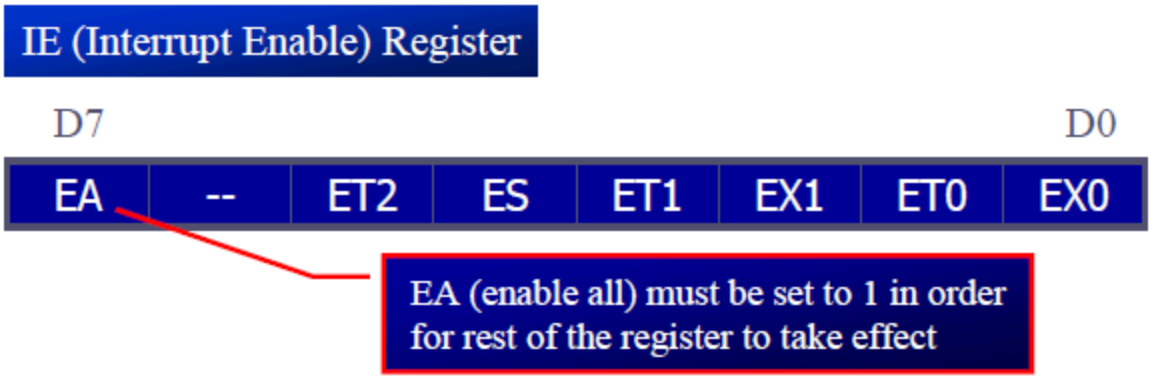
```assembly
CLR ET0    ; Disable Timer0 interrupt
```

(iii) Disable all interrupts with a single instruction

We clear **EA (global enable)**.

```assembly
CLR EA     ; Disable all interrupts
```

C. Explain the bit contents of IE register.



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. List the steps involved in executing interrupts in 8051 microcontroller.

- Upon activation of an interrupt, the microcontroller goes through the following steps

1. It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack
2. It also saves the current status of all the interrupts internally (i.e: not on the stack)
3. It jumps to a fixed location in memory, called the interrupt vector table, that holds the address of the ISR
4. The microcontroller gets the address of the ISR from the interrupt

vector table and jumps to it

It starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine which is RETI (return from interrupt)

0. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted

First, it gets the program counter (PC) address from the stack by popping the top two bytes of the stack into the PC Then it starts to execute from that address

b. Assume XTAL = 11.0592 MHz. Use timer0 to create square wave. Write a assembly program that continuously gets a 8 bit data from P(0) and sends it to P(1). While simultaneously creating square wave of 200microsec period on p2.5.

1. Problem Breakdown

- **XTAL = 11.0592 MHz**
  → Machine cycle frequency = `XTAL / 12 = 11.0592 MHz / 12 ≈ 921.6 kHz`
  → Machine cycle period = `1 / 921.6k ≈ 1.085 µs`.
- **Square wave requirement**:
  - Period = **200 µs**
  - Half period = **100 µs** (time to keep P2.5 HIGH or LOW).
- **Timer0 role**: Generate delay of **100 µs**.
- **Data transfer role**:
  - Continuously read **8-bit data from Port 0**
  - Output the same data to **Port 1**
- **Square wave role**:
  - Toggle **P2.5** after every **100 µs** delay.

So, the program will:

1. Read input from Port 0 → Write to Port 1.
2. Toggle P2.5 after 100 µs using Timer0 delay.

2. Timer0 Delay Calculation (Mode 1, 16-bit)

- Timer increments every machine cycle = **1.085 µs**.
- For 100 µs delay → Number of counts = `100 / 1.085 ≈ 92`.
- 16-bit timer max = 65536 counts.
- Initial value = `65536 – 92 = 65444 = 0xFFA4`.

So, load **TH0 = 0xFF** and **TL0 = 0xA4** for 100 µs delay.

```
ORG 0000H

MAIN:   MOV TMOD, #01H      ; Timer0, Mode1 (16-bit timer)
        SETB P2.5           ; Initialize P2.5 = 1

LOOP:   MOV A, P0           ; Read data from Port0
        MOV P1, A           ; Send data to Port1

        ACALL DELAY_100US   ; Call delay of 100 us
        CPL P2.5            ; Toggle P2.5 (square wave)

        SJMP LOOP           ; Repeat forever
```

```
;----------------------------------------
; Subroutine: 100 µs delay using Timer0
;----------------------------------------
DELAY_100US:
        MOV TH0, #0FFH      ; Load TH0 = FFH
        MOV TL0, #0A4H      ; Load TL0 = A4H
        SETB TR0            ; Start Timer0
WAIT:   JNB TF0, WAIT       ; Wait until TF0 = 1 (overflow)
        CLR TR0             ; Stop Timer0
        CLR TF0             ; Clear overflow flag
        RET

END
```

C. Write the interrupt priority upon reset in 8051. Also explain how how the priority of interrupts can be set using IP register.

1. Interrupt Priority upon Reset (Default) in 8051 When the 8051 is reset: By default, all interrupts have equal priority (lowest priority). If two interrupts occur at the same time, the 8051 services them according to a fixed polling sequence (hardware priority). Default Hardware Priority Order (Highest → Lowest): 1. External Interrupt 0 (INT0) 2. Timer 0 Interrupt 3. External Interrupt 1 (INT1) 4. Timer 1 Interrupt 5. Serial Interrupt (RI or TI)

2. Setting Interrupt Priority using IP Register

The **Interrupt Priority (IP) register** is **byte-addressable (at address B8H)** and used to assign **high or low priority** to interrupts.

IP Register Format

```
markdown

-   -   PT2   PS   PT1   PX1   PT0   PX0
7   6    5    4    3     2     1     0
```

- **PX0 (IP.0)** → Priority for External Interrupt 0
- **PT0 (IP.1)** → Priority for Timer 0
- **PX1 (IP.2)** → Priority for External Interrupt 1
- **PT1 (IP.3)** → Priority for Timer 1
- **PS (IP.4)** → Priority for Serial port
- (PT2 used only in 8052 for Timer2)

Module-5

9.a. With a neat diagram, write an assembly language program to interface stepper motor to 8051 microcontroller.

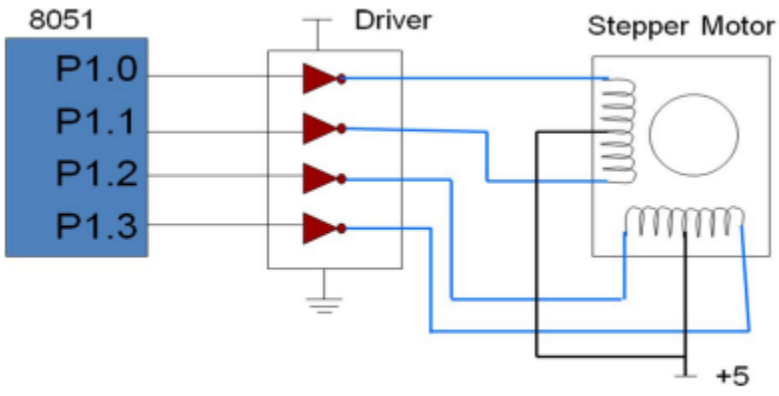


Figure 2: 8051 interfaces to stepper motor

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

b. Explain DAC interface with a diagram and write program to generate triangular waveform.

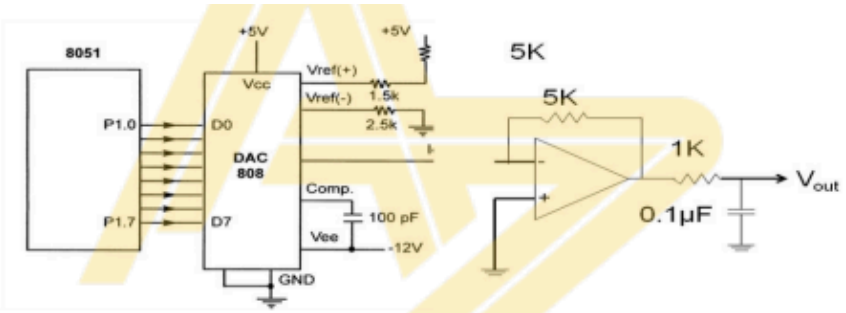


Figure 6: 8051 connections to DAC0808

The following examples 9, 10 and 11 will show the generation of waveforms using DAC0808.

Example 9: Write an ALP to generate a triangular waveform.

Program:

```
        MOV A, #00H
INCR:   MOV P1, A
        INC A
        CJNE A, #255, INCR
DECR:   MOV P1, A
        DEC A
        CJNE A, #00, DECR
        SJMP INCR
        END
```

10.a. With neat diagram, write an assembly language program to interface LCD to 8051 microcontroller.

Example 11: Write an ALP to initialize the LCD and display message "YES". Say the command to be given is :38H (2 lines ,5x7 matrix), 0EH (LCD on, cursor on), 01H (clear LCD), 06H (shift cursor right), 86H (cursor: line 1, pos. 6)

Program:

```
;calls a time delay before sending next data/command ;P1.0-P1.7 are connected to LCD data pins D0-D7
;P2.0 is connected to RS pin of LCD ;P2.1 is connected to  R/W pin of LCD ;P2.2 is connected to E pin of LCD
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
```

```
        MOV A,#'Y'              ;display letter Y
        ACALL DATAWRT           ;call display subroutine
        ACALL DELAY             ;give LCD some time
        MOV A,#'E'              ;display letter E
        ACALL DATAWRT            ;call display subroutine
        ACALL DELAY             ;give LCD some time
        MOV A,#'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
```

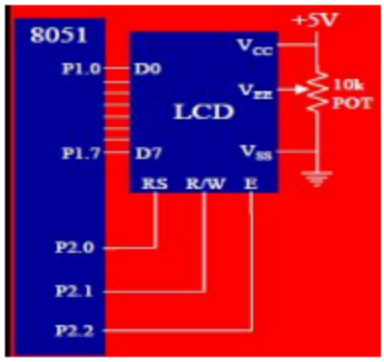


Figure : 8051 Connection to LCD

10.b. A door sensor is connected to the P1.1 pin and a buzzer is connected to P1.7. Write 8051 C program to monitor the door sensor and when it opens, sound the buzzer. The buzzer can be sound by sending a square wave of a few hundred hertz.

Frequency Calculation

Let's assume buzzer frequency ≈ **500 Hz**.

- Period = `1 / 500 = 2 ms`
- Half-period (for toggle) = `1 ms`.

  So, we need a **1 ms delay** to toggle the buzzer pin.

With **XTAL = 11.0592 MHz**,

- Machine cycle = `1.085 µs`.
- To get 1 ms → about `921 counts`.

  We can use **Timer0 (Mode 1, 16-bit)** for ~1 ms delay.

```c
#include <REG51.H>   // 8051 register definitions

sbit DOOR   = P1^1;   // Door sensor at P1.1
sbit BUZZER = P1^7;   // Buzzer at P1.7

// Function: 1 ms delay using Timer0
void delay_1ms(void) {
    TMOD = 0x01;          // Timer0 Mode1 (16-bit)
    TH0 = 0xFC;           // Load for 1ms delay (921 counts, 0xFC67 approx)
    TL0 = 0x67;
    TR0 = 1;              // Start Timer0
    while(TF0 == 0);      // Wait for overflow
    TR0 = 0;              // Stop Timer0
    TF0 = 0;              // Clear flag
}
```

```c
// Function: Generate square wave when door is open
void buzzer_on(void) {
    BUZZER = ~BUZZER;     // Toggle buzzer pin
    delay_1ms();          // Half-period delay
}

void main(void) {
    BUZZER = 0;           // Initially buzzer OFF

    while(1) {
        if(DOOR == 1) {   // Door open
            buzzer_on();  // Generate square wave
        }
        else {
            BUZZER = 0;   // Door closed → buzzer OFF
        }
    }
}
```