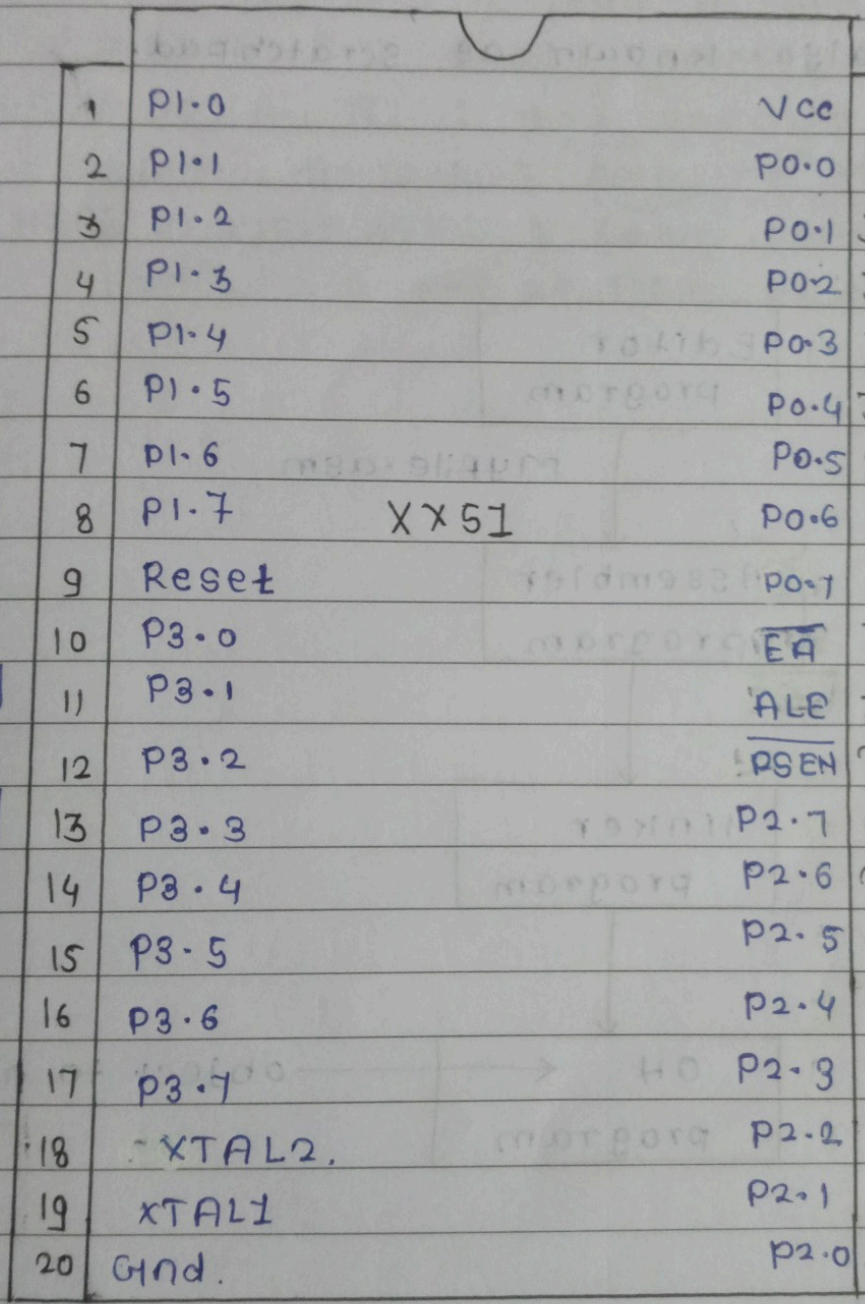
Q.3 Difference between Microcontroller and Microprocessor.

Microprocessor	Microcontroller.
• CPU is stand-alone. RAM, I/O ports, timers and other peripherals are seprate	• CPU, RAM, ROM, I/O and timers are all on a single chip.
• Designer can decide the amount of ROM, RAM and I/O ports.	• Fix amount of on-chip ROM, RAM, I/O ports.
• Expensive	• For applications in which cost, power and space are critical.
• general-purpose	• single-purpose.
• Need external peripheral to become operational.	• Can function as a computer with adding any external parts.
• Have many op-codes to move data from external memory to CPU.	• Have few op-codes.
• May have one or two types of bit handling Instructions.	• Multiple bit handling Instruction.

- Special Features 8051

- 8 bit CPU with Registers A & B.
- 16 bit Program counter and Data pointer
- 8 bit program status word and stack pointer.
- 128 bytes RAM.
- 32 I/O pins. / 4 ports - P0, P1, P2, P3.
- Two 16 bit timers / counters.
- Control Register - TCON, TMOD, SCON, PCON, IP, IE
- Two - External and 3 internal sources.
- oscillator = 11.0892 MHz.
- 16 bit DPTR.
  ↓
  DPH and DPL → To access external data.

Pin Diagram of 8051.



$\overline{EA}$:- External access input.

$\overline{WR}$ - Write to exteral. additional RAM.

$\overline{RD}$ - Read from external RAM.

XTAL1, XTAL2 - are i/p o/p pins for the oscillator.

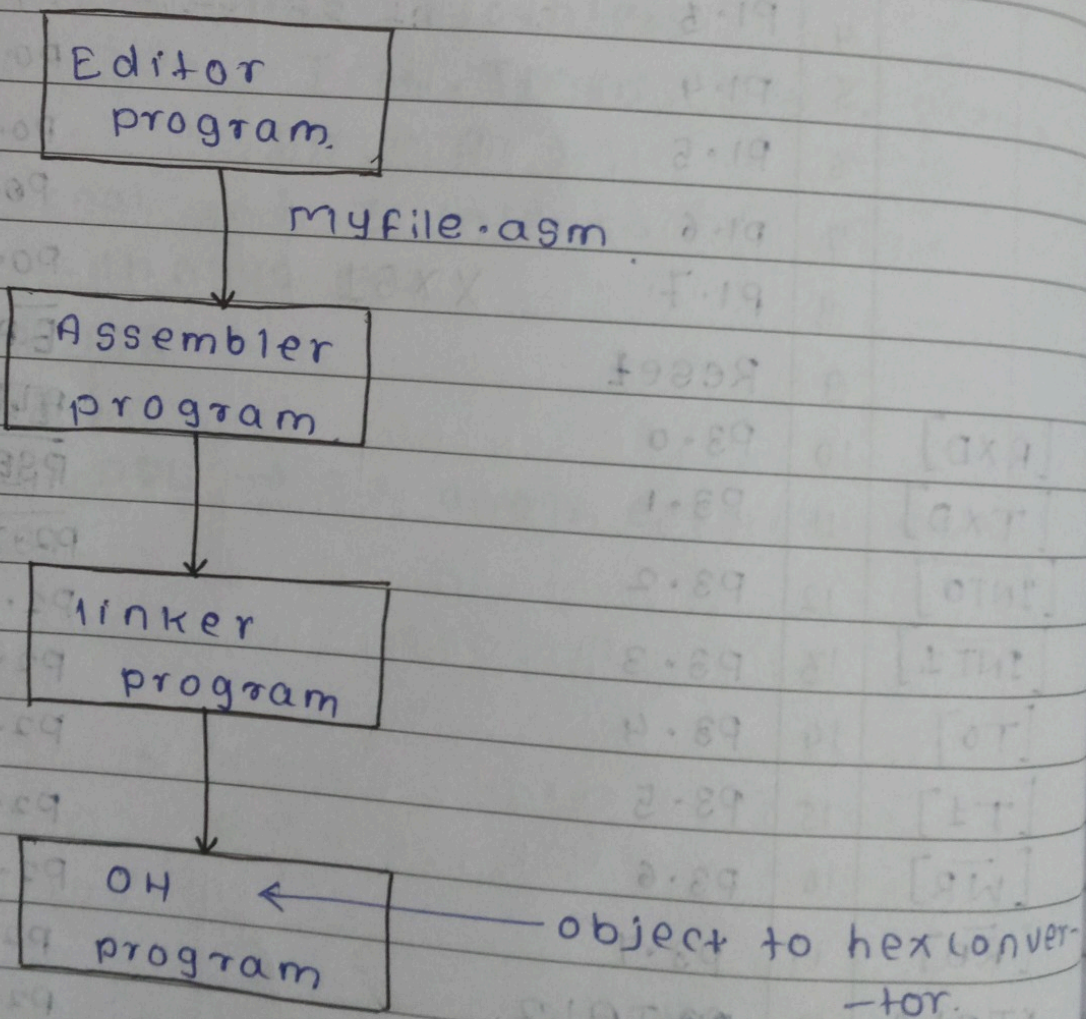
$\overline{PSEN}$:- program store enable.

ALE :- Address latch enable.

- **8051 Memory Organization**

ROM :- 4K.

RAM is also known as Scratchpad.

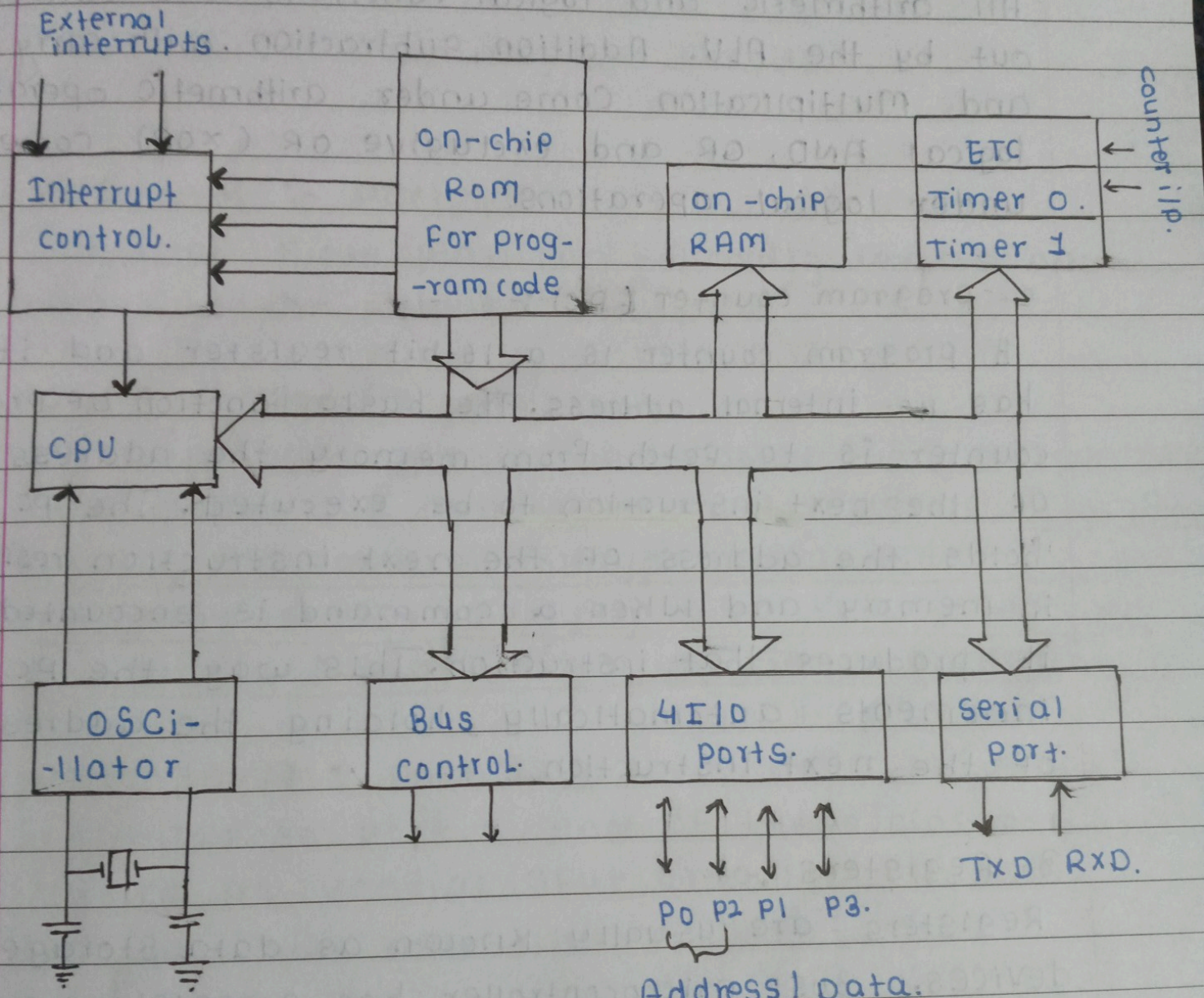


```
ORG 0H      ; Start (origin) at 0000.
MOV R5, #25H ; load 25H into R5
MOV R7, #34H. ; load 34H into R7
MOV A, #0   ; load 0 into A
ADD A, R5   ; add contents of R5 to A
                now A=A+R5.
ADD A, R7   ; Add contents of R7 to A.
                now A=A+R7
ADD A, #12H. ; Add to A value 12H.
                nowA=A+12H
HERE: SJMP HERE. ; Stay in this loop
END.        ; end of asm source file.
```

MCA Assignment :- 1.

Q. Draw block diagram Architecture of 8051 micro controller and explain program status word, stack pointer & PC.

Internal Architecture of 8051 micro-controller.



1. ALU :-

All arithmetic and logical functions are carried out by the ALU. Addition, subtraction with carry, and multiplication come under arithmetic operations logical AND, OR and exclusive OR (XOR) come under logical operations.

2. program counter (PC) :-

A program counter is a 16-bit register and it has no internal address. The basic function of program counter is to fetch from memory the address of the next instruction to be executed. The PC holds the address of the next instruction residing in memory and when a command is encounted, it produces that instruction. This way the PC increments automatically, holding the address of the next instruction.

3. Registers :-

Registers are usually known as data storage devices. 8051 microcontroller has 2 registers namely Register A and Register B. Register A serves Accumulator and while Register B functions as a general purpose register. These registers are used to store the output of mathematical and logical instructions. The operations of addition, subtraction, multiplication and division are carried out by Register. Register A also involved in data transfers between the microcontroller and external memory.

Pin configuration :-

8051 is 40 pin Integrated circuit.

Pins 1-8 :- Port 1
Each of these pins can be configured as an input or an output.

Pin 9 : RST (Reset)
A logic one on this pin disables the microcontroller and clears the contents of most registers. In other words, the positive voltage on this pin reset the microcontroller. By applying logic zero to this pin the program starts execution from the beginning.

Pins 10-17 : Port 3 :-
Similar to port 1, each of these pins can serve as general input or output. Besides, all of them have alternative functions :.

Pin 10 : RxD
Serial asynchronous communication input or serial Synchronous communication output.

Pin 11 : TxD
Serial asynchronous communication output or serial Synchronous Communication clock output.

Pin 12 : INT0 Interrupt 0 input.

Pin 13 : INT1 Interrupt 1 input.

Pin 14 : T0 Counter 0 clock input.

Pin 15 : T1 Counter 1 clock input.

Pin 16 : $\overline{WR}$
Write to external (additional) RAM.

Pin 17 : $\overline{RD}$
Read from external RAM.

Pin 18, 19 : X2 X1
Internal oscillator input and output. A quartz crystal which specifies operating frequency is usually connected to these pins. Instead of it, miniature ceramic resonators can also be used for frequency stability. later version of micro-controllers operate at a frequency of 0Hz to over 50 Hz.

Pin 20 : GND Ground :-

Pin 21-28 : Port 2
If there is no intention to use external memory then these port pins are configured as general inputs/outputs. In case external memory is used, the higher address byte, i.e. addresses A8-A15 will appear on this port. Even though memory with capacity of 64Kb is not used, which means that not all eight port bits are used for its addressing, the reset of them are not available as input/outputs.

Pin 29 : $\overline{PSEN}$ (program store enable)
If external ROM is used for storing program then a logic zero (0) appears on it every time the microcontroller reads a byte from memory.

Pin 30 : ALE (Address latch enable)
prior to reading from external memory. the microcontroller puts the lower address byte (A0-A7) on P0 and activates the ALU output. After receiving signal from the ALE pin, the external register memorizes the state of P0 and uses it as a memory chip address. Immediately after that, the ALU pin is returned its previous logic state and P0 is now used as a Data Bus. As seen, port data multiplexing is performed by means of only one additional (and cheap) integrated circuit. In other words, this port is used for both data and address transmission.

P31 : $\overline{EA}$ (External access input)
By applying logic 0 to this pin, P2 and P3 are used for data and address transmission with no regard to whether there is internal memory or not. It means that even there is a program written to the microcontroller, it will not be executed. Instead, the program written to external ROM will be executed. By applying logic one to the EA pin, the microcontroller will use both memories. First internal then External (if exists).

Pin 32 - 39 :- Port 0 :
Similar to P2, if external memory is used

these pins can be used as general input / outputs. Otherwise, P0 is configured as output (A0-A7) when the ALE pin is driven high (1) or as data output (Data Bus) when the ALE pin is driven low (0).

Pin 40 : Vcc.
+5 V power supply.

◉ PSW (Program Status Word) Register :-

PSW is also called as flag register.

CY	AC	F0	RS1	RS0	OV	—	P.

CY	PSW·7	carry flag
AC	PSW·6	Auxiliary carry flag
F0	PSW·5	Available to the user for general purpose.
RS1	PSW·4	Register bank selector 1.
RS0	PSW·3	Register bank selector 0.
OV	PSW·2	overflow flag.
--	PSW·1	user definable bit.
P	PSW·0.	parity flag set / cleared by hardware each instruction cycle, to indicate an odd/even number of 1 bits in the accumulator

RS1	RS0	Register Bank	Address.
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3.	18H - 1FH

CY (carry flag) :-

This flag is set whenever there is a carry out from the D7 bit. This flag bit is affected after an 8-bit addition or subtraction. It can also be set to 1 or 0 directly by an instruction such as "SETB C" and "CLR C" where "SETB C" is "set bit carry" and "CLR C" for "clear carry".

AC (Auxiliary carry):-

IF there is a carry from D3 to D4 during ADD or SUB operation, this bit is set; otherwise it is cleared. This flag is used by instructions that performed BCD (binary coded decimal) arithmetic.

P (parity flag) :-

The parity flag reflects the number of 1s in the A (Accumulator) register only. IF the A register contains an odd number of 1s then P=1. Therefore, P=0 IF A has an even number of 1s.

OV (overflow) :-

This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the signed bit. In general, the carry flag is only used to detect errors in unsigned arithmetic operation. The overflow flag is only used to detect errors in signed arithmetic operations.

SFR :- special Function Register.

M T W T F S S
Page No.:
Date:
YOUVA

◉ Stack pointer :- (This SFR is top of stack).
Stack is the part of RAM.

properties of Stack :-

① FIFO :- First In Last out.
② LIFO :- Last In First Out.

- There are two operation related to stack
- push :- In this operation stack pointer is incremented by 1 and then value is place into stack.
  SP ← SP+1. and now value is copied.

- pop :- In this operation First value is copied from Stack and then stack pointer is decremented by one.
  SP ← SP-1.

- By Defaul stack pointer holds 07H which is last location of Bank 0.

Q. Explain addressing modes of 8051 ?

Addressing Modes of 8051 :-

It is a way to specify the address of an operand. i.e. telling to microcontroller where is data.

There are various addressing modes.

1) Immediate addressing Mode.
2) Register addresing mode.
3) Direct addressing mode.
4) Indirect addressing Mode.
5) Index addressing mode.

- **Immediate Addressing mode :-**

In this mode data is a part of instruction operand is preceded by # hash symbol. This is used to copy data into register.
[the operand is comes immediately opcode]

Instruction :-

```
        operand
          ↓
MOV R0, # data.
↑
opcode
```

[Destination can not be immediate]

MOV DPTR, # 4321

- **Register addressing Mode :-**

In this mode operands are located in working register (Active Bank). for ex. R0 to R7, Acc or B register. DPTR (data pointer).

MOV A, R0
MOV Rd, RS.

MOV R0, DPTR → invalid. [size not match]
MOV R0, DPL → valide
MOV R1, DPH → valid
[both are 8 bit]

[Source and destination register must matched in size.
We can move data between Acc and Rn but not move data Register to ~~register~~ Rn]

This source and destination both are register and specified in opcode.

- **Direct addressing Mode :-**

In this mode the address of the data is specified in the instruction.

The source and destination can be memory location.

Ex :-

① MOV A , 20H.

② MOV A , 30H.

[Although 128 bytes of RAM can be accessed using Direct addressing mode]

Register.

- **Indirect addressing Mode :-**

In this addressing mode the address is given in one of the Register operand.

The register R0 and R1 can only be used as pointer to the memory.

This register points the memory location from which required data can be read or write. @ symbol is used in this instruction.

Ex :-

MOV A, @ R0.

MOV A , @ R1

MOV B , @ R0.

[Register indirect addressing mode, a register is used as a pointer to the pointer to the data. If the data is inside to the CPU. only register R0 and R1 are used for this purpose]

- **Index Addressing mode :-**

This addressing mode uses base register for addressing i.e. either PC or DPTR.

Index addressing mode is used to create Jump table or lookup table.

eg. MOV C A, @ A + DPTR.

The address of the source operand is calculated by adding the Contains of A with DPTR and then data is copied to Accumulator.

eg. MOVCA, @ A + PC.

The address of the source operand is calculated by adding the Contains of A with PC and then data is copied to Accumulator.

[This addressing mode is mainly used for ROM using DPTR or PC only.]

Date: YOUVA

Q.4 ■ 8051 Data types and Directives.

The 8051 microcontroller has only one data type. It is 8 bits, and the size of each register is also 8 bits. It is the job of the programmer to break down data larger than 8 bits (00 to FFH or 0 to 255 in decimal). to be process by the CPU.

DB (define byte):-

The DB directive is the most widely used data directive in the assembler. It is used to defined 8-bit data. when DB is used to defined data, the numbers can be decimal, binary, hex, or ASCII formats. For decimal the "D" after the decimal number is optional, but using "B" (binary) and "H" (hexadecimal) for the others is required. Regardless which is used, the assemble& will convert the numbers into hex. To indicate ASCII, simply place the characters in quatation marks ('like this'). The assembler will assign the ASCII code for the numbers or characters automatically. The DB used for all ASCII data definitions.

```
DATA 1    DB   28.          ; Decimal (1C in Hex)
DATA 2    DB   00110101B.   ; Binary (35 in Hex)
          DB   39H          ; Hex
          DB   "2991"       ; ASCII Number.
          DB "my name is "  ; ASCII character.
```

◉ Assembler Directives :-

The following are some more widely used directives of the 8051.

ORG (origin) :-

The ORG directive is used to indicate the beginning of the address. The number that comes after ORG can be either hex or decimal. IF the number is not followed by H. it is decimal and the assembler will convert it to hex. Some assembler use "..ORG" instead of "ORG" for the origin directive.

EQU (equate) :-

This is used to define a constant without occupying a memory location. The EQU directive does not set aside storage for a data item but associates a constant value with a data label so that when the label appears in the program, its constant value will be substituted for the label.

The following uses EQU for the counter constant and then the constant used to load the R3 register.

```
COUNT   EQU 25
.....
MOV     R3, # COUNT.
```

When executing the instruction "MOV R3, # COUNT" the Register R3 will be loaded with the value 25.

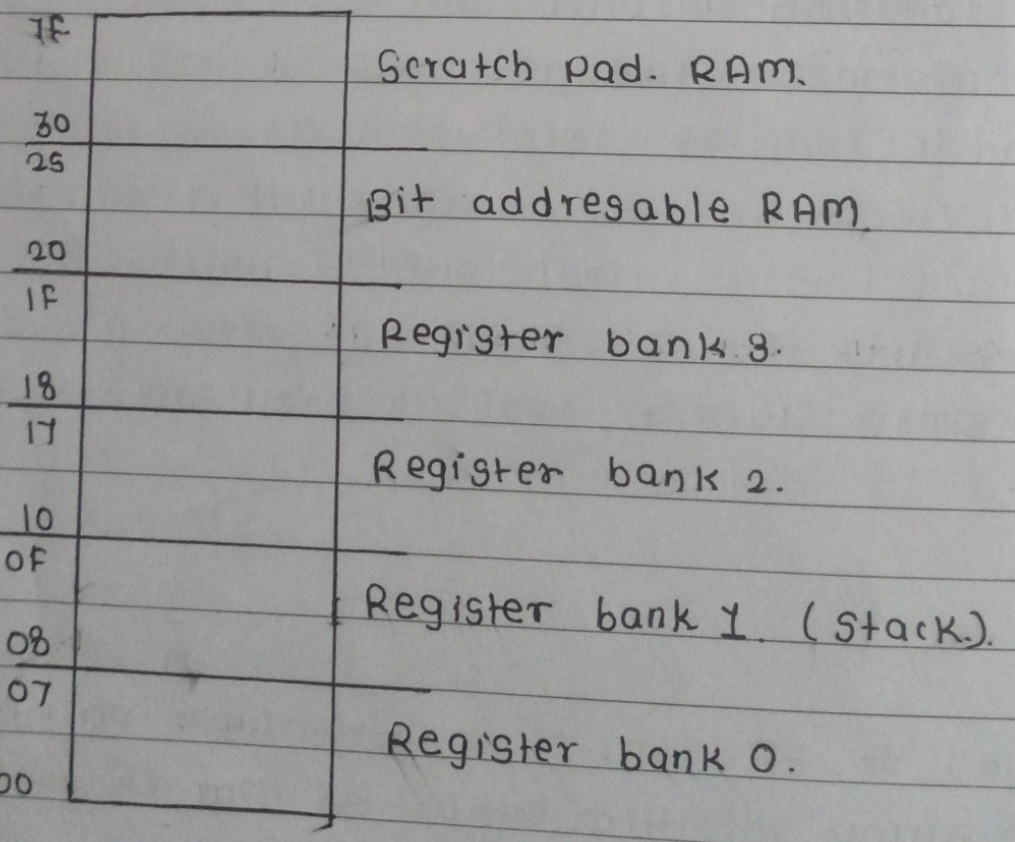
• END Directive :-

Another important pseudocode is the END directive. This indicates the assembler the end of the source (asm) file. The END directive is the last line of an 8051 program, meaning that in the source code anything after the END directive is ignored by the assembler. Some assembler use "**.END**" insted of "**END**".

Register Bank in the 8051:-

A total of 32 byte of RAM are set aside for the Register bank and stack. These 32 bytes are divided into 4 bank of Register in which each bank has 8 registers, R0-R7 RAM locations from 0 to 07 are set aside for bank 0 of R0-R7 where R0 is RAM location 0, R1 is RAM location 1, R2 is location 2 and so on.... until memory location 7, which belongs to R7 of bank 0. The second bank of registers R0-R7 starts at RAM location 08 and goes to location 0FH.

The third bank R0-R7 starts at memory location 10H and goes to location 17H. Finally, RAM location 18H to 1FH are set aside for the fourth bank of R0-R7.



default Register bank :-

IF RAM locations 00-1F are set aside for the four register banks. which register bank of R0-R7 do we have access to when the 8051 is power up? The answer is register bank 0; that is, RAM locations 0, 1, 2, 3, 4, 5, 6 and 7 are accessed with the names R0, R1, R2, R3, R4, R5, R6 and R7 when programming the 8051. It is more much easier to refer to these RAM location with names such as R0, R1 and so on.... than by their memory locations.

	Bank 0		Bank 1		Bank 2		Bank 3
7	R7	F	R7	17	R7	1F	R7
6	R6	E	R6	16	R6	1E	R6
5	R5	D	R5	15	R5	1D	R5
4	R4	C	R4	14	R4	1C	R4
3	R3	B	R3	13	R3	1B	R3
2	R2	A	R2	12	R2	1A	R2
1	R1	9	R1	11	R1	19	R1
0	R0	8	R0	10	R0	18	R0

- How to switch register banks.

Register bank 0 is the default when the 8051 is powered up. we can switch to other bank by used of the PSW. register. Bits D4 and D3 of the PSW are used to select the desired register bank.

	RS1 (PSW.4)	RS0 (PSW.3)
Bank 0	0	0
Bank 1	0	1
Bank 2	1	0
Bank 3.	1	1

The D3 and D4 bits of register PSW are often referred to as PSW.4 and PSW.3. Since they can be accessed by the bit-addressable instruction SETB and CLR for ex. "SETB PSW.3" will make PSW.3 = 1 & select bank register 1.

How stacks are accessed in the 8051

If the stack is a section of RAM, there must be registers inside the CPU to point to it. The register used to access the stack is called the SP (stack pointer) register. The stack pointer in the 8051 is only 8 bits wide, which means that it can take values of 00 to FFH. when the 8051 is powered up, the SP register contains value 07. This means that RAM location 08 is the first location used for the stack by the 8051. The storing of a CPU register in the stack is called a PUSH, and pulling the contents off the stack back into a CPU register is called a POP. In other words, a register is pushed onto the stack to save it and popped off the stack to retrive it. The job of the SP is very critical when push and pop actions are performed.

Pushing onto the Stack :-

In the 8051 the stack pointer (SP) points to the last used location of the stack. As we push data onto the stack, the stack pointer (SP) is decremented when data is pushed onto the Stack.

```
MOV  R6, #25H
MOV  R1, #12H
MOV  R4, #0F3H.
PUSH 6.
PUSH 1
PUSH 4
```

Solution :-

	After push 6.	After push 1	After push 4.
0B	0B	0B	0B
0A	0A	0A	0A F3
09	09	09 12	09 12
08.	08 25	08 25	08 25.
Start SP=07	SP=08	SP=09	SP=0A

- **Popping from the stack.**

  popping the contents of the stack back into a given register is the opposite process of pushing. With every pop, the top byte of the stack is copied to the register specified by the instruction and the stack pointer is decremented once.

The upper limit of the stack.

locations 08 to 1F in the 8051 RAM can be used for the stack. This is because locations 20-2FH of RAM are reserved for bit addresable memory and must not be used by the stack. If in a given program we need more than 24 bytes (08 to 1FH = 24 bytes) of stack, we can change the SP to point to RAM locations 30-7FH. This is done with the instruction "mov SP, #xx"

```
pop 8;
pop 5;
pop 2;
```

	After pop 3	After pop5	After pop2.
0B 54	0B	0B.	0B
0A F9	0A F9	0A	0A
09 76	09 76	09 76	09
0B 6C	08. 6C.	0B 6C	0B 6C
Start SP 0B.			

Q.5 ◉ Explain Following Arithmetic Instructions.

SUBB (Subtract with borrow) when Cy = 1.

This instruction is used for multibyte numbers numbers and will take care of the lower operand. IF cy = 1 prior executing the SUBB instruction, it also subtracts 1 from the result.

For example :-

```
CLR C
MOV A, #4CH  ; load A with value 4CH  (A = 4cH)
SUBB A, #6EH ; Subtract 6E from A.
JNC NEXT     ; IF Cy = 0 jump to NEXT target
CPL A        ; IF cy = 1 then take 1'S complement.
INC A        ; and increment to get 2'S complement.
NEXT : MOV R1, A ; gave A in R1.
```

Solution :-

Following steps ~~for~~ for. ie SUBB A', #6EH.

```
  4C      0100 1100
- 6E      0110 1110
-22
```

2'S complement.

```
0100 1100
1001 0010
-------------
0 1101 1110.
```

= 1, the result is negative in 2's complement

MUL (Multiplication of Unsigned numbers)

The 8051 supports byte-by-byte multiplication, only. The bytes are assumed to be unsigned data. The syntax is follows.

MUL AB ; A X B, place 16-bit result in B and A.

In byte-by-byte multiplication, one of the operands must be in register A, and the second operand must be in register B. After multiplication, the result is in the A and B registers ; the lower byte is in A, and upper byte is in B. The following example multiplies 25H by 65H. The result is a 16-bit data that is held by the A and B registers.

```
MOV A, #25H   ; load 25H to reg. A.
MOV B, #65H   ; load 65H in reg. B.
MUL AB.       ; 25H * 65H = E99 where
              ; B = 0EH and A = 99H.
```

- DIV (Division of Unsigned numbers)

In the division of unsigned numbers, the 8051 supports byte over byte only. The syntax is follows.

```
DIV AB.  ; divide A by B.
```

When dividing a byte by a byte, the numerator must be in register A and denominator must be in B. After the DIV instruction is performed, the quotient is in A and the reminder is in B.

Unsigned Division summary (DIV AB).

Division byte/byte.	Numerator A	Denominator B	Quotient A	Remainder B.

(IF B=0 then OV = 1 indicating an error).

```
MOV A, #95   ; load 95 into A.
MOV B, #10   ; load 10 into B.
DIV AB.      ; now A = 09 (quotient) and
             ; B = 05 (remainder).
```

Notes:-

1. This instruction always makes CY=0 and OV=0 if the denominator is not 0.
2. IF the denominator is 0 (B=0), OV=1 indicate an error, and CY=0. The standard practice in all microprocessor when dividing a number by 0 is to indicate in some way the invalid result of infinity. In the 8051, the OV flag is set to 1.

- DA instruction :-

The DA (decimal Adjust For addition) instruction in the 8051 is provided to correct the aforementioned problem Associated with BCD addition. The mnemonic "DA" has as its only operand the accumulator "A" The DA instruction will add 6 to the lower nibble if needed; otherwise, it will leave the result alone.

```
MOV A, #47H  ; A=47H first BCD operand.
MOV B, #25H  ; B=25  second BCD operand.
ADD A, B     ; hex (binary) addition (A=6CH)
DA  A        ; adjust for BCD addition (A= 72H)
```

After the Execution of program, Register A will contain 72H (47+25=72). The "DA" instruction works only on A.

Summary of DA action.

After an ADD or ADDC instruction.

1. If the lower nibble (4 bits) is greater than 9, or if AC=1, add 0110 to the lower 4 bits.

2. If the upper nibble is greater than 9, or if CY=1, add 0110 to the upper 4 bits.

Hex	BCD.
29	0010 1001
18	0001 1000
41	0100 0001
6	0110
47	0100 0111.

AC = 1.

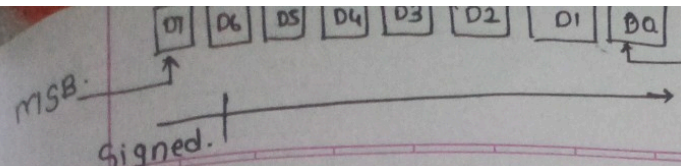
Since, AC=1 after the addition, "DA A." will add 6 to the lower nibble. The final result is in BCD format.

- ADDC and adition of 16-bit numbers.

When adding two 16-bit data operands, we need to concerned with the propogation of carry from the lower byte to the higher byte. The instruction ADDC (add with carry) is used on such occasion. for example. ~~look~~

```
  3C E7          00111100 11100111
+ 3B 8D        + 00111011 10001101
-------        -------------------
  78 74          01111000 01110100
```

when the first byte is added (E7+8D = 74, Cy=1) The carry is propogated to the higher, which result in 3C+3B+1 = 78 (all in hex).



M T W T F S S
Page No.:
Date:
YOUVA

- Signed number concepts and arithmetic operations.

- Signed 8-bit operands.

In signed byte operands, D7 (MSB) is the sign and D0 to D7 are set aside for the magnitude of the number. IF D7=0, the operand is positive, and if D7=1. It is negative.

- positive numbers :-

The range of positive numbers that can be represented by the format is 0 to +127 if a positive number is larger than +127, a 16-bit size operand must be used. since 8051 does not support 16 bit data.

- Negative numbers :-

For negative numbers D7 is 1; however magnitude is represented in its 2's complement. Although the assembler does the conversion, it is still important to understand how the conversion works. To convert to nega-tive number representation (2's complement).

1. write the magnitude of the number is 8-bit binary. (no sign)
2. Inverted each bit.
3. Add 1 to it.

- **INC**

Increment the operand by 1.

- The operand can be register, a direct address, an indirect address, the data pointer.

Example:-
```
INC A       ; increment Accumulator by 1.
INC Rn      ; Increment Register.
INC direct  ; increment direct byte.
```

- **DEC.**

-Decrement the operand by one.

- The operand can be a register, a direct address, an indirect address.

Ex -
```
DEC A       ; Decrement Accumulator
DEC Rn      ; Decrement Register
DEC direct  ; Decrement direct byte.
DEC @ Ri    ; Decrement Indirect RAM
DEC DPTR    ; Decrement Data pointer by 1.
```

```
INC @ Ri    ; Increment Indirect RAM.
INC DPTR    ; Increment Data pointer by 1.
```

When is the OV flag is set ?

In 8-bit signed number operation, OV is set to 1 if either of the following two condition occurs.

1. There is carry from D6 to D7 but no carry out of D7 (cy=0).
2. There is carry from out (cy=1) but no carry from D6 to D7.

the overflow flag is set to 1 if there is a carry from D6 to D7 or from D7 out, but not both. This means that if there is a carry both D6 to D7 and D7 out, OV=0.

$$\begin{array}{r} -128 \\ +\quad -2 \\ \hline -130 \end{array} \qquad \begin{array}{r} 1000\ 0000 \end{array}$$

Logic and compare instructions.

AND ANL A, #0FH.

ANL destination, source.

MOV A, #35H

ANL A, #0FH.

Inputs		output
A	B	y
0	0	0
0	1	0
1	0	0
1	1	1

Solution :

35H	0011	0101
0FH	0000	1111
05H.	0000	0101

ANL A, #50H.

ANL A, R1

ANL A, 17H

ANL A, @R1

ANL 25H, A.

ANL 25H, #50H

OR

ORL Destination, source ; dest OR source.

MOV A, #04 ; A = 04

ORL A, #30H. ; A = A OR 30H

Inputs		output
A	B	y
0	0	0
0	1	1
1	0	1
1	1	1

Solution :-

04H	0000	0100
30H	0011	0000
34H	0011	0100

04H OR 30H = 34H.

ORL A, #50H.

ORL A, R1.

ORL A, 17H

ORL A, @R1

ORL 25H, A

ORL 25H, #50H.

- **XOR.**

XRL destination, source. ; dest = dest xOR Source.

```
mov A, #54
ORL A, #78H.
```

Solution :-

```
54H     0101 0100
+ 78H   + 0111 1000
          1111
        1101 1100
```

Inputs		output.
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

```
XRL A, #50H
XRL A, R1
XRL A, 17H
XRL A, @R1
XRL 25H, A
XRL 25H, #50H
```

- **CPL A** (Complement accumulator)

This instruction Complements the contents of Register A. The complement action changes the 0s to 1s and 1s to 0s. This is also called 1's complement.

```
  00011011
→ 11100100 → CPL Form
```

- **Compare instruction :-**

The 8051 has an instruction for the compare operation.

CJNE destination, source, relative address.

In 8051, the action of comparing and jumping are combi-ned into a single instruction called CJNE (compare and jump if not equal). The CJNE instruction compare two operands, and jump if they are not equal.
In addition it changes the CY flag to indicate if the destination operand is larger or smaller.

```
MOV A, #55H.
CJNE A, #99H, NEXT
   .....
   ......
NEXT:
```

Solution :-

a) yes, it jumps because 55H and 99H are not equal.

b) A = 55H, its original value before the compa-rision.

Carry flag setting for CJNE Instruction.

Compare.	Carry flag.
destination ≥ source	CY = 0
destination ≤ source	CY = 1.

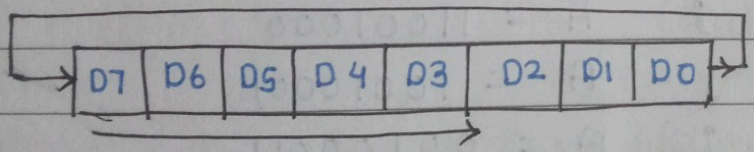
Rotating instruction and data serialization.

There is an need to perform a bitwise rotation of an operand. In the 8051 the rotation instructions RL, RR, RLC and RRC are designed specifically for that purpose. They allow to rotate accumulator right or left.

- RRA ; Rotate Right A.

In rotate right, the 8 bits of the accumulator are rotate right one bit, and bit D0 exist from the least significant bit and enters into D7 (MSB)

```
MOV A, #36H     ; A = 0011 0110.
RR A            ; A = 0001 1011
RR A            ; A = 1000 1101
RR A            ; A = 1100 0110
RR A            ; A = 0110 0011
```

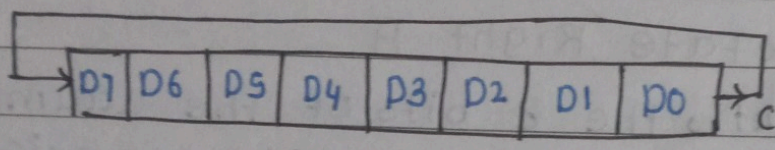


- Rotating through the carry

RRC A Rotate Right throug carry.

In RRC, as bits are rotate from left to right they exist the LSB to the carry flag, and the carry flag enters the MSB. In other words, in RRC A the LSB is moved to (carry) CY and CY is moved to the MSB. In reality the carry flag acts as if it is part of Register A, making it a 9-bit register.

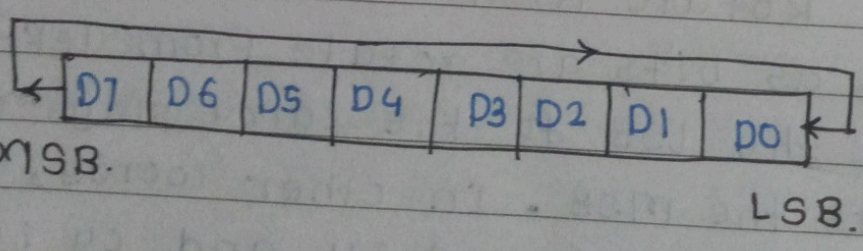
```
CLR C ; make cy =0
MOV A , 26H ; A = 00100110
 RRC A ;        A = 00010011   cy =0
 RRC A ;        A = 10001001   cy =1.
 RRC A ;        A = 11000100   cy =1
```



- **RL A** — Rotate left Accumulator.

In Rotate left the 8 bits of the accumulator are rotated left one bit, and bit D7 exist from the MSB and enters into D0 (LSB).

```
MOV A , #72H ; A=0011 0010
 RL. A      ;    A = 01100100
 RL A       ;    A = 11001000
 RL A       ;    A = 10010001
RL A        ;    A = 00100011
```



RLC A Rotate left through the carry.

As bits are shifted from Right to left they exit the MSB and enter the carry flag, and the carry flag enters the LSB. In other words, in RLC the MSB is moved to cy (carry flag) and cy is moved to the LSB.

```
SETB C          ; make cy=1.
MOV A, #15H     ; A = 0001 0101.
RLC A ;         A = 00101010   cy=0
RLC A ;         A = 01010100   cy=0
RLC A ;         A = 10101000   cy=0
RLC A ;         A = 01010000   cy=1.
```

Single bit instruction with carry.

Instruction	Function.
SETB C	Make cy=1
CLR C	clear carry bit (cy=0)
CPL C	Complement carry bit.
MOV b, C	copy carry status to bit location (cy=b)
MOV C, b	copy bit location status to carry (b=cy)
JNC target	jump to target if cy=0
JC target	jump to target if cy=1
ANL C, bit	AND cy with bit and save it on cy.
ANL C, /bit	AND cy with inverted bit and save it on cy.
ORL C, bit	OR cy with bit and save it on cy.
ORL C, /bit.	OR cy with inverted bit and save it on cy.

- **SWAP A.**

SWAP instruction works only on the accumulator (A). It swap the lower nibble and the higher nibble. In other words, the lower 4 bits are put into the higher 4 bits, and the higher 4 bits are put into the lower 4 bits.

before :-

D7 - D4	D3 - D0
0010	1100

after SWAP :-

D3 - D0	D7 - D4
1100	0010

- **Boolean instruction in 8051.**

A> set, clear and complement carry

SETB C ⟶ flag bit carry ⟶ C = 1
CLR B C ⟶ clear carry bit ⟶ C = 0
CPL C ⟶ Complemet flag bit carry ⟶

B> set, clear and complement bit instructions.

```
SETB  P0.2  ;   P0.2  ← 1
CLR B0.2    ;   P0.2  ← 0
CLR  06H    ;   06H   ← 0

CPL  P0.5   ;  P0.5  ← complement OF p0.5
CPL  05H    ;  05H   bit location complement.
```

- Move bit instruction.

```
MOV C, P0.2    ; carry flag ← P0.2
MOV P0.2, C    ; P0.2 ← carry.
MOV 07H, C     ; 07H ← carry.
```

- Logic AND, OR.

```
ANL C, P0.2    ; C ← C AND P0.2.
ANL C, 07H.    ; C ← C AND 07H.
ANL C, /b      ; C And {not} P0.2.
ANL C, /P0.2   ; C And {not}
```

OR logic.

```
ORL C, P0.1    ; C ← C or P0.1.
ORL C, 07H     ; C ← C or 07H.
ORL C, /b      ; b = bit.
ORL C, /P0.2.  ; C ← C or not P0.1.
ORL C, /05H    ; C ← C OR not 05H.
```

◉ Branch instruction.

	SJMP Short Jump	AJMP Absulate Jump	LJMP. long Jump.
Syntax :-	SJMP label1	AJMP label 2	LJMP label 2.
Range :-	-128 to +127	Maximum Range 2KB	Maximum Range 64 KB.
Size :-	2 Bytes [1 byte opcode, 1 byte label]	2 bytes. [1 byte opcode & 1 byte label]	[3 byte 1 byte opcode 2 byte label]
Address calculation.	PC of next instruction label.	PC = 1st 5 bit address of PC + 8 byte For AJMP and 8 bits of label.	PC = 16 bit address labels.
Instruction	SJMP All conditional Jump.	AJMP and ACALL	LJMP & LCALL.
use :-	Same prog- -ram	within the same range of 2KB.	use to jump anywhere in 64 KB in 8051.

Date: YOUVA

Loop and jump instruction :

- Byte jump

1) CJNE A, add, radd ; Compare the contents of A register with the contents of the direct address. if they are not equal, then jump to relative address set the carry flag to 1. if A < the contents of the direct address otherwise set carry flag to 0.
[jump if A ≠ Data]

2) DJNZ Rn, radd ; Decrement Rn by 1 and jump to the relative address. if the result is not zero.
[Decremen and jump if register ≠ 0]

3) JZ radd. ; Jump to relative address if A = 0.

4) JNZ radd ; Jump to relative address if A ≠ 0.

Programs :-

- Addition of two 8-bit

```
MOV  R4, #20H   ; R4=20H
MOV  R2, #21H   ; R2 = 21H.
MOV  A , R2     ; Accumulator = 21H
MOV  A , R4     ; A = A+R4
                ; A = 21+20
End.
```

OR.

```
MOV A, #20H   ; A=20H
ADD A, #21H   ; A = A+21H.
MOV R4, A     ; MOV Accumulator data ~~from~~ into R4.
```

For ROM memory.

```
MOV DPTR, #2000H.
MOV A, #40H
ADD A, #20H
MOV@DPTR , A   ; gave result at 2000H.
```

MOV C ← for RAM.
MOV X ← for external Memory

- Write a program to add value 9 to the Accumulator 10 times.

```
ADD A, #09 ;
```

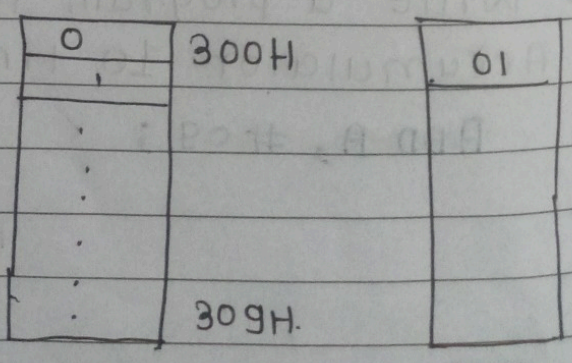
```
ORG 0000H
MOV A, #00 ;  Acc =00
MOV R2,#10 ; Set counter of 10 count in R2.
LOOP1:ADD A,#09 ;  09 + Acc.
DJNZ R2, loop1 ; Decrement R2 register
MOV R5, A      ;
END            ; End program.
```

- Write a program to load Accumulator with value 55H. and complement the Accumulator 700 times... [Nested loop used....]

```
ORG 0000H      ; Start program for 000DH
MOV A ,#55H    ;  A = 55H.
MOV R3, #10    ; for external loop. (R3=10)
Loop2: MOV R2, #70 ;  for internal loop. (R2=70)
Loop1: CPL A       ; Complement Accumulator.
DJNZ R2 Loop 1 ; if is internal loop.
DJNZ R3 Loop 2 ; for external loop.
END.
```

- Write a program to find square of Numbers 0 to 9 given in a RAM location. used look up table to find the square of number.

```
ORG 0000H
MOV DPTR, # 300H
MOV A , 60H
MOVC A , @ A+DPTR
MOV R5 , A
```



- Find the Sum of the values 79H, F5H, E2H put the sum in Registers R0 (lower byte) and R5 (Higher byte)

```
    MOV A,#00
    MOV R5,A
    ADD A,#79H
    JNC LOOP1
    INC R5
LOOP1: ADD A,#0F5H
    JNC LOOP2
    INC R5
LOOP2: ADD A,#0E2H
    JNC LOOP3
    INC R5
LOOP3: MOV R0,A
```

- Write a program to add 16 bit numbers :-

```
ORG 0000H      :- start program for 0000H
MOV A,#0E7H ; load the low byte A = 0E7H
ADD A,#8DH  ; add the low byte, A=74H & CY=1.
MOV R6, A      ; Save the low byte in R6.
MOV A,#3CH  ; load the high byte.
ADDC A,#3BH ; add with the carry.
MOVR7, A.      ; save the high byte of the sum
```

- Write a program to add two 8 bit numbers with carry or.
  Write a program to add two 8bit numbers if sum is greater than 8 bits.

```
MOV R0 , #20H
MOV R1 , # 40H
MOV R4 , #00
MOV A , @ R0
INC R0
ADD A , @ R0
JNC LOOP
INC R4
LOOP: MOV @ R1, A
INC R1
MOV @ R1, R4
```

- Multibyte Addition.
  Add two 16 bit number.

```
MOV R2 , #00H
MOV A , 20H
Add A , 22H
MOV 24H , A
MOV A , 21H
ADDC A , 23H
JNC LOOP
INC R2
Loop MOV 25H, A
MOV 26H , R2
HERE : SJMP HERE
END.
```

- Write a program to store nibble of R7 to both nibbles of R6.

```
MOV A, R7          ;  ACC = 1010 0101
ANL A, #0F0H       ;  ACC = 1010 0000
MOV R6, A          ;  R6 = 1010 0000
SWAP A             ;  ACC = 0000 1010
ORL A, R6          ;  ACC = 10101010
MOV R6, A          ;  = 10101010.
```

- Find the time period of machine cycle for 8051 based system having

1> 11.0592 MHz

T = 8.

$F = \frac{11.0592 \text{ MHz}}{12}$

$F = 921.6 \text{ KHz}$

$T = \frac{1}{F} = \frac{1}{921.6 \text{ KHz}}$

$T = 1.085 \ \mu\text{sec.}$

2> 16 MHz

$F = \frac{16 \text{ MHz}}{12}$

$F = 1.33 \text{ MHz.}$

$= 1.33 \times 10^6$

$T = \frac{1}{F}$

$T = \frac{1}{1.33 \times 10^6}$

$= 10^{-3} \text{ Hz.}$

921.6

T = 0.75 μsec.

3> 20 MHz

$F = \frac{20 \text{ MHz}}{12}$

$F = 1.66 \text{ MHz.}$

$T = \frac{1}{F} = T = 0.60 \mu\text{sec.}$

- For 8051 system of having freq 11.0592 MHz find time delay for following subroutin.

```
DELAY  MOV R3, # 250
HERE : NOP
       NOP
       NOP
       NOP
       DJN2 R3, HERE
       RET
```

$T = [250(2+4)+1+2] \times 1.0852$

$= 1630.755 \mu sec.$

- For 8051 system of 11.0592 MHz. & find ~~how many long it~~ How long it takes to execute each instruction.

(a)

```
MOV R3, #55
DEC R3
DJNE R2.
```

- **8051 Versions :-**

Chip	clock per machine cycle.
1) AT89C51 Atmel	12
2) P89C64X2 philips	6
3) DS5000 Dallas Semi	4
4) DS89C420/30/40	1.

Write a program to toggle all the bits of port 1 by sending to it values 55H and AAH continu-ously. put a time delay between each issuing of data to port one.

```
    ORG 0000H
BACK: MOV A, #55H
      MOV P1, A
      LCALL DELAY
      MOV A, #0AAH
      MOV P1, A
      LCALL DELAY
      SJMP BACK.
      ORG 300H
DELAY MOV R5, #0FFH
AGAIN DJNZ R5, AGAIN
      RET
      END.
```

OR

```
    ORG 0000H
Back: MOV A, #55H
BACK: MOV P1, A
      ACALL DELAY
      CPL A
      SJMP Back
```

- The 4 eight bits I/O ports i-e P0, P1, P2, P3 uses 8 pins. All the ports open reset configured as input means ready to be used as input port.

When 1st ~~P0~~.0 is return to a port ~~output port.~~ it will be an o/p.

2. ~~1 is~~ TO reconfigure it as input a one must send to the port.

- Write a program to that will continuously send the value 55H and AAH to the port 0. providing a delay to issue at P0.

Find the cycle of following program if the crystal frequency is 11.0592.

	M/C.
DELAY: MOV R2, #200	1 HERE loop: 250(2+1+1) × 1.085 = 1085 μsec
AGAIN: MOV R3, #250	1 AGAIN: 200(HERE) = 200 × 1085 μsec. = 2/7000 μsec
HERE: NOP	1 outside loop = (2+1) × 1.085 = 3.25.
NOP	1
DJNZ R3, HERE	2
DJNZ R2, AGAIN	2
RET	2

Find the time delay for loop section of the following subroutine if it is run DS89C420 / 30 chip. 11.0592 MHz.

$T = \frac{1}{f}$ = 0.090422 μsec ≈ 90 μsec.

	M/C.
MOV R3, #250	1
HERE: NOP	1 t_d = (4+1+1+1+1) × 90 μsec. = 18000 ns
NOP	1
NOP	1
NOP	4
DJNZ R3, HERE	
RET.	

Write a program to blink all the LEDS are connected to P1. at ~~sloperate~~ slow rate delay so that blinking clearly seen assume the freq of 22MHz & that the system is using AT89C51.

```
    MOV A, #0FFH.
AGAIN: MOV P1, A.
       ACALL DELAY
       CPL A
       SJMP AGAIN


DELAY:
       MOV R2, #7
HERE1: MOV R1, #255
HERE2: MOV R0, #255
HERE3: DJNZ R0, HERE3
       DJNZ R1, HERE2
       DJNZ R2, HERE
```

Page No.:
Date:
YOUVA

- Write a program for DS89c420 to toggle all the bits of P0, P1, P2 every $\frac{1}{4}$ of second.

Assume crystal frequency of 11.0592 MHz.

```
      ORG 0000H
BACK: MOV A, #55H
      MOV P0, A
      MOV P1, A
      MOV P2, A
      ACALL @SDELAY
      MOV A, #0AAH
      MOV P0, A
      MOV P1, A
      MOV P2, A
      ACALL @SDELAY
      SJMP BACK.
```

$[11 \times 240 \times (255)] \times 4 \times 90 \text{nsec.}$

$= 9.25$

- Write a program to send port 0 alternating values of 55H and AAH continuously:-

```
      MOV A, #55H
BACK: MOV P0, A
      ACALL DELAY
      CPL A
      SJMP BACK.
```

- Make P0 as input port 0 and Send port 0 data to P1 port :-

```
MOV A, #0FFH
MOV P0, A
BACK: MOV A, P0
      MOV P1, A
      SJMP BACK.
```

* Make P1 port as input port and Saved data from P1 to R7, R6, and R5.

```
MOV A, #0FFH
MOV P1, A
MOV A, P1
MOV R7, A
ACALL DELAY
MOV A, P1
MOV R6, A
ACALL DELAY
MOV A, P1
MOV R5, A.
END.
```

Make P2 part as an input and send data from P2 to P1.

```
ORG 000H.
MOV A, #055H
MOV P2, A
BACK: MOV A, P2
      MOV P1, A
      SJMP BACK.
```

To toggle single point of P1.2 continuously :-

```
ORG 0000H
SETB P1.2
ACALL DELay.
CLR P1.2
ACALL DELAY
  SJMP AGAIN
```