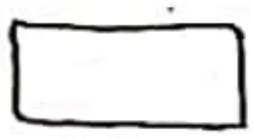
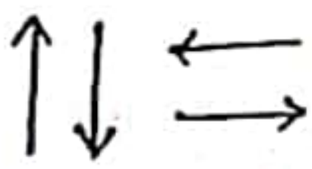
3. 8051 programming concepts — 19 marks

3.1. Various symbols used in drawing flow charts (1)

→ A flowchart is a visual or graphical representation of an algorithm. The following symbols are used in preparation of a flowchart.

	Symbol	function
1)	 predefined process	Indicates any type of internal operation inside the processor or memory.
2)	 Input/ output	Used for any input/output operation. Indicates that the computer is to obtain data or output results.
3)	 decision	Used to ask a question that can be answered in binary format
4)	 Connector	Allows the flowchart to be drawn without a reverse flow.
5)	 Predefined process	Used to invoke a subroutine or an interrupt program.
6)	 terminal	Indicates the starting or ending of the program, process & interrupt program.
7)	 Flow line	shows direction of flow

3.2 Program to illustrate the application of data copy instructions.

⇒ Program to copy the value 36H into 50H to 55H memory location by using direct and indirect addressing mode with and without a loop.

1) Without loop :-

```
MOV R0, #50H ;[ load R0 with 50H]
MOV A , #36H ;[ move 36H into A]
MOV @R0, A   ;[ move 36H into 50H]
INC R0       ;[ increment R0] 51H
MOV @R0, A   ;[ move 36H into 51H]
INC R0       ;[ increment R0] 52H
MOV @R0, A   ;[ move 36H into 52H]
INC R0       ;[ increment R0] 53H
MOV @R0, A   ;[ move 36H into 53H]
INC R0       ;[ increment R0] 54H
MOV @R0, A   ;[ move 36H into 54H]
INC R0       ;[ increment R0] 55H
MOV @R0, A   ;[move 36H into 55H]
END
```

2) With loop :

```
MOV R3, #06H ;[ load count value 06H into R3]
MOV R0, #50H ;[ R0 points to 50H]
MOV A , #36H ;[ move 36H into A]
L1: MOV @R0, A ;[ move 36H into memory location from 50H]
    INC R0     ;[ increment R0 by 1]
    DJNZ R3, L1 ;[ decrement R3 by 1] if not equal to 0 jump to L1
    END
```

3)

```
MOV A, #36H ;[ load 36H into A]
MOV 50H, A  ;[ move 36H into 50H location]
MOV 51H, A  ;[ move 36H into 51H location]
MOV 52H, A  ;[ move 36H into 52H ]
MOV 53H, A  ;[ move 36H into 53H]
MOV 54H, A  ;[ Move 36H into 54H]
MOV 55H, A  ;[
```

} using direct addressing mode.

3.3 Programs to perform addition and subtraction

1) Write an assembly language program to compute the sum of two 16 byte numbers. Assume that 1st number is stored in internal data memory location 30H to 3FH and the second number in locations 40H to 4FH. Store the 16 byte sum in locations 40H to 4FH.

```
MOV R3 , #10H
MOV CLR C
MOV R0 , #30H
MOV R1 , #40H
AGAIN : MOV A , @R0
ADDC A , @R1
MOV @R1 , A
INC R0
INC R1
DJNZ R3 , AGAIN
END
```

2) Write a program to subtract 1296H from 2762H and store in result in 50H and 51H.

```
MOV A , #62H
SUBB A , #96H
MOV 50H , A
MOV A , #27H
SUBB A , #12H
MOV 51H , A
END
```

3) Write a program to add two numbers located at external RAM memory locations 4100H and 4101H. Store the result at 4102H and 4103H.

```
MOV DPTR , #4100H
MOVX A , @DPTR
MOV R4 , A
MOV R3 , #00H
INC DPTR
MOVX A , @DPTR
ADD A , R4
INC DPTR
MOVX @DPTR , A
JNC L1
INC R3
L1 : INC DPTR
MOV A , R3
MOVX @DPTR , A
HERE : SJMP HERE
END
```

8.4 Program using Jump instructions

1) Write a program to find smallest number from an array of an array of n numbers. Let the array count is stored at 2400H. Array starts from 2401H. Store the result at 2450H.

Program :-

```
MOV  DPTR , # 2400H
MOVX  A , @DPTR
MOV  R3 , A
INC  DPTR
MOVX A , @DPTR
MOV  B , A
DEC  R3
L1 : INC  DPTR
     MOVX  A , @DPTR
     CJNE  A , B, L2
     SJMP  L3
L2 : JNC  L3
     MOV B, A
L3 : DJNZ  R3 , L1
     MOV  DPTR , # 2450H
     MOV A, B
     MOVX  @DPTR , A
END
```

2) Write a program to find biggest number from an array of n numbers. Let the array count is stored in 2400H, and array starts from 2401H. And store the result in the 2450H.

Program:

```
MOV DPTR , #1 2400H
MOVX  A , @DPTR
MOV  R3 , A
INC  DPTR
MOVX  A , @DPTR
MOV  B , A
DEC  R3
L1 : INC DPTR
     MOVX  A , @DPTR
     CJNE  A , B , L2
     SJMP  L3
L2 : JC  L3
     MOV B, A
L3 : DJNZ  R3 , L1
     MOV DPTR, # 2450H
     MOV A, B
     MOVX @DPTR, A
END
```

3.5 Delay Subroutine to introduce Time delay of Time Period.

⇒ The cpu takes a certain number of clock cycles to execute an instruction.

⇒ In 8051 clock cycles called as machine cycle. One machine cycle for 12 Oscillations periods.

→ To calculate the machine cycle for 8051, we have to consider 1/12 of crystal frequency.

1) Find the size of the delay in the following program if the crystal frequency is 11.0592 MHz

```
    MOV A, #55H
Again: MOV P1, A
       ACALL delay
       CPL A
       SJMP again
Delay: MOV R3, #200
Here: DJNZ R3, Here
       RET
```

Crystal frequency = 11.0592 MHz

Frequency of machine cycle = 11.0592 / 12 = 0.9216 MHz

Period of machine cycle = $\frac{1}{0.9216\ MHz}$ = 1.085 μsec

Program	machine cycle	no. of times it Executes	no. of Machine cycles
Delay: MOV R3, #200	1	1	1
Here: DJNZ R3, Here	2	200	400
RET	2	1	2

Total machine cycles = 1 + 400 + 2 = 403

Time delay = 403 × 1.085 μsec = 436.955 μsec

2) Find the time delay for sub routine

```
Delay: MOV R3, #250
Here: NOP
      NOP
      NOP
      NOP
      DJNZ R3, Here
      NOP.
```

Sol:- Crystal frequency = 11.0592 MHz

Frequency of machine cycle = 11.0592/12 = 0.9216 MHz

Period of machine cycle = 1/0.9216 MHz = 1.085 μsec

Program	machine cycles	No of times it executes	No of machine cycles
Delay: MOV R_3, #250	1	1	1
Here NOP	1	250	250
NOP	1	250	250
NOP	1	250	250
DJNZ R_3, here	2	250	500
RET	2	1	2

Total machine cycles = 1503

Time delay = 1503 x 1.085 μsec

= 1630.755 μsec

3) For an 8051 system of 11.0592 MHz, write a subroutine program to generate the time delay of 5msec

Ans:- Crystal frequency = 11.0592 MHz

Frequency of machine cycle = 0.9216 MHz

period of machine cycle = 1.085 μsec

Time delay = n x Clock period

$5 \times 10^{-3} = n \times 1.085 \times 10^{-6}$

$n = 4608$.

Instruction	machine cycle	Executed No. of times	No. MC
MOV R_3, #09H	1	1	1
Here: MOV R_3, #254	1	9	9
Again: DJNZ R_3, Again	2	254x9	4572
DJNZ R_3, Here	2	9	18
NOP	1	1	1
NOP	1	1	1
NOP	1	1	1
NOP	1	1	1
NOP	1	1	1
NOP	1	1	1
RET	2	1	2

Initial count = (Max. Value + 1) − (Time delay × Timer clock frequency)

$= (65,535+1) - \left(\frac{5m \times 11.0592\ MHz}{12}\right)$

$= 65536 - 460E = 6078 = \underbrace{EE}_{TH_1}\ \underbrace{00}_{TL_1}11$

```
Program: MOV TMOD, #10H ; Timer1, mode1 Initialize
Again:   MOV TL1, #00H  ; Initalize TL1 & TH1
         MOV TH1, #0EEH
         SETB TR1       ; Start timer1
Wait:    JNB TF1, Wait  ; Wait until timer1 counts
         CLR TR1        ; Stop timer
         CLR TF1        ; clear TimeLoverflow
         SJMP Again
```

Write a delay subroutine (RET) to introduce time delay of given time period (in milliseconds) without using 8051 internal timer?

Using loop technique procedure:

① Start ② Load number in RAM Location ③ Decrement RAM location ④ IS RAM = 00? If No Go to 3 ⑤ Stop.

program 1ms

```
MOV R6, #01h
ACall Delay  ; call 1ms delay
CLR P2.0
MOV R6, #01h
ACall Delay

Delay:
LOOP2: MOV R7, #0FAh
LOOP1: NOP      ; 1 cycle
       NOP      ; 1 cycle
       DJNZ R7, LOOP1 ; 2 cycle
       DJNZ R6, LOOP2 ; 4 x 250 = 1000 cycles
       RET
```

Time delay = Total No. of cycles × Time period

$= 1000 \times 1\ \mu s = 1000\ \mu s = 1\ ms$

3.6 Write a program to introduce time delay of seven time periods using 8051 internal timers

Sol

Timer clock frequency = $\frac{\text{Crystal frequency}}{12}$

Timer clock period = $\frac{12}{\text{Crystal frequency}}$

period of machine cycle = $\frac{1}{\text{Timer clock frequency}}$

(1) Time delay = [max. value − Initial count + 1] × Timer clock period

Time delay = [Max. Value − Initial count + 1] × $\frac{12}{\text{Crystal frequency}}$

(2) Initial count = [Max. Value + 1] − (Time delay × Timer clock frequency)

Initial count = [Max. Value + 1] − $\left[\frac{\text{Timer delay} \times \text{crystal frequency}}{12}\right]$

(3) Maximum count value

Mode	Timer size	Maxm count Hex & decimal	
Mode 0	13-bit	1FFF H	8191 D
Mode 1	16-bit	FFFF H	65535
Mode 2	8-bit	FF H	255 D

M_1	M_0	
0	0	← Mode 0
0	1	← Mode 1
1	0	← Mode 2
1	1	← Mode 3

Procedure for time Delay generation:

Step1: Initialize TMOD for Required Timer (0 or 1) and mode (0 or 1 or 2)

Step2: Load the initial count value in Register TL and TH (Timer 0 or 1)

Step3: Start the timer (set TR_1 or TR_0 bit of TCON Register.

Step4: Wait until the timer flag (CTF or TF of TCON) is set

$CTF_1 = 1$ when timer overflows the maximum value)

Step5: stop timer (TR_1 (or) TR_0 mode 2 e.g.)

Step6: clear TF flag

Step7: Repeat from step2 for next delay.

Program: Write a 8051 ALP to Generate a delay of 5ms using timer1 in mode1 with XTAL frequency of 11.0592 MHz

(TF) { Count = 1 ; Timer = 0 }

Sol

Step1: TMOD Register programmed as:

	Timer1				Timer0			
Gate	C/T	M_1	M_0	Gate	C/T	M_1	M_0	
0	0	0	1	0	0	0	0	→ 10H

TMOD = 10H

Timer1 Mode1

Step2: Load TL_1 & TH_1

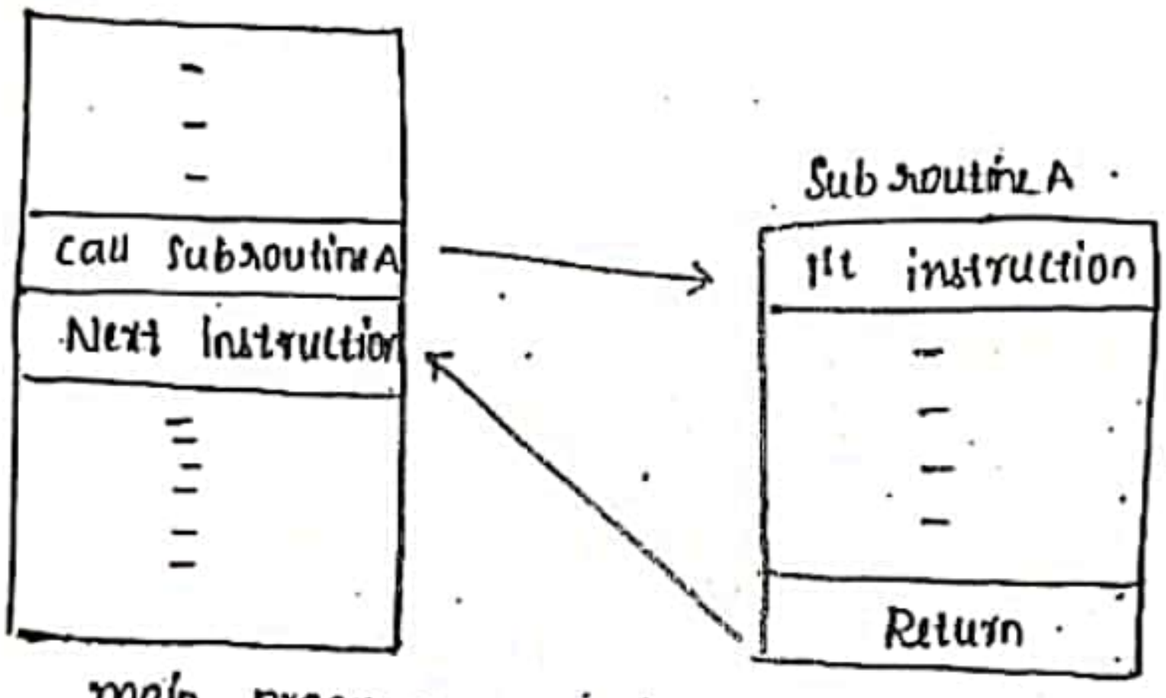
3.7 Subroutine

→ A subroutine is a group of instructions that performs a subtask which occurs repeatedly.

→ Some operations may occurs several times and they are not available as individual instructions. Such a small program is called subroutine

Advantages:-

1) Length of the program reduces
2) Easy to understand
3) Fast in Execution
4) Efficient use of memory
5) It makes the main program appears simple..



3.8 Sequence of program when subroutine is called & Executed

CALL and RET opcodes are used execution of a smaller programs which is termed as subroutine.

⇒ Whenever CALL opcode is used in the main program. The sequence has to follow the following sequence of events takes place automatically by 8051.

1) The address of an instruction, next to CALL instruction is placed into the stack.
2) The status of internal registers is saved on the stack if it is necessary.

3) The subroutine address is specified in the CALL instruction is placed in to program counter.

4) The subroutine is executed.

5) At the end of the subroutine, RET opcode appears in the subroutine. The SOFT retrieves the address of data stored on the stack.

6) The execution of a main program begins again.

3.9 Information Exchange b/w PC and stack in subroutine

⇒ ACALL instruction appears in main program.

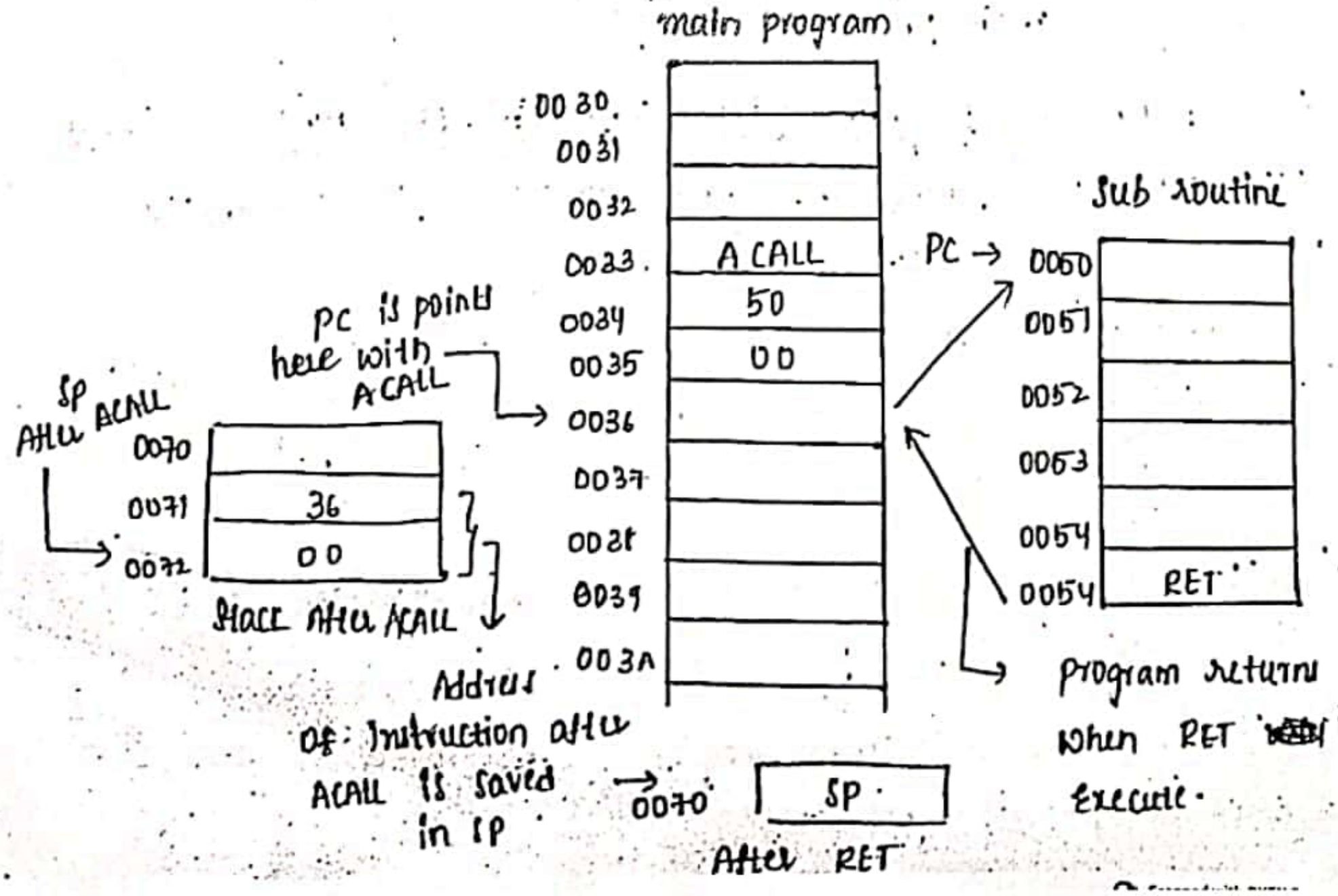
⇒ The return address 0036H of the next instruction after the call instruction is found in PC.

⇒ The return address bytes 0036H are pushed on stack. location 0071H and 0072H. [One is lower and another is higher byte]. The SP is incremented for each push operation on to stack.

⇒ The subroutine address 0050H is placed in PC

⇒ The subroutine is executed.

⇒ RET opcode is encountered at the end of subroutine



→ After execution of RET instruction, the content of stack location 0071H and 0070H are copied into PC.

→ Two 8bit pop operations restore the return address to PC from stack. The stack pointer is decremented for each address byte pop.

→ The main program executed from same location where it was terminated after subroutine's call.

3.10 PUSH, POP Instructions

Pushing onto stack: In 8051 SP points to last used location of stack. So, the pushed to stack, then the SP is incremented by 1. The push operation is executed the contents of the register are also saved on stack and SP is incremented by 1.

Example:-
```
MOV R6, #25H
MOV R1, #12H
MOV R4, #87H
PUSH R6
PUSH R1
PUSH R4
```

Memory address		PUSH R6	PUSH R1	PUSH R4
0B				
0A				87
09			12	12
08		25	25	25
	SP 07	SP 08	SP 09	SP 0A

Popping from stack:

Popping the contents from stack to a specific register is the opposite process of pushing. For every pop, the top byte of stack is copied to register, and the SP is decremented by one.

Example:

```
POP R3
POP R6
POP R2
```

memory address		POP R3	POP R6	POP R2
0B	64			
0A	F9	F9		
09	7C	7C	76	
08	6C	6C	6C	6C
	SP-0B	SP-0A	SP-09	SP-0 08

3.11 Debugging a program

→ Debugging is the procss of finding and eliminating errors in a program.

→ Debugging procss is divided into 2 parts they are static debugging and dynamic debugging.

⇒ Static debugging is a visual inspection by paper and pencil check of a flowchart and machine code.

⇒ Dynamic debugging observes the op of register contents, following the execution of each instructions or a group.

Common source of Errors:

1) Selecting a wrong code. 2) Forgetting the second and third byte of instruction 3) Specifying the wrong jump location.

3.12. Debugging techniques:

1) Single step debugging: A keyboard allows to execute one instruction at a time and to observe results for each one. It allows to check 1) Incorrect address 2) Incorrect jump location for loops 3) Incorrect data or missing code. By using this technique, we have to reduce loop and delay count. If it is large we can't observe flag status properly and it is very useful for small programs.

2) Break point debugging:

It allows to execute a program in sections. It can be set in your computer by using RST instruction in Assembly language program. when you push execute key, your program will be execute until the breakpoint. A 2nd break point. can be set at a subsequent memory address to debug the next segment of program. We can isolate the segment of program with error. Then the segment can be debugged with single step facility. This technique is used to check out timing loop, I/O section & interrupts.