Unit 3:
Stack, I/O Port Interfacing and Programming

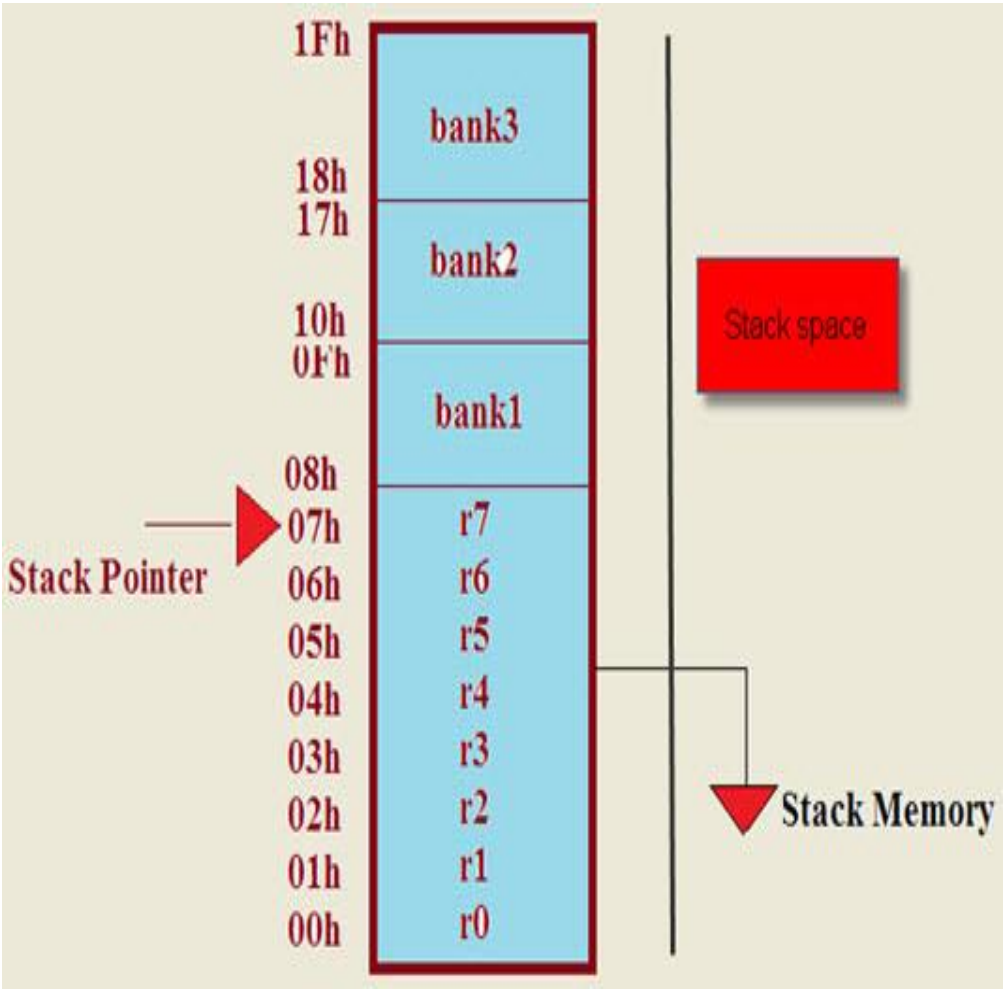
Stack and Subroutine instructions

- The stack is an area of random access memory (RAM) allocated to hold temporarily all the parameters of the variables.
- The stack is also responsible for reminding the order in which a function is called so that it can be returned correctly.
- Whenever the function is called, the parameters and local variables associated with it are added to the stack (PUSH).
- When the function returns, the parameters and the variables are removed (“POP”) from the stack.
- This is why a program’s stack size changes continuously while the program is running.

- The register used to access the stack is called stack pointer register.
- The stack pointer is a small register used to point at the stack.
- When we push something into the stack memory, the stack pointer increases.

Example

- When an 8051 microcontroller power up, the stack pointer contained value is 07, by default, as shown in the above figure.
- If we perform 'PUSH' operation, then the stack pointer address will be increased and shifted to another register.
- To avoid this problem, before starting the program, we have to assign a different address location to the stack pointer.



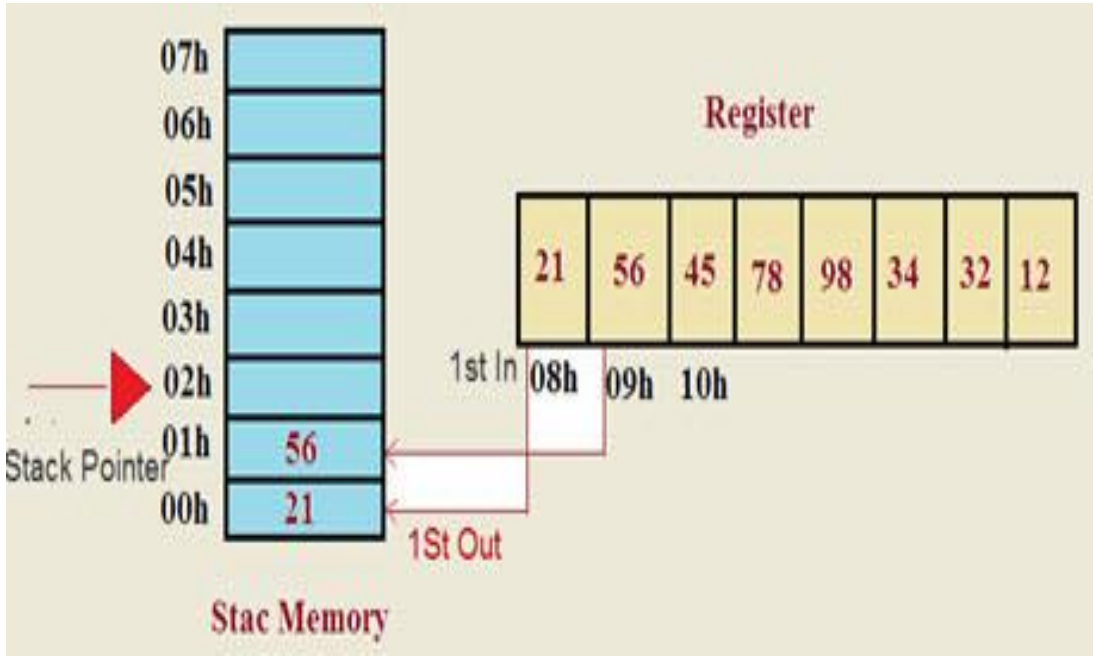
PUSH operation

The 'PUSH' is used for taking the values from any register and storing in the starting address of the stack pointer, i.e., 00h by using 'PUSH' operation. And, for the next 'PUSH', it increments +1, and stores the value in the next address of the stack pointer, i.e., 01h.

PUSH operation means (First in First out)

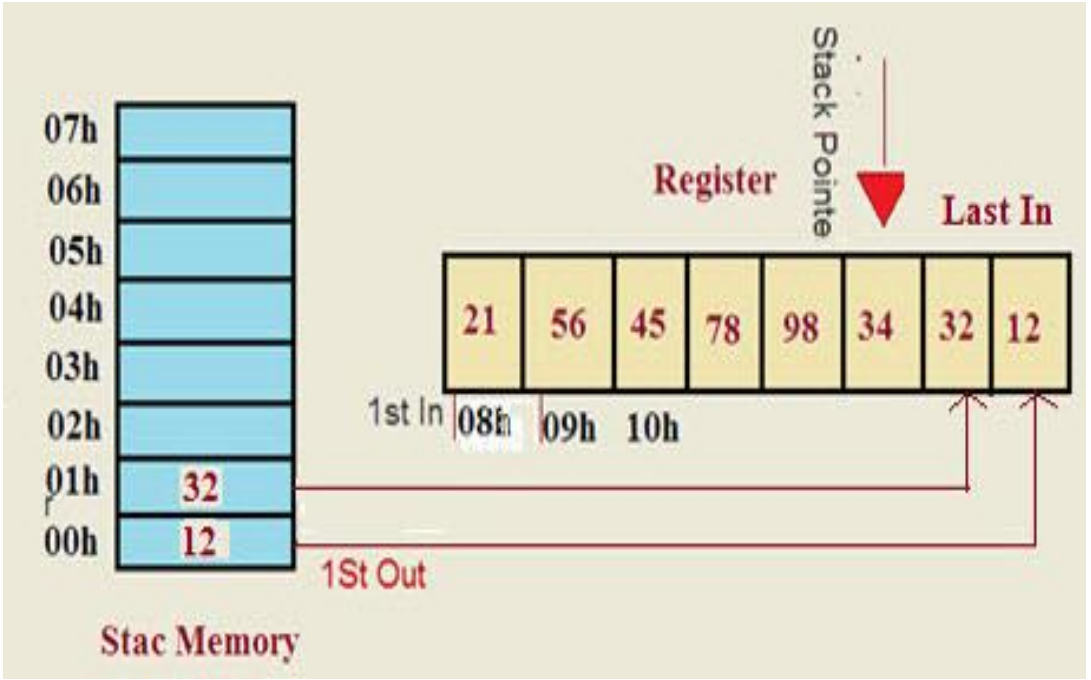
Example: WAP in assembly language for PUSH operation

Org 0000h
MOV 08h, #21h
MOV 09h, #56h
PUSH 00h
PUSH 01h
END



POP Operation

It is used for placing the values from the stack pointer's maximum address to any other register's address. If we use this 'POP' again, then it decrements by 1, and the value stored in any register is given as 'POP'.



POP operation means 'Last in First out'.

```
 org 000H
MOV 00H, #12H
MOV 01H, #32H
POP 1FH
POP 0EH
END
```

Subroutines instructions

- Subroutines are often used to perform tasks that need to be performed frequently.
- This makes a program more structured in addition to saving memory space.
- In the 8051 there are two instructions for call:

❖ **LCALL (long call)**
❖ **ACALL (absolute call).**

- Deciding which one to use depends on the target address.

LCALL (long call)

- In this 3-byte instruction, the first byte is the opcode and the second and third bytes are used for the address of the target subroutine.
- Therefore, LCALL can be used to call subroutines located anywhere within the 64K-byte address space of the 8051.
- To make sure that after execution of the called subroutine the 8051 knows where to come back to, the processor automatically saves on the stack the address of the instruction immediately below the LCALL.
- When a subroutine is called, control is transferred to that subroutine, and the processor saves the PC (program counter) on the stack and begins to fetch instructions from the new location.
- After finishing execution of the subroutine, the instruction RET (return) transfers control back to the caller.
- Every subroutine needs RET as the last instruction. See Example 3-8.

The following points should be noted for the program in Example 3-8.

1.Notice the DELAY subroutine. Upon executing the first “LCALL DELAY”, the address of the instruction right below it, “MOV A, #OAAH”, is pushed onto the stack, and the 8051 starts to execute instructions at address 300H.

2.In the DELAY subroutine, first the counter R5 is set to 255 (R5 = FFH); therefore, the loop is repeated 256 times. When R5 becomes 0, control falls to the RET instruction, which pops the address from the stack into the program counter and resumes executing the instructions after the CALL.

Example 3-8

Write a program to toggle all the bits of port 1 by sending to it the values 55H and AAH continuously. Put a time delay in between each issuing of data to port 1. This program will be used to test the ports of the 8051 in the next chapter.

Solution:

```
        ORG     0
BACK:   MOV     A,#55H      ;load A with 55H
        MOV     P1,A        ;send 55H to port 1
        LCALL   DELAY       ;time delay
        MOV     A,#0AAH     ;load A with AA (in hex)
        MOV     P1,A        ;send AAH to port 1
        LCALL   DELAY
        SJMP    BACK        ;keep doing this indefinitely
;-------this is the delay subroutine
        ORG     300H        ;put time delay at address 300H
DELAY:  MOV     R5,#0FFH    ;R5 = 255(FF in hex),the counter
AGAIN:  DJNZ    R5,AGAIN    ;stay here until R5 becomes 0
        RET                 ;return to caller (when R5 = 0)
        END                 ;end of asm file
```

ACALL (absolute call)

- ACALL is a 2-byte instruction in contrast to LCALL, which is 3 bytes.
- Since ACALL is a 2-byte instruction, the target address of the subroutine must be within 2K bytes because only 11 bits of the 2 bytes are used for the address.
- There is no difference between ACALL and LCALL in terms of saving the program counter on the stack or the function of the RET instruction.
- The only difference is that the target address for LCALL can be anywhere within the 64K-byte address space of the 8051 while the target address of ACALL must be within a 2K-byte range.
- In many variations of the 8051 marketed by different companies, on-chip ROM is as low as IK byte. I
- n such cases, the use of ACALL instead of LCALL can save a number of bytes of program ROM space.

Rewrite Example 3-8 as efficiently as you can.

Solution:

```
            ORG    0
            MOV    A,#55H     ;load A with 55H
BACK:       MOV    P1,A       ;issue value in reg A to port 1
            ACALL  DELAY      ;time delay
            CPL    A          ;complement reg A
            SJMP   BACK       ;keep doing this indefinitely

;------------this is the delay subroutine
DELAY:
            MOV    R5,#0FFH   ;R5=255(FF in hex), the counter
AGAIN:      DJNZ   R5,AGAIN   ;stay here until R5 becomes 0
            RET               ;return to caller
            END               ;end of asm file
```

Notice in this program that register A is set to 55H. By complementing 55H, we have AAH; and by complementing AAH we have 55H. Why? "01010101" in binary (55H) becomes "10101010" in binary (AAH) when it is complemented; and "10101010" becomes "01010101" if it is complemented.

Looping :

Repeating a sequence of instructions until a certain condition is satisfied is called loop. In 8051, the loop action is performed by **DJNZ reg, label.**

Example 3-1

Write a program to
(a) clear ACC, then
(b) add 3 to the accumulator ten times.

Solution:

```
;This program adds value 3 to the ACC ten times

        MOV   A,#0        ;A=0, clear ACC
        MOV   R2,#10      ;load counter R2=10
AGAIN:  ADD   A,#03       ;add 03 to ACC
        DJNZ  R2,AGAIN    ;repeat until R2=0(10 times)
        MOV   R5,A        ;save A in R5
```

Example 3-3

Write a program to (a) load the accumulator with the value 55H, and (b) complement the ACC 700 times.

Solution:

Since 700 is larger than 255 (the maximum capacity of any register), we use two registers to hold the count. The following code shows how to use R2 and R3 for the count.

```
                ORG 0
        MOV     A,#55H          ;A=55H
        MOV     R3,#10          ;R3=10, the outer loop count
NEXT:   MOV     R2,#70          ;R2=70, the inner loop count
AGAIN:  CPL     A               ;complement A register
        DJNZ    R2,AGAIN        ;repeat it 70 times (inner loop)
        DJNZ    R3,NEXT
                END
```

In this program, R2 is used to keep the inner loop count. In the instruction "DJNZ R2,AGAIN", whenever R2 becomes 0 it falls through and "DJNZ R3,NEXT" is executed. This instruction forces the CPU to load R2 with the count 70 and the inner loop starts again. This process will continue until R3 becomes zero and the outer loop is finished.

1. Write a program to add the values of locations 50H and 51H and store the result in locations in 52h and 53H.

```
ORG 0000H ; Set program counter 0000H
MOV A,50H ; Load the contents of Memory location 50H into A
ADD A,51H ; Add the contents of memory 51H with CONTENTS A
MOV 52H,A ; Save the LS byte of the result in 52H
MOV A, #00 ; Load 00H into A
ADDC A, #00 ; Add the immediate data and carry to A
MOV 53H,A ; Save the MS byte of the result in location 53h
END
```

2. Write a program to store data FFH into RAM memory locations 50H to 58H using direct addressing mode

```
ORG 0000H ; Set program counter 0000H
MOV A, #0FFH ; Load FFH into A
MOV 50H, A ; Store contents of A in location 50H
MOV 51H, A ; Store contents of A in location 5IH
MOV 52H, A ; Store contents of A in location 52H
MOV 53H, A ; Store contents of A in location 53H
MOV 54H, A ; Store contents of A in location 54H
MOV 55H, A ; Store contents of A in location 55H
MOV 56H, A ; Store contents of A in location 56H
MOV 57H, A ; Store contents of A in location 57H
MOV 58H, A ; Store contents of A in location 58H
END
```

3. Write a program to subtract a 16 bit number stored at locations 51H-52H from 55H-56H and store the result in locations 40H and 41H. Assume that the least significant byte of data or the result is stored in low address. If the result is positive, then store 00H, else store 01H in 42H.

```
ORG 0000H ; Set program counter 0000H
MOV A, 55H ; Load the contents of memory location 55 into A
CLR C ; Clear the borrow flag
SUBB A,51H ; Sub the contents of memory 51H from contents of A
MOV 40H, A ; Save the LSByte of the result in location 40H
MOV A, 56H ; Load the contents of memory location 56H into A
SUBB A, 52H ; Subtract the content of memory 52H from the content A
MOV 41H,A ; Save the MSbyte of the result in location 41h.
MOV A, #00 ; Load 00H into A
ADDC A, #00 ; Add the immediate data and the carry flag to A
MOV 42H, A ; If result is positive, store00H, else store 0lH in 42H
END
```

Example: Perform (FEDC-1234) using 8051 assembly language.

```
ORG 0000h
MOV R0, #ODCh
MOV R1, #OFEh
MOV R2,#34h
MOV R3,#12h
CLR C
MOV A,R0
SUBB A,R2
MOV 22h, A
MOV A,R1
SUBB A,R3
MOV 21h,A
MOV 00h,C
END
```

4. Write a program to add two 16 bit numbers stored at locations 51H-52H and 55H-56H and store the result in locations 40H, 41H and 42H. Assume that the least significant byte of data and the result is stored in low address and the most significant byte of data or the result is stored in high address.

```
ORG 0000H ; Set program counter 0000H
MOV A,51H ; Load the contents of memory location 51H into A
ADD A,55H ; Add the contents of 55H with contents of A
MOV 40H,A ; Save the LS byte of the result in location 40H
MOV A,52H ; Load the contents of 52H into A
ADDC A,56H ; Add the contents of 56H and CY flag with A
MOV 41H,A ; Save the second byte of the result in 41H
MOV A,#00 ; Load 00H into A
ADDC A,#00 ; Add the immediate data 00H and CY to A
MOV 42H,A ; Save the MS byte of the result in location 42H
END
```

5. Write a program to compute 1 + 2 + 3 + N (say N=15) and save the sum at70H

```
        ORG 0000H ; Set program counter 0000H
        N EQU 15 ; EQU instruction assigns absolute or relocatable values to symbols
        MOV R0,#00 ; Clear R0
        CLR A ; Clear A
again: INC R0 ; Increment R0
        ADD A, R0 ; Add the contents of R0 with A
        CJNE R0,#N,again ; Loop until counter, R0, N
                                ; CJNE instruction compares the first two operands
and branches to the specified destination if their values are not equal.
        MOV 70H,A ; Save the result in location 70H
        END
```

6. Write a program to multiply two 8 bit numbers stored at locations 70H and 71H and store the result at memory locations 52H and 53H. Assume that the least significant byte of the result is stored in low address.

```
ORG 0000H ; Set program counter 00 OH
MOV A, 70H ; Load the contents of memory location 70h into A
MOV B, 71H ; Load the contents of memory location 71H into B
MUL AB ; Perform multiplication
MOV 52H,A ; Save the least significant byte of the result in location 52H
MOV 53H,B ; Save the most significant byte of the result in location 53
END
```

7. Write a program to find the cube of an 8 bit number program is as follows

```
ORG 0000H
MOV R1,#N
MOV A,R1
MOV B,R1
MUL AB //SQUARE IS COMPUTED
MOV R2, B
MOV B, R1
MUL AB
MOV 50,A
MOV 51,B
MOV A,R2
MOV B, R1
MUL AB
ADD A, 51H
MOV 51H, A
MOV 52H, B
MOV A, # 00H
ADDC A, 52H
MOV 52H, A //CUBE IS STORED IN 52H,51H,50H
END
```

8. Write a program to count the number of and o's of 8 bit data stored in location 6000H.

```
ORG 0000h ; Set program counter 0000h
        MOV DPTR, #6000h ; Copy address 6000H to DPTR
        MOVX A, @DPTR ; Copy number to A
        MOV R0,#08 ; Copy 08 in RO
        MOV R2,#00 ; Copy 00 in R2
        MOV R3,#00 ; Copy 00 in R3
        CLR C ; Clear carry flag
BACK: RLC A ; Rotate A through carry flag
        JC NEXT ; If CF = 1, branch to next
        INC R2 ; If CF = 0, increment R2 AJMP NEXT2
        NEXT: INC R3 ; If CF = 1, increment R3
        NEXT2: DJNZ R0,BACK ; Repeat until RO is zero
        END
```

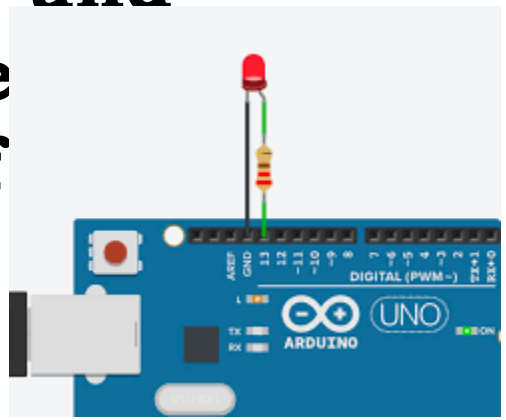
9. Write a program to shift a 24 bit number stored at 57H-55H to the left logically four places. Assume that the least significant byte of data is stored in lower address.

```
        ORG 0000H ; Set program counter 0000h
        MOV R1,#04 ; Set up loop count to 4
again: MOV A,55H ; Place the least significant byte of data in A
        CLR C ; Clear the carry flag
        RLC A ; Rotate contents of A (55h) left through carry
        MOV 55H,A
        MOV A,56H
        RLC A ; Rotate contents of A (56H) left through carry
        MOV 56H,A
        MOV A,57H
        RLC A ; Rotate contents of A (57H) left through carry
        MOV 57H,A
        DJNZ R1,again ; Repeat until R1 is zero
        END
```

Interfacing simple switch and LED to I/O ports to switch on/off LED with respect to switch status

Toggle switch: a toggle switch is a device that opens and closes an electric circuit by moving a lever back and forth. It is used to switch between two conditions in a circuit, namely ON and OFF.

LED : is a semiconductor PN-junction diode which emits light when an electric current passes through it. It has two terminals Anode and Cathode. LED emits light when anode connected to +ve and cathode to ground of battery.



Interrupts: Interrupts is a sub-routine call that given by the microcontroller when some other program with high priority is request for acquiring the system buses the ‘n’ interrupts occur in current running program.

Types of interrupt in 8051 Microcontroller:

Let's see the five sources of interrupts in 8051 Microcontroller:

- Timer 0 overflow interrupt - TF0
- Timer 1 overflow interrupt - TF1
- External hardware interrupt - INT0
- External hardware interrupt - INT1
- Serial communication interrupt - RI/TI

Interrupt Enable (IE) Register Contd..

Name	Pin	Description of Pin
EA	IE.7	EA=0, Disables all the interrupts and EA=1 enables or diables particular interrupt based on individual enable bit.
-	IE.6	Reserved for future use
-	IE.5	Reserved for future use.
ES	IE.4	Enables/disables serial port interrupt.
ET1	IE.3	Enables/disables timer1 overflow interrupt.
EX1	IE.2	Enables/disables external interrupt1.
ET0	IE.2	Enables/disables timer0 overflow interrupt.
EX0	IE.0	Enables/disables external interrupt0

Timer Control (TCON) Register

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

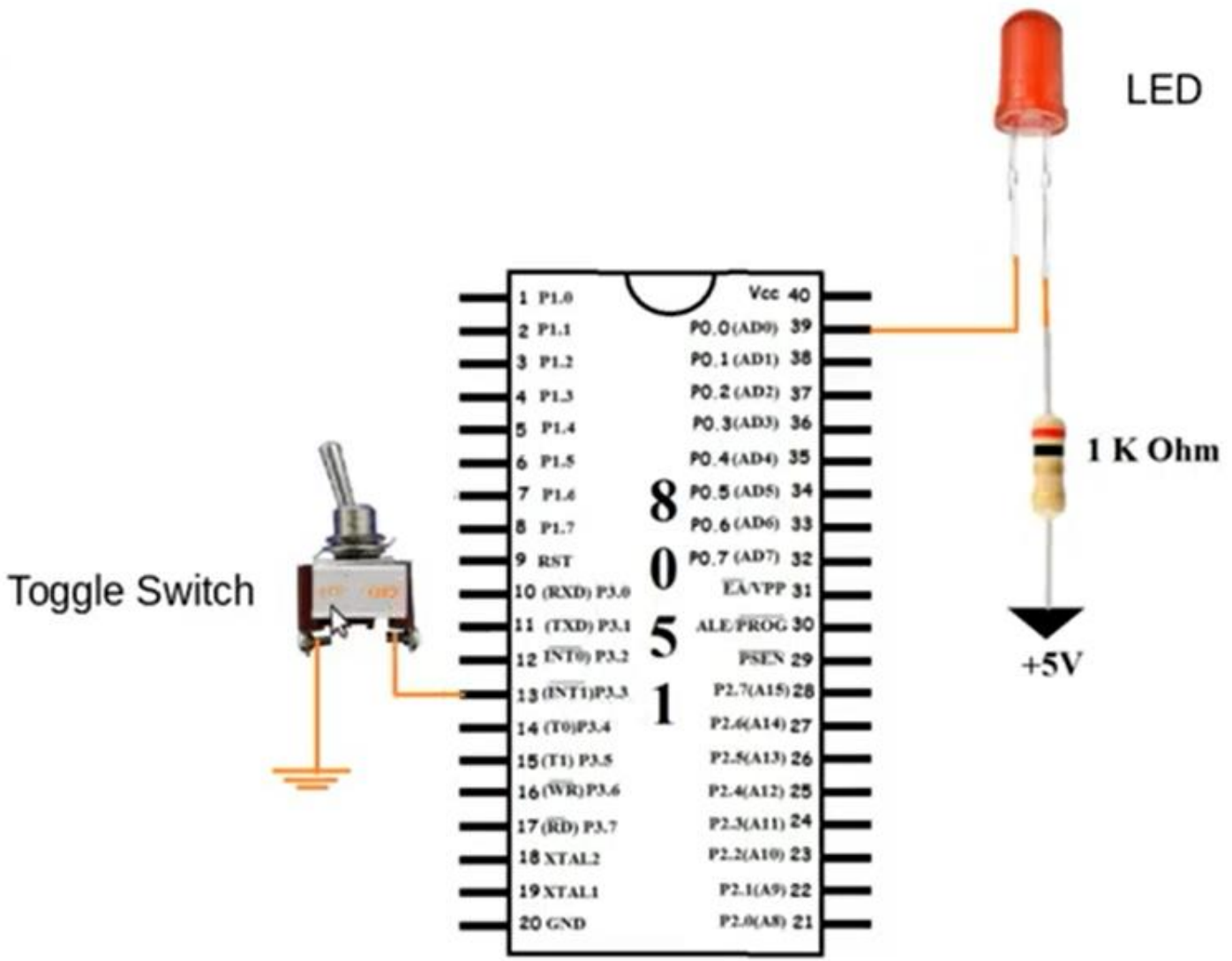
- TF1/TF0 - Timer 1/Timer 0 overflow flags; set to 1 by hardware whenever timer overflows
- TR1/TR0 - Timer 1/Timer 0 run control bit; used to start and stop timers
- IE1/IE0 - External interrupt 1/interrupt 0 Edge flag; set by hardware whenever there is external interrupt. Cleared automatically after servicing the interrupt.
- IT1/IT0 - External interrupt 1/interrupt 0 signal type control bit; when set to 1- Interrupt is Falling edge triggered and when set to 0, low level triggered.

Interrupt Vector Table (IVT)

- Every interrupt causes the program to make a hardware call to one of the dedicated addresses in program memory.
- The programmer has to place the subroutine at the addresses that will service the interrupt.

Name of the Interrupt	Address (in HEX)
RESET	0000
External Interrupt 0 (INT0)	0003
Timer 0 Overflow Interrupt (TF0)	000B
External Interrupt 1 (INT1)	0013
Timer 1 Overflow Interrupt (TF1)	001B
Serial Interrupt (RI/TI)	0023

Interfacing Circuit



B. K. Gudur

Algorithms (Logic of first part)

Step 1: Initialize one port pin as input and other as output.

Step 2:initialize interrupt INT1 for edge trigger.

Step 3: Wait for an interrupt.

Step 4: Upon receiving the interrupt, execute ISR to turn ON the LED and check status of the pin to which switch has been connected.

Step 5: If pin is at logic 0 (switch ON) go to step 5

else

switch OFF go to step 6.

Step 6: Turn OFF the LED and return to step 3.

Algorithms (Logic of first part)

Step 1: Initialize one port pin as input and other as output.

Step 2:initialize interrupt INT0 for edge trigger.

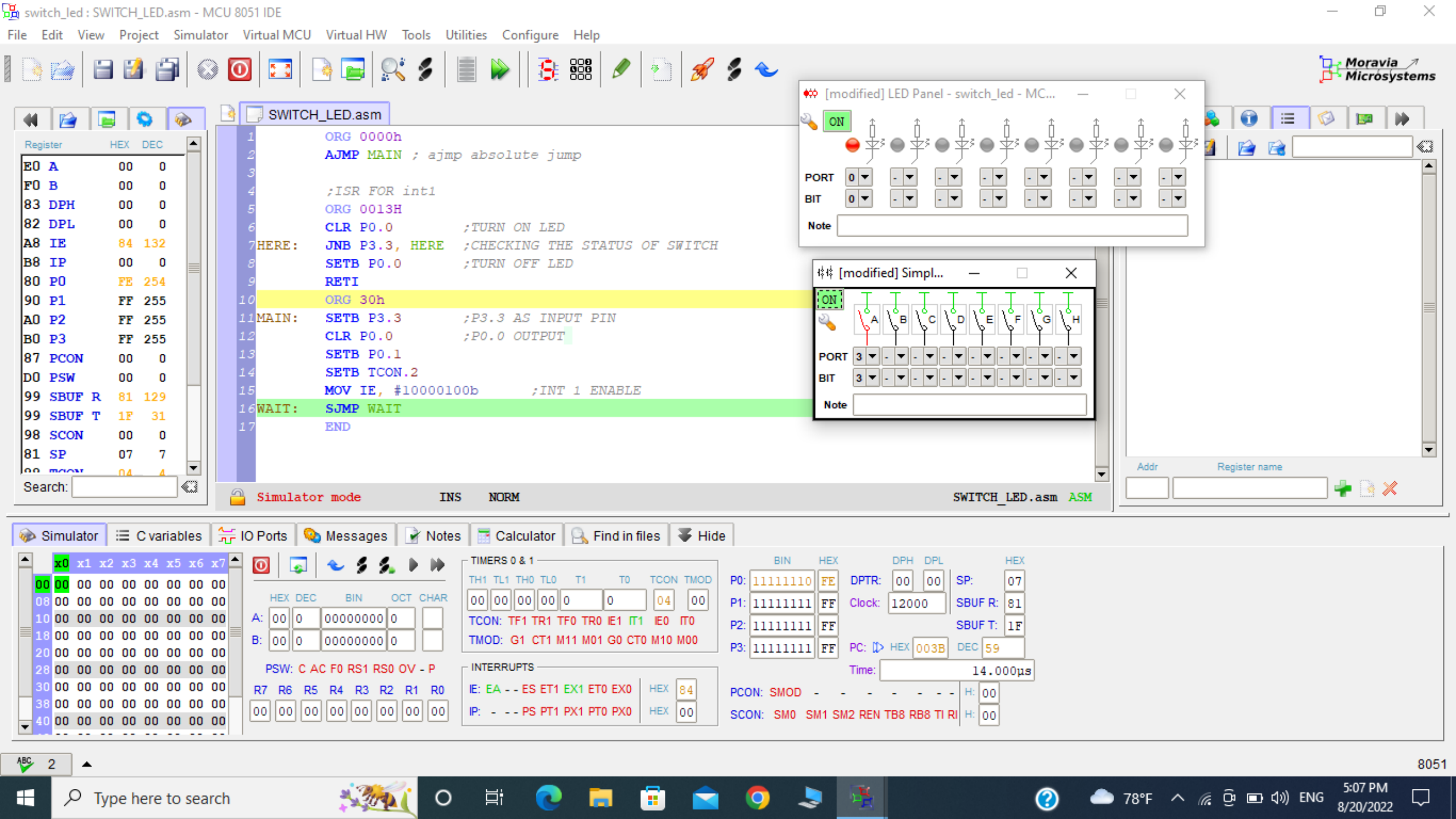
Step 3: Wait for an interrupt.

Step 4: Upon receiving the interrupt, turn ON the LED and call delay subroutine.

Step 5: After returning from delay subroutine turn OFF the LED and go to step 3.

```
        ORG 0000h
        AJMP MAIN ; ajmp absolute jump

        ;ISR FOR int1
        ORG 0013H
        CLR P0.0  ;TURN ON LED
HERE:   JNB P3.3, HERE      ;CHECKING THE STATUS OF SWITCH
        SETB P0.0;TURN OFF LED
        RETI
        ORG 30h
MAIN:   SETB P3.3;P3.3 AS INPUT PIN
        CLR P0.0  ;P0.0 OUTPUT
        SETB P0.1
        SETB TCON.2
        MOV IE, #10000100b          ;INT 1 ENABLE
WAIT:   SJMP WAIT
        END
```


switch_led : SWITCH_LED.asm - MCU 8051 IDE
File Edit View Project Simulator Virtual MCU Virtual HW Tools Utilities Configure Help
Moravia Microsystems
SWITCH_LED.asm
1 ORG 0000h
2 AJMP MAIN ; ajmp absolute jump
3
4 ;ISR FOR int1
5 ORG 0013H
6 CLR P0.0 ;TURN ON LED
7 HERE: JNB P3.3, HERE ;CHECKING THE STATUS OF SWITCH
8 SETB P0.0 ;TURN OFF LED
9 RETI
10 ORG 30h
11 MAIN: SETB P3.3 ;P3.3 AS INPUT PIN
12 CLR P0.0 ;P0.0 OUTPUT
13 SETB P0.1
14 SETB TCON.2
15 MOV IE, #10000100b ;INT 1 ENABLE
16 WAIT: SJMP WAIT
17 END
Register HEX DEC
E0 A 00 0
F0 B 00 0
83 DPH 00 0
82 DPL 00 0
A8 IE 84 132
B8 IP 00 0
80 P0 FE 254
90 P1 FF 255
A0 P2 FF 255
B0 P3 FF 255
87 PCON 00 0
D0 PSW 00 0
99 SBUF R 81 129
99 SBUF T 1F 31
98 SCON 00 0
81 SP 07 7
Search:
[modified] LED Panel - switch_led - MC...
ON
PORT 0
BIT 0
Note
[modified] Simpl...
ON
A B C D E F G H
PORT 3
BIT 3
Note
Addr Register name
Simulator mode INS NORM SWITCH_LED.asm ASM
Simulator C variables IO Ports Messages Notes Calculator Find in files Hide
x0 x1 x2 x3 x4 x5 x6 x7
HEX DEC BIN OCT CHAR
A: 00 0 00000000 0
B: 00 0 00000000 0
PSW: C AC F0 RS1 RS0 OV - P
R7 R6 R5 R4 R3 R2 R1 R0
TIMERS 0 & 1
TH1 TL1 TH0 TL0 T1 T0 TCON TMOD
TCON: TF1 TR1 TF0 TR0 IE1 IT1 IE0 IT0
TMOD: G1 CT1 M11 M01 G0 CT0 M10 M00
INTERRUPTS
IE: EA - - ES ET1 EX1 ET0 EX0
IP: - - - PS PT1 PX1 PT0 PX0
BIN HEX DPH DPL HEX
P0: 11111110 FE DPTR: 00 00 SP: 07
P1: 11111111 FF Clock: 12000 SBUF R: 81
P2: 11111111 FF SBUF T: 1F
P3: 11111111 FF PC: HEX 003B DEC 59
Time: 14.000µs
PCON: SMOD - - - - - - - H: 00
SCON: SM0 SM1 SM2 REN TB8 RB8 TI RI H: 00
8051
Type here to search
78°F ENG 5:07 PM 8/20/2022