FILE ON EMBEDDED SYSTEMS LAB

ESCC303

ON KEIL AND MULTISIM

DEPARTMENT OF ELECTRONIC SCIENCE

UNIVERSITY OF DELHI

By,

T Raja Aadhithan

19/1031

TABLE OF CONTENTS

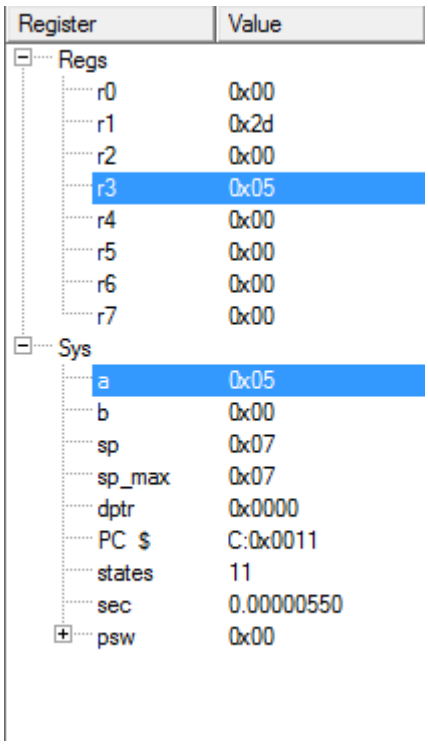
EXPERIMENT 1.1

Addition and subtraction of two eight bit hexadecimal numbers by immediate addressing mode and store the result in general purpose registers.

Code:

```
1           ORG 00H
2           MOV A,#10H ; load 10H to A
3           ADD A,#1DH ; add 1DH to A
4           MOV R1,A   ; store result in R1
5           JNC L1
6           INC R0     ; store carry in R0
7       L1: MOV A,#0AH ; load 0AH in A
8           SUBB A,#05H; Sub 05H from A
9           MOV R3,A   ; store result in R3
10          JNC L2
11          INC R2     ; store carry in R2
12      L2: NOP
13          END
```

Output:



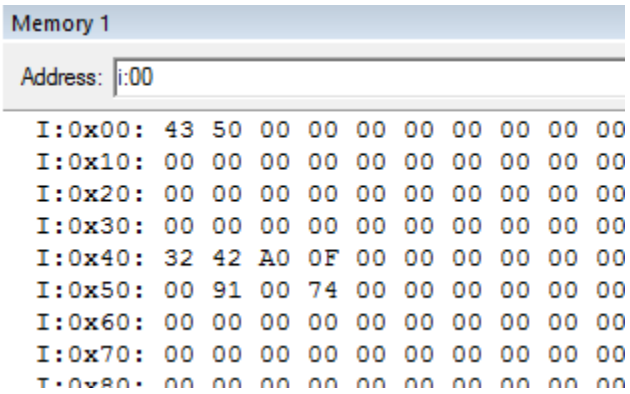
EXPERIMENT 1.2

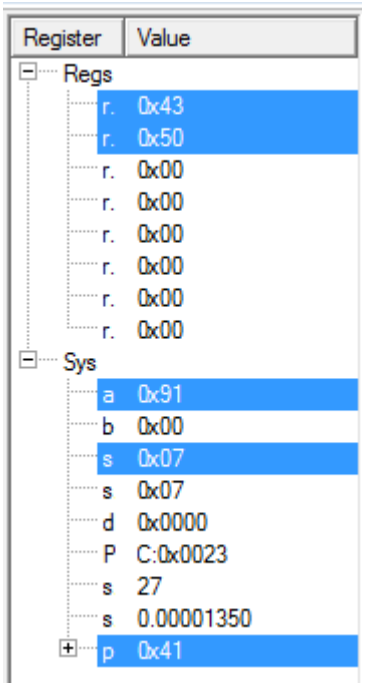
Addition and subtraction of two eight bit hexadecimal numbers by register indirect addressing mode and store the result in scratch pad memory location.

Code:

```
        ORG 00H
        MOV 40H, #32H
        MOV 41H, #42H
        MOV 42H, #0A0H
        MOV 43H, #0FH
        MOV R0, #40H ; pointer for inputs
        MOV R1, #53H ; pointer for outputs
        MOV A ,@R0   ; move 1st value to A
        INC R0       ; inc A
        ADD A ,@R0   ; add 2nd value to A
        INC R0       ; inc A
        MOV @R1,A    ; store result
        DEC R1       ; dec R1
        JNC L1       ; check for carry
        INC @R1      ; store carry
    L1: DEC R1       ; dec R1
        MOV A ,@R0   ; move 3rd value to A
        INC R0       ; inc R0
        SUBB A,@R0   ; sub 4th value from A
        MOV @R1,A    ; store result i
        DEC R1       ; dec R1
        JNC L2       ; check for carry
        INC @R1      ; store carry
    L2: NOP
        END
```

Output:





EXPERIMENT 1.3

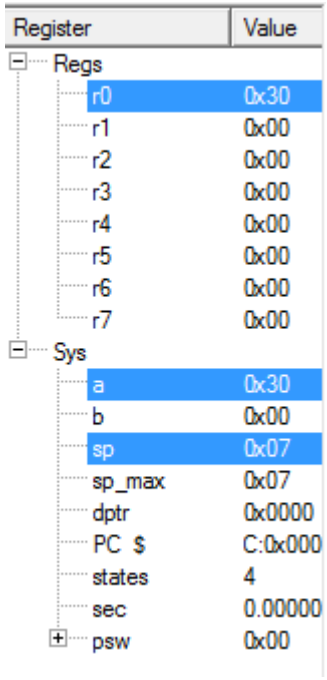
Addition and subtraction of two eight bit BCD numbers.

Code:

```
1         ORG 0000H
2         MOV A,#10H ; 1ST BCD NO
3         ADD A,#20H ; 2ND BCD NO
4         DA A       ; CONVERT TO BCD
5         MOV R0,A   ; STORE IN R0
6         END
```

Output:

stored in R0



EXPERIMENT 1.4

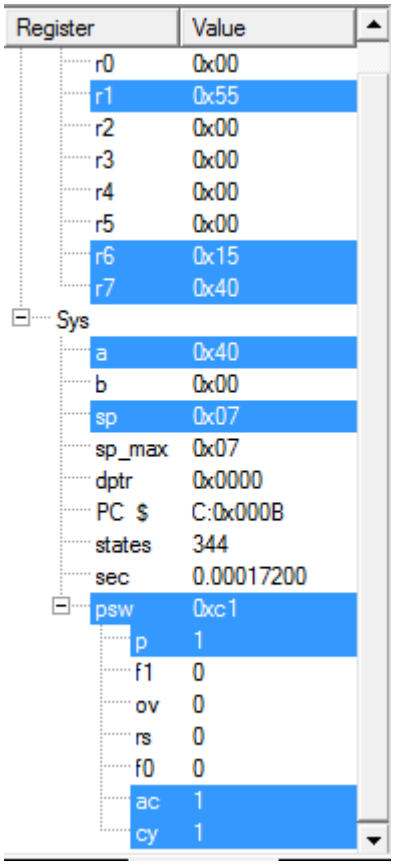
Multiplication of two eight-bit hexadecimal numbers by successive addition.

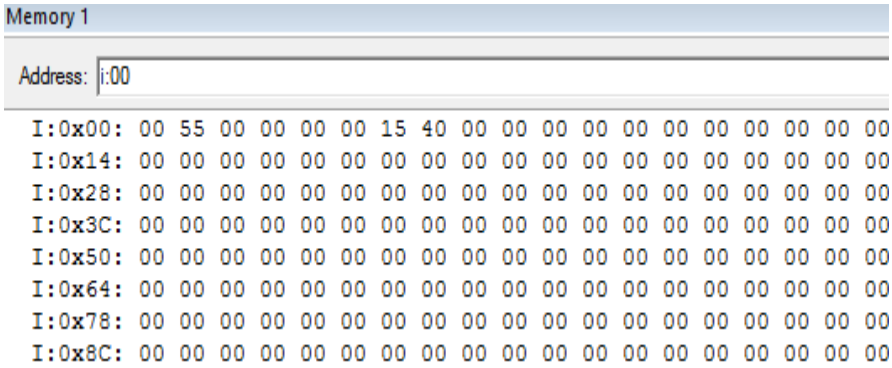
Code:

multi.asm

```
1 ;multipling 55h and 40h
2         MOV R1, #55H ; 1st number
3         MOV R0, #40H ; 2nd number
4 NEXT:   ADD A, R1    ; adding R1 repeatedly
5         JNC LABEL
6         INC R6       ; increment R2 on carry
7 LABEL:  DJNZ R0 , NEXT ; add R1 to acc
8         MOV R7, A
```

Output:





THE OUTPUT IS FOUND AT R7,R6

EXPERIMENT 1.5

Multiplication of two eight-bit hexadecimal numbers using MUL command and division of two eight-bit hexadecimal numbers

Code:

```
1         ORG 00H
2         MOV 40H, #32H
3         MOV 41H, #42H
4         MOV 60H, #0A0H
5         MOV 61H, #0FH
6         MOV R0, #40H ; pointer for inputs
7         MOV R1, #44H ; pointer for outputs
8         MOV A ,@R0   ; move 1st value to A
9         INC R0       ; inc R0
10        MOV B  ,@R0  ; mov 2nd value to b
11        MUL AB       ; MULTIPLY
12        MOV @R1,A    ; store result
13        DEC R1       ; dec R1
14        MOV @R1,B    ; store result
15        MOV R0,#60H
16        MOV R1,#64H
17        MOV A ,@R0   ; move 3rd value to A
18        INC R0       ; inc R0
19        MOV B,@R0    ; move 4th value to B
20        DIV AB       ; DIVIDE
21        MOV @R1,A    ; store reminder
22        DEC R1       ; dec R1
23        MOV @R1,B    ; store quotient
24        END
```

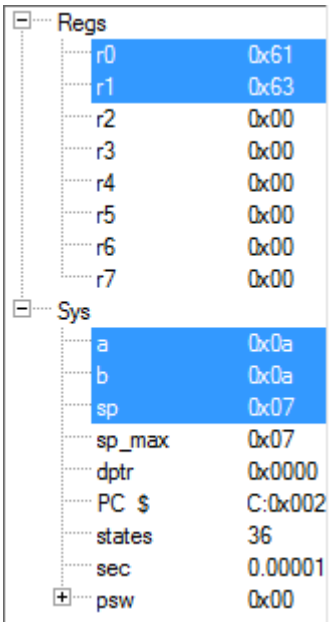
MULTIPLICATION:

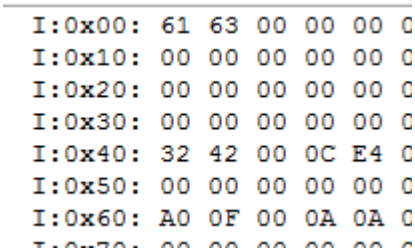
inputs: 40H , 41H
outputs: 43H , 44H

DIVISION:

inputs: 40H , 41H
outputs: 43H , 44H

Output:





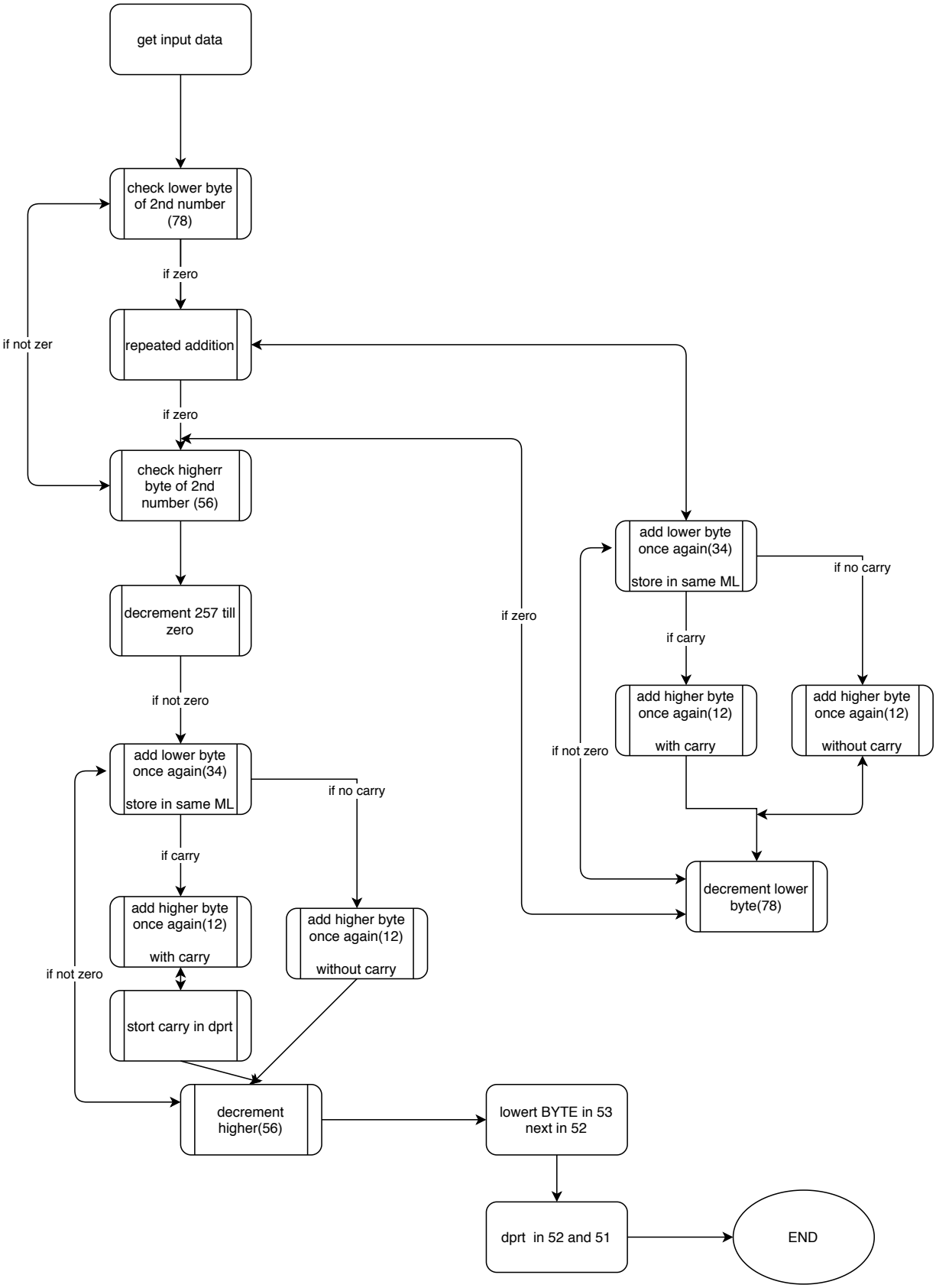
EXPERIMENT 2
Multiplication of 16 bit numbers

Code:

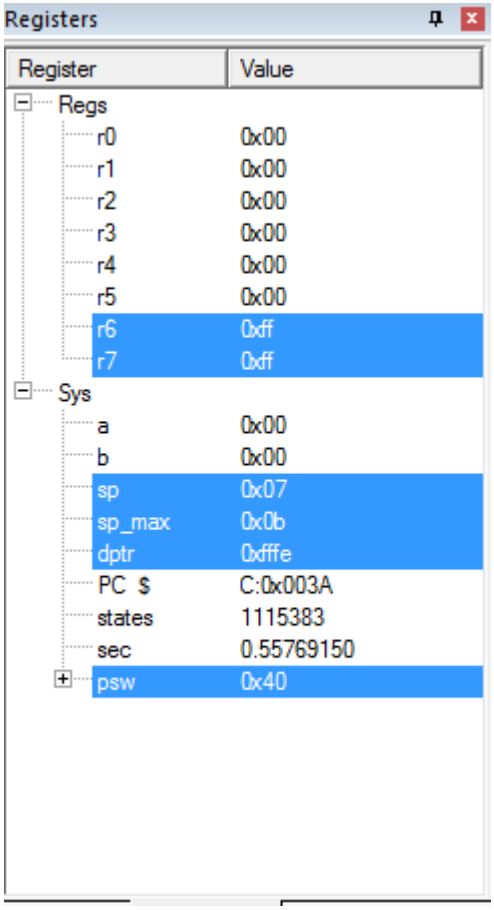
16mul.asm

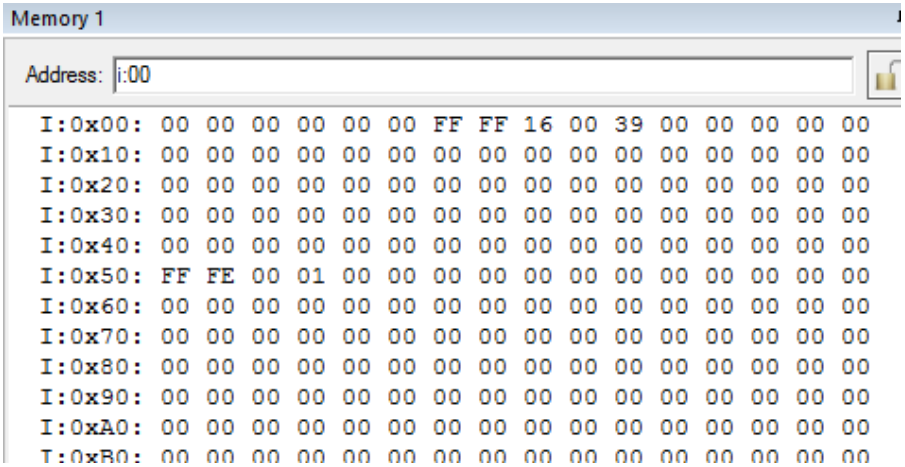
```
; multipling 16 bit number
;1ST NUMBER IS 1234
;2ND NUMBER IS 5678

        MOV R7, #0FFH ; LOWER BYTE OF NUMBER 1
        MOV R6, #0FFH ; UPPER BYTE OF NUMBER 1
        MOV R5, #0FFH ; LOWER BYTE OF NUMBER 2
        MOV R4, #0FFH ; UPPER BYTE OF NUMBER 2

;multipling lower byte of 1st operand

        MOV A,R5
        JZ L8      ; checks if R5 is zero
    L4: LCALL L1   ; repeated addition loop
        DJNZ R5,L4

;multipling higher byte of 1st operand

    L8: MOV A,R4
        JZ  L5     ; checks if R4 is zero
    L6: LCALL L3   ; repeated addition loop
        DJNZ R4,L6

    L5: MOV 51H, DPL ;moves value of DPTR to these registers
        MOV 50H, DPH
        SJMP  L9      ; jumps to last line

;1st SUBROUTINE

    L1: MOV A,53H
        ADD A,R7   ; adds lower byte of 2nd operand
        MOV 53H,A
        MOV A,52H ; 4th byte of result
        ADDC A,R6 ; adds higher byte of 2nd operand
        MOV 52H,A ; 3rd byte of result
        JNC L2
        INC DPTR   ; 1st and 2nd byte of results
        CLR C
    L2: RET

; 2ND SUBROUTINE ( multiplying higher byte by 100H)

    L3: MOV R3, #0FFH
    L7: LCALL L1    ; calls subroutine 255 times
        DJNZ R3,L7
        LCALL L1    ; calls subroutine for 256th time
        RET         ; this sub routine has been executed 100H times

    L9:             ; result is seen at 50H 51H 52H and 53H registers
        END
```

FLOW CHART:


get input data
check lower byte of 2nd number (78)
if zero
repeated addition
if not zer
if zero
check higherr byte of 2nd number (56)
decrement 257 till zero
if not zero
add lower byte once again(34)
store in same ML
if no carry
if carry
add higher byte once again(12)
with carry
add higher byte once again(12)
without carry
if not zero
stort carry in dprt
decrement higher(56)
lowert BYTE in 53 next in 52
dprt in 52 and 51
END
if zero
add lower byte once again(34)
store in same ML
if no carry
if carry
add higher byte once again(12)
with carry
add higher byte once again(12)
without carry
if not zero
decrement lower byte(78)

OUTPUT:





The outputs are found at 50H to 53H registers

EXPERIMENT 2.1

Hexadecimal number to BCD equivalent

Code:

bcd.asm

```
; HEX TO BCD
        MOV  R0, #0FFH ; value to be coverted
        MOV  A , R0    ; store the value in A
        MOV  B , #64H  ; to be divided for MSB
        DIV  AB        ; divide for 100th place
        MOV  R1, A     ; store MSB in R1
        MOV  A , B     ; load A with remainder
        MOV  B , #0AH  ; 10 is the divider
        DIV  AB        ; divide for tenth place
        SWAP A         ; make it the upper nibble
        ADD  A , B     ; add remainder to oneth place
        MOV  R2, A     ; store in R2
        END            ; result in R1 and R2
```

Output:

Register	Value
Regs	
r0	0xff
r1	0x02
r2	0x55
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
Sys	
a	0x55
b	0x05
s.	0x07
s.	0x07
d.	0x0000
P.	C:0x0012
s.	19
s.	0.00000950
p.	0x00

Address: i:00

```
I:0x00: FF 02 55 00 00 00 00 00 00 00 00 00
I:0x10: 00 00 00 00 00 00 00 00 00 00 00 00
I:0x20: 00 00 00 00 00 00 00 00 00 00 00 00
I:0x30: 00 00 00 00 00 00 00 00 00 00 00 00
I:0x40: 00 00 00 00 00 00 00 00 00 00 00 00
I:0x50: 00 00 00 00 00 00 00 00 00 00 00 00
```

input: R0
output: R1 and R2

EXPERIMENT 2.2

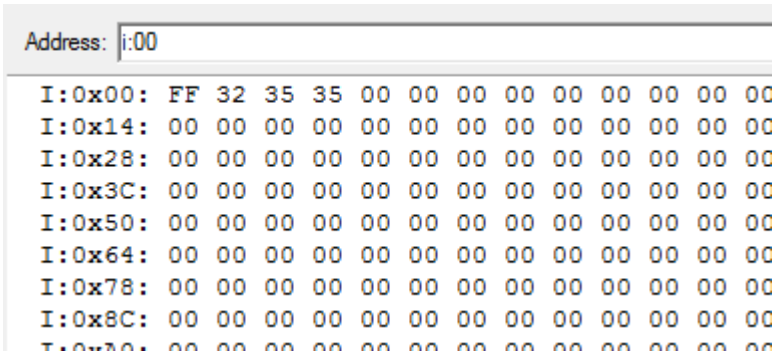
Hexadecimal number to ASCII equivalent

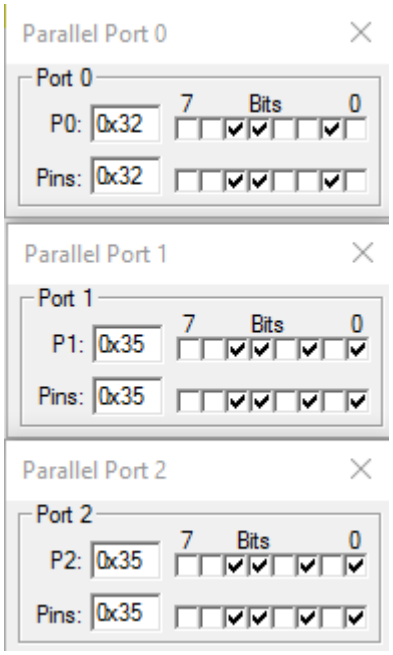
Code:

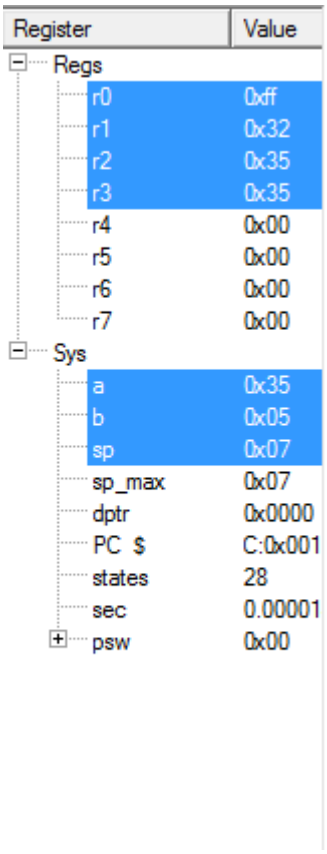
ascii.asm

```
; HEX TO ASCII convertion
        MOV  R0, #0FFH ; value to be coverted
        MOV  A , R0    ; store the value in A
        MOV  B , #64H  ; to be divided for MSB
        DIV  AB        ; divide for 100th place
        ADD  A , #30H  ; convert to ascii
        MOV  R1, A     ; store MSB in R1
        MOV  A , B     ; load A with remainder
        MOV  B , #0AH  ; 10 is the divider
        DIV  AB        ; divide for tenth place
        ADD  A ,  #30H ; convert to ascii
        MOV  R2, A     ; store in R2
        MOV  A , B     ;
        ADD  A ,  #30H ; convert to ascii
        MOV  R3, A     ; store in R2
        MOV  P0, R1    ; show results in ports
        MOV  P1, R2    ;
        MOV  P2, R3    ;
        END            ; result in R1 and R2
```

Output:







EXPERIMENT 2.3

Decimal number to Hexadecimal equivalent

Code:

dec2hex.asm

```
1  ;DECIMAL TO HEXA DECIMAL
2              MOV R0 , #99H          ; enter desired value in R0
3        LOOP:CLR C                   ; clear carry generated from CJNE
4              INC A                  ; inc acc
5              INC R1                 ; inc R1
6              DA A                   ; convert A to decimal equivalent
7              CJNE A , 00H , LOOP    ; compare if A has reached R0
8              MOV P0 , R0            ; display input in R0
9              MOV P1 , R1            ; display output in R1
10             END
```

Outputs:

various values of inputs have been displayed:

port 0 is input and port 1 is output

DECIMAL	HEX
99	**63H**
32	**20H**
02	**02H**
15	**0FH**

EXPERIMENT 3.1

Sorting of ten 8-bits numbers stored in internal data memory in ascending order

Code:

```
        ORG 0000H
        ACALL READ      ;copy data loop
        MOV R1,#0AH     ;number of bytes to process
AGAIN:  MOV A,R1
        MOV R2,A        ;number of bytes
        MOV R0,#30H     ;starting address of data
BACK:   MOV A,@R0       ;1st valur to acc
        INC R0          ;next byte
        MOV B,@R0       ;2nd value to B
        CLR C           ;carry from previous process
        SUBB A,B        ;compare 2 nos
        JC SKIP         ;skip swapping
        MOV B,@R0       ;put 2nd no in B
        DEC R0          ;previous byte
        MOV A,@R0       ;put 1st no in A
        MOV @R0,B       ;swap 2nd no
        INC R0          ;next byte
        MOV @R0,A       ;swap 1st no
SKIP:   DJNZ R2,BACK    ;repeat for next position
        DJNZ R1,AGAIN   ;repeat for next number
        SJMP LAST       ;end statement

READ:   MOV R0,#30H     ;1st byte of source
        MOV R1,#20H     ;1st byte of destination
        MOV R6,#0AH     ;number of bytes
COPY:   MOV A,@R0       ;copying input ...
        MOV @R1,A       ;   for reference
        INC R1          ;next byte
        INC R0          ;next byte
        DJNZ R6,COPY    ;repeat for n bytes
        RET             ;return to main program

LAST:   NOP             ;close program
        END
```

Output:

Internal memory:

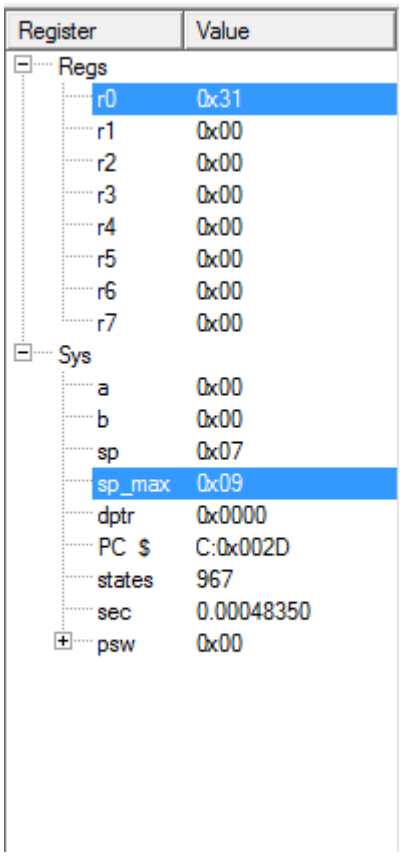
before running the code:

```
I:0x20: 00 00 00 00 00 00 00 00 00 00
I:0x30: 10 12 14 10 00 FF A0 06 02 13
I:0x40: 00 00 00 00 00 00 00 00 00 00
```

after running the code:

```
I:0x20: 10 12 14 10 00 FF A0 06 02 13 00
I:0x30: 00 00 02 06 10 10 12 13 14 A0 FF
I:0x40: 00 00 00 00 00 00 00 00 00 00 00
```

SFRs:



EXPERIMENT 3.2

Sorting of ten 8-bits numbers stored in internal data memory in descending order

Code:

```
        ORG 0000H
        ACALL READ      ;copy data loop
        MOV R1,#0AH     ;number of bytes to process
AGAIN:  MOV A,R1
        MOV R2,A        ;number of bytes
        MOV R0,#3AH     ;ending address of data
BACK:   MOV A,@R0       ;1st valur to acc
        DEC R0          ;prev byte
        MOV B,@R0       ;2nd value to B
        CLR C           ;carry from previous process
        SUBB A,B        ;compare 2 nos
        JC SKIP         ;skip swapping
        MOV B,@R0       ;put 2nd no in B
        INC R0          ;next byte
        MOV A,@R0       ;put 1st no in A
        MOV @R0,B       ;swap 2nd no
        DEC R0          ;prev byte
        MOV @R0,A       ;swap 1st no
SKIP:   DJNZ R2,BACK    ;repeat for next position
        DJNZ R1,AGAIN   ;repeat for next number
        SJMP LAST       ;end statement

READ:   MOV R0,#30H     ;1st byte of source
        MOV R1,#20H     ;1st byte of destination
        MOV R6,#0AH     ;number of bytes
COPY:   MOV A,@R0       ;copying input ...
        MOV @R1,A       ;   for reference
        INC R1          ;next byte
        INC R0          ;next byte
        DJNZ R6,COPY    ;repeat for n bytes
        RET             ;return to main program

LAST:   NOP             ;close program
        END
```

Output:

Internal memory:

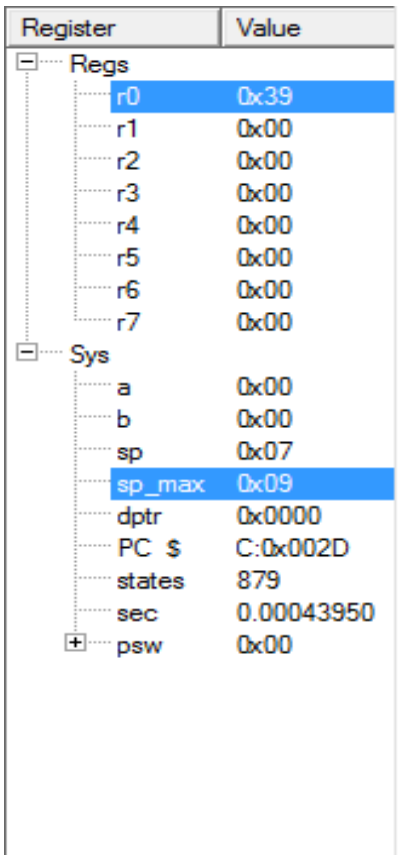
before running the code:

```
I:0x20: 00 00 00 00 00 00 00 00 00 00
I:0x30: 3A 12 FF AC 10 00 12 10 13 01
I:0x40: 00 00 00 00 00 00 00 00 00 00
```

after running the code:

```
I:0x20: 3A 12 FF AC 10 00 12 10 13 01 (
I:0x30: FF AC 3A 13 12 12 10 10 01 00 (
I:0x40: 00 00 00 00 00 00 00 00 00 00 (
```

SFRs:



EXPERIMENT 4

Write an assembly language program to generate the delay of X microseconds and toggle the Y bit of port 1. Using KEIL software.

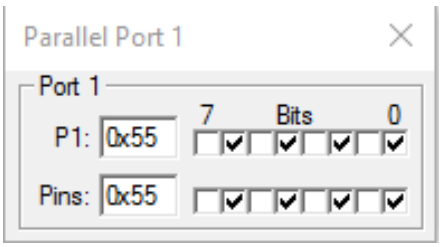
Code:

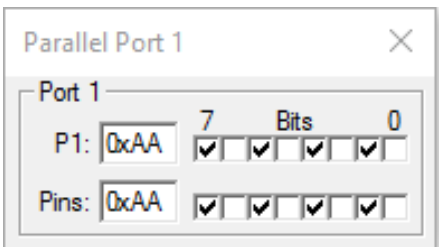
delay.asm*

```
1     ;delay and toggle P1
2     L3:      MOV P1,#55H ; SET ODD BITS OD P1
3              LCALL DELAY ; CALLS DELAY
4              MOV P1,#0AAH ; SETS EVEN BITS OF P1
5              LCALL DELAY
6              SJMP L3 ;REPEATS THE PROGRAM
7
8     DELAY:   MOV R2,#02H      ;CREATES A SEC DELAY
9     L2:      MOV R1,#0FFH
10    L1:      MOV R3,#0FFH
11    L4:      MOV R4,#0FFH
12    L5:      DJNZ R4,L5
13             DJNZ R3,L4
14             DJNZ R1,L1
15             DJNZ R2,L2
16             RET
17
18             END
19
```

Output:

with a delay of 1 sec





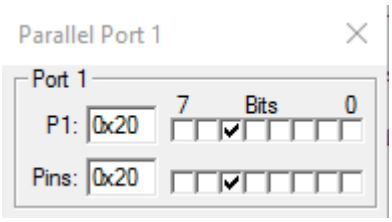
EXPERIMENT 5.1

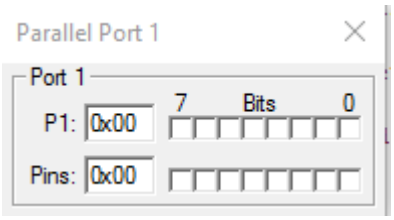
Delay of 250 microseconds and toggle the bit P1.5 using 16 bit timer mode

Code:

```
; 250 us DELAY
; 250 us / 1.085 us = 230 and 65536 - 230 = 65306
; 65306 which in hex is FF1A H
; TL = 1A and TH = FF
        ORG 0000H
        MOV  P1   , #00H ; Clearing the Port
        MOV  TMOD, #10H ; Timer 1 , Mod 1
AGAIN:MOV  TL1 , #1AH ; Lower byte
        MOV  TH1 ,#0FFH ; Higher byte
        SETB TR1        ; Start timer
 BACK:JNB  TF1 ,BACK    ; Check timer status
        CLR  TR1        ; Stop timer
        CPL  P1.5       ; Compliment bit
        CLR  TF1        ; Reset flag
        SJMP AGAIN      ; Reload
        END
```

Output:





The port toggles between these two states every 250us.

EXPERIMENT 5.2

Delay of 4 seconds and toggle the bit P1.0 using 8 bit auto reload timer mode

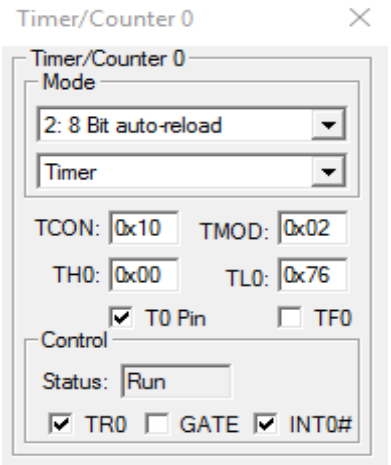
Code:

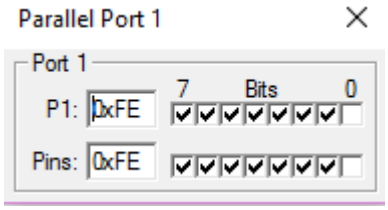
```
1  ; 4 SEC delay using mode 2 timer 0
2
3  ; TH = 0 implies 256 loops
4  ; 4 SEC / 256*0.5425 us = 28800
5  ; 0.5425 is used as its 24MHz
6  ; 28800 = 144 ( 90H ) * 200 ( CD H )
7
8              ORG 0000H
9              MOV TMOD , #02H ; TIMER 0 MODE 2
10             MOV TH0  , #00H ; 256 loops
11       LOOP: MOV R5   , #90H
12             ACALL DELAY
13             CPL P1.0        ; compliment value
14             SJMP LOOP
15
16     DELAY: MOV R4   , #0CDH
17        L1: SETB TR0         ;start timer
18      BACK: JNB  TF0, BACK   ;check status
19             CLR TR0         ;stop timer
20             CLR TF0         ;reset timer
21             DJNZ R4, L1
22             DJNZ R5, DELAY  ; 4 sec delay
23             RET
24
25             END
```

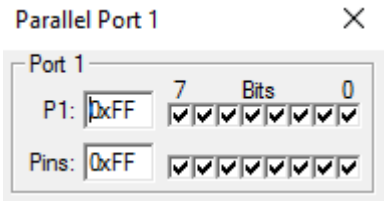
Output:

TMOD REG:

The port toggles between these two states every 4 s.







EXPERIMENT 6

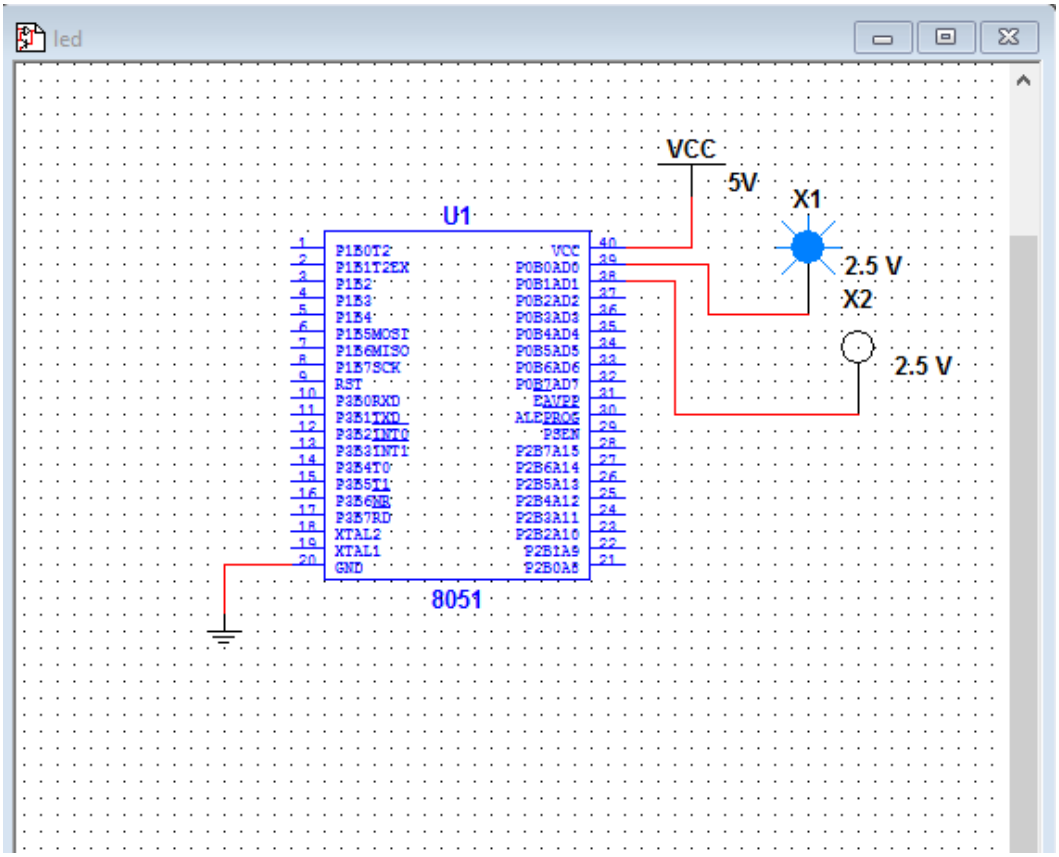
Toggling two LEDS in such fashion when one is on the Another is off for equal time delay

Code:

led.asm

```
$MOD51  ; This includes 8051 definitions

LOOP:    SETB P0.1
         CLR  P0.0
         ACALL DELAY
         CLR P0.1
         SETB P0.0
         ACALL DELAY
         SJMP LOOP

    DELAY: MOV R1,01H
       L1: DJNZ R1,L1
           RET

          END
```

Outputs:

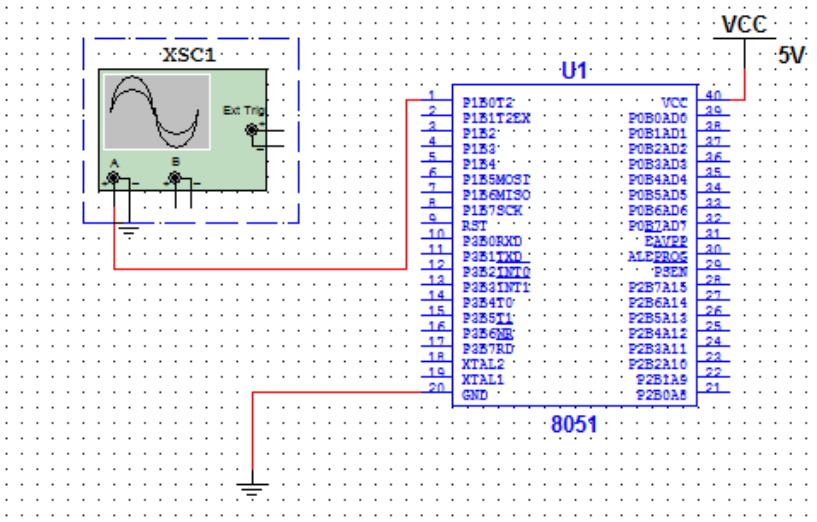


EXPERIMENT 7
Generating a square wave of bit P1.0 shown on oscilloscope.

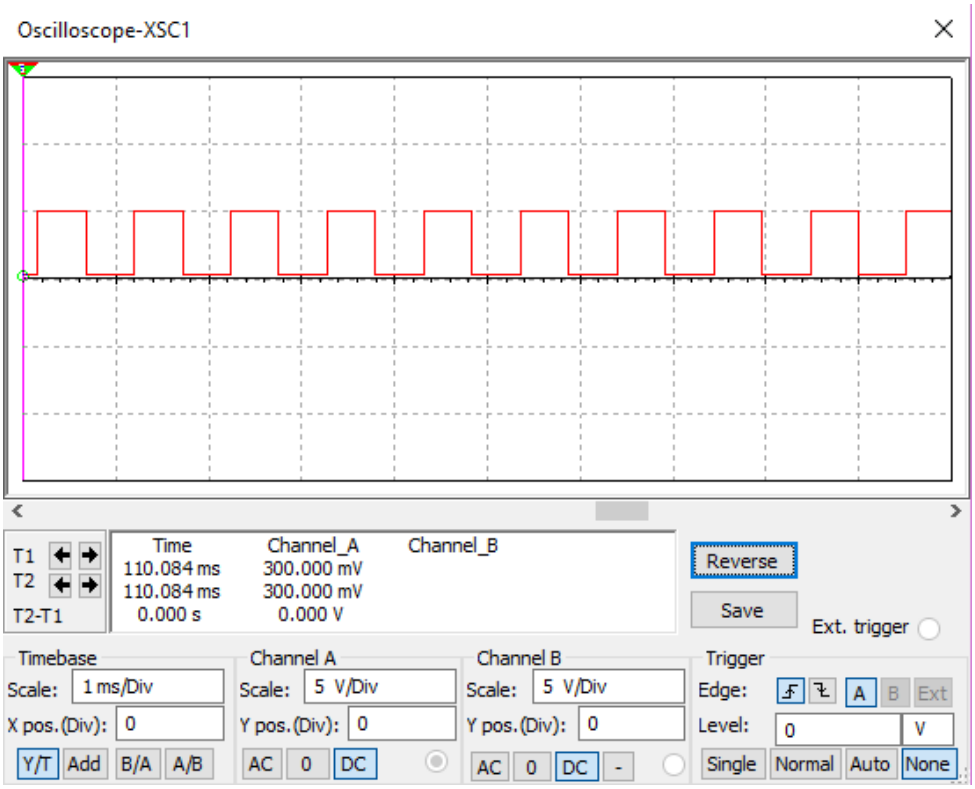
Code:

```
$MOD51

    LOOP:SETB P1.0
         ACALL DELAY
         CLR P1.0
         ACALL DELAY
         SJMP LOOP

     DELAY: MOV R1,01H
        L1: DJNZ R1,L1
            RET

          END
```

Circuit:



Outputs:



EXPERIMENT 8
Seven segment display and 2x2 matrix keyboard

Code:

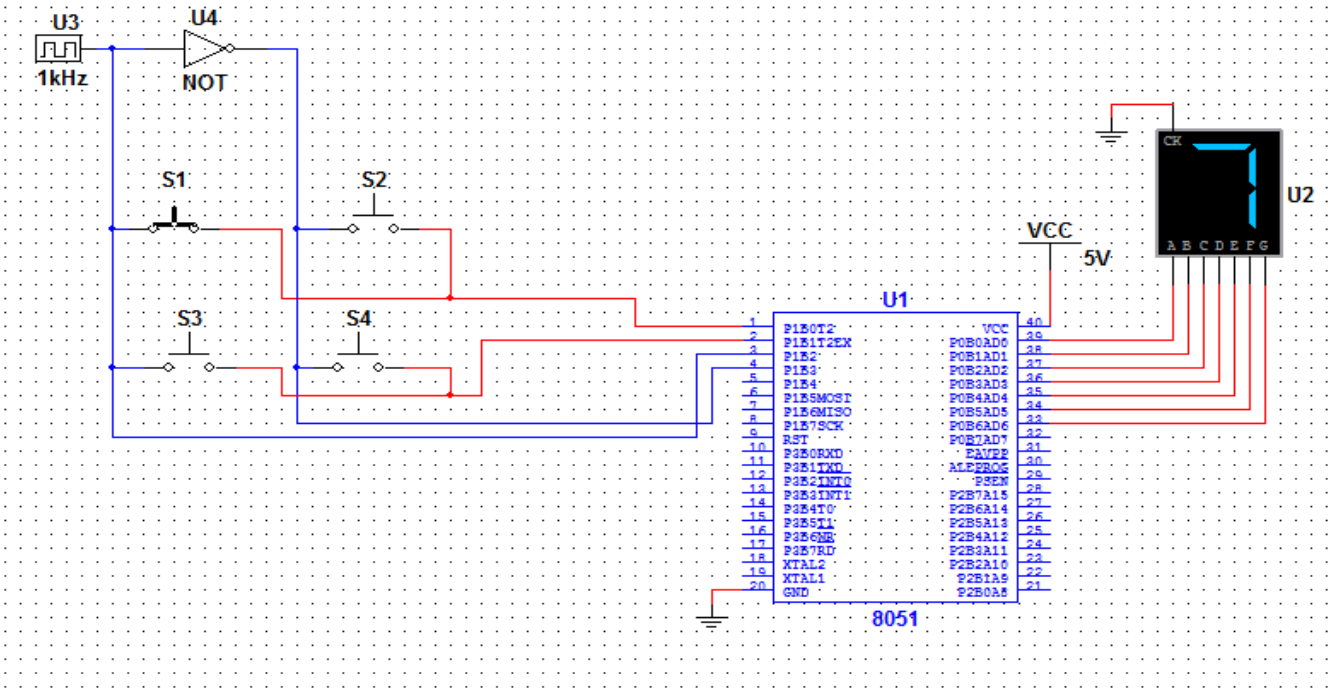
```
$MOD51
        MOV P1,#00H
        MOV P0,#00H

loop:   MOV A , #00H
; CHECK THE VALUE OF INPUT
        JNB P1.0, L1
        ADD A , #01H

    L1: JNB P1.1, L2
        ADD A , #02H

    L2: JNB P1.2, L3
        ADD A , #04H

    L3: JNB P1.3, L4
        ADD A , #08H

; MAP THE INPUT TO OUTPUT
    L4: MOV B, A
        CJNE A,#05H ,L5
        MOV P0,#07H
    LA: JNB P1.0, LX
        JB P1.2, LA
    LX: LJMP L16

    L5: MOV A,B
        CJNE A,#06H ,L6
        MOV P0,#4FH
    LB: JNB P1.0, LY
        JB P1.3, LB
    LY: LJMP L16

    L6: MOV A,B
        CJNE A,#09H ,L7
        MOV P0,#07H
    LC: JNB P1.1, LZ
        JB P1.2, LC
    LZ: LJMP L16

    L7: MOV A,B
        CJNE A,#0AH ,L16
        MOV P0,#06H
    LD: JNB P1.1, L16
        JB P1.3, LD

   L16: LJMP loop


END
```

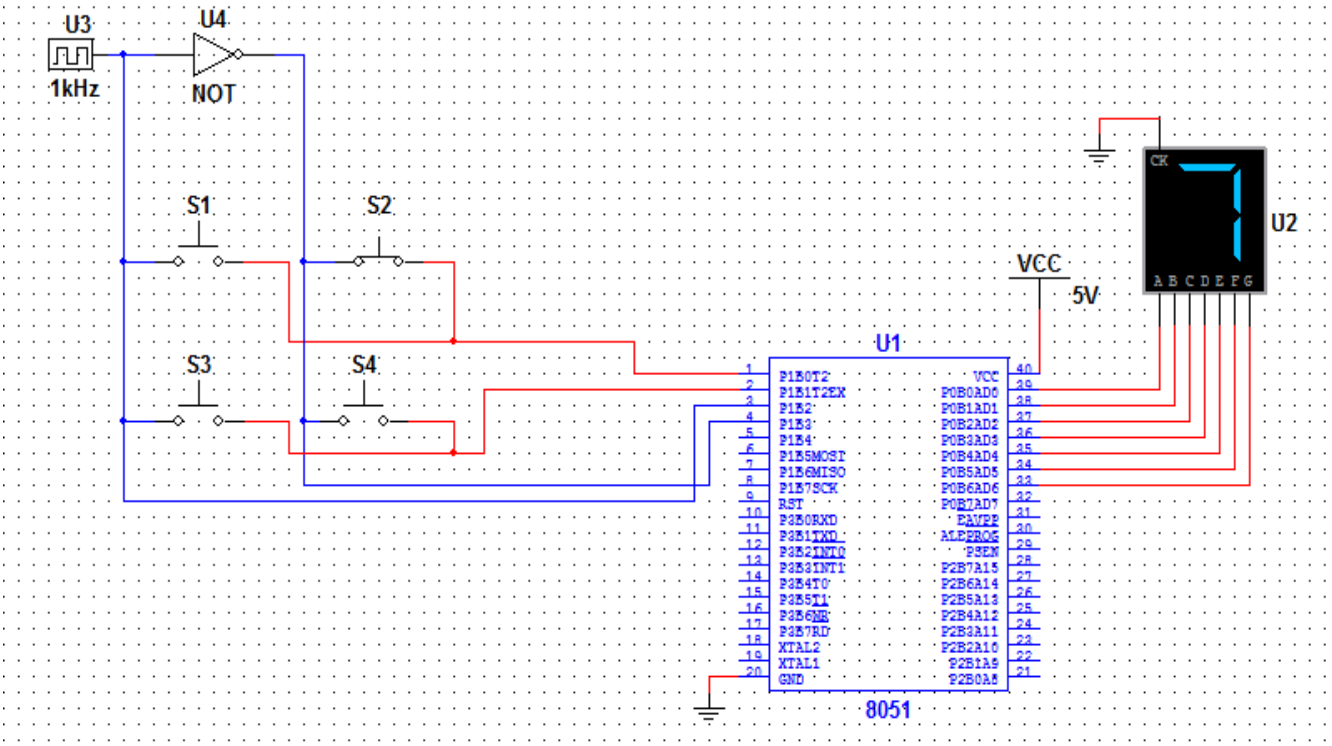
Theory:

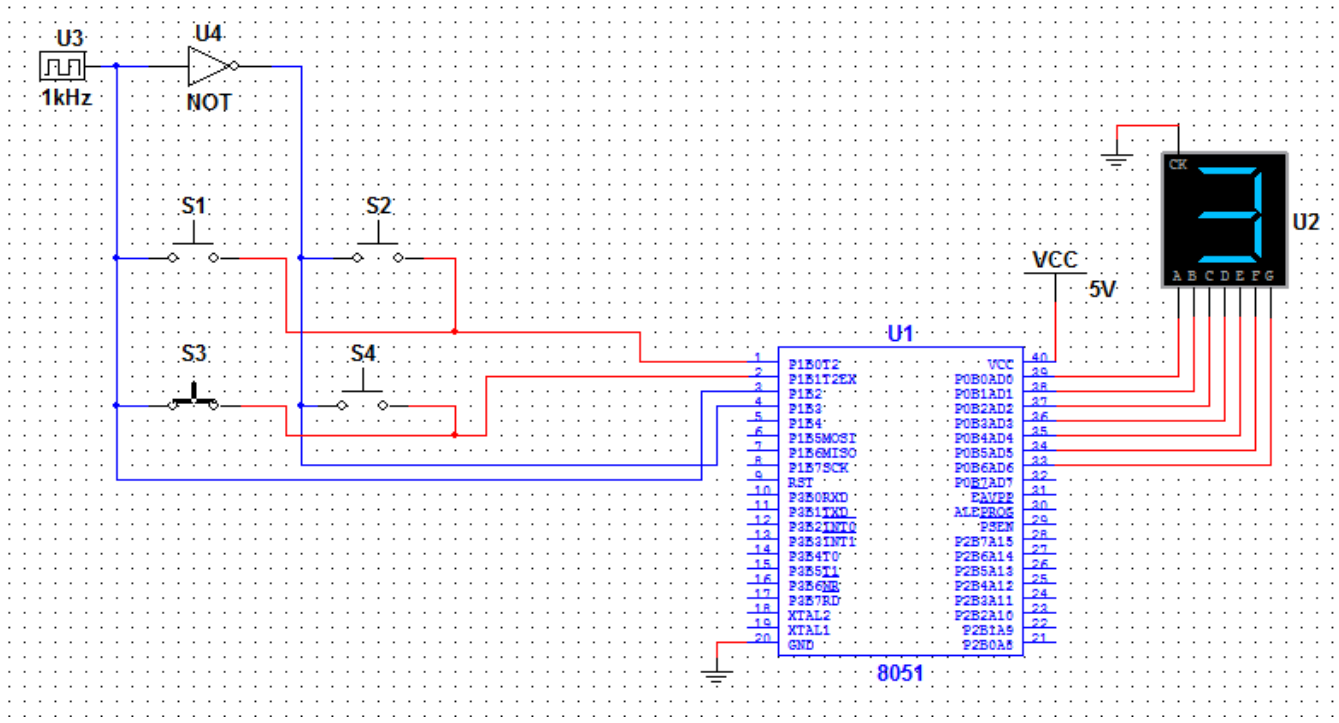
Above, the button at B1 is held. As the scan progresses, it does the following:

1. Nothing is selected. Outputs A and B are both high, and we don't worry about inputs 1 and 2.
2. Column A is selected with a logical low.
    - The system reads inputs 1 and 2. Both are open, so the pull-up resistors cause the inputs to be pulled high. Since this is an active low matrix, the high inputs indicate nothing is pressed.
3. The system deselects column A by driving it high and selects column B is by driving it low.
    - The system reads inputs 1 and 2. Since switch B1 is held, the low level from the column selection shows up at input 1.
    - By pairing the column output (B) with the detected switch (1), the system knows that switch B1 is pressed.
4. Finally, everything is deselected by driving both outputs high.

Circuit and Output:




U3
1kHz
U4
NOT
S1
S2
S3
S4
VCC
5V
U1
8051
U2
CK
A B C D E F G
P1.0/T2
P1.1/T2EX
P1.2
P1.3
P1.4
P1.5/MOSI
P1.6/MISO
P1.7/SCK
RST
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
XTAL2
XTAL1
GND
VCC
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
EA/VPP
ALE/PROG
PSEN
P2.7/A15
P2.6/A14
P2.5/A13
P2.4/A12
P2.3/A11
P2.2/A10
P2.1/A9
P2.0/A8


U3
1kHz
U4
NOT
S1
S2
S3
S4
VCC
5V
U1
8051
U2
CK
A B C D E F G
P1.0/T2
P1.1/T2EX
P1.2
P1.3
P1.4
P1.5/MOSI
P1.6/MISO
P1.7/SCK
RST
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
XTAL2
XTAL1
GND
VCC
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
EA/VPP
ALE/PROG
PSEN
P2.7/A15
P2.6/A14
P2.5/A13
P2.4/A12
P2.3/A11
P2.2/A10
P2.1/A9
P2.0/A8

EXPERIMENT 9
16x1 LCD DISPLAY

Code:

```
$MOD51 ; this includes 8051
ORG 0000
;STORING NAME IN 60H TO 6FH
        MOV R1 ,#60H
        MOV @R1,#'T'
        INC R1
        MOV @R1,#' '
        INC R1
        MOV @R1,#'R'
        INC R1
        MOV @R1,#'A'
        INC R1
        MOV @R1,#'J'
        INC R1
        MOV @R1,#'A'
        INC R1
        MOV @R1,#' '
        INC R1
        MOV @R1,#'A'
        INC R1
        MOV @R1,#'A'
        INC R1
        MOV @R1,#'D'
        INC R1
        MOV @R1,#'H'
        INC R1
        MOV @R1,#'I'
        INC R1
        MOV @R1,#'T'
        INC R1
        MOV @R1,#'H'
        INC R1
        MOV @R1,#'A'
        INC R1
        MOV @R1,#'N'
        MOV A,#38H    ; initialize LCD
        ACALL COMNWRT ; Call command Subroutine
        MOV A, #0EH   ; Display on, cursor on.
        ACALL COMNWRT ; Call command Subroutine.
        MOV A, # 01   ; Clear LCD.
        ACALL COMNWRT ; Call command subroutine
        MOV A, #80H ; Cursor at line 1 position 0
        ACALL COMNWRT ; Call command subroutine.

; // MESSAGE DISPLY
        MOV R0,#16
        MOV R1,#60H
   LOOP:MOV A,@R1
        ACALL DATAWRT
        INC R1
        DJNZ R0,LOOP
 AGAIN: SJMP AGAIN

COMNWRT:MOV P1, A
        CLR P3.1; RS=0 FOR COMMAND WRITE
        CLR P3.0; R/W=0FOR WRITE
        SETB P3.2; E=1 FOR HIGH PUSLSE
        CLR P3.2 ;E=0 FOR H-TO-L PULSE
        RET

DATAWRT:MOV P1, A; WRITE DATA TO LCD
        SETB P3.1; RS=1 FOR DATA
        CLR P3.0; R/W=0 FOR WRITE
        SETB P3.2; E=1 FOR HIGH PULSE
        CLR P3.2; E=0 FOR H-TO-L PULSE
        RET
        END
```

Theory:

16×1 LCD module is a very common type of LCD module that is used in 8051 based embedded projects. It consists of 16 rows and 1 column of 5×7 or 5×8 LCD dot matrices. The module were are talking about here is a type which is a very popular one . It is available in a 16 pin package with back light ,contrast adjustment function and each dot matrix has 5×8 dot resolution.

Circuit and Output: