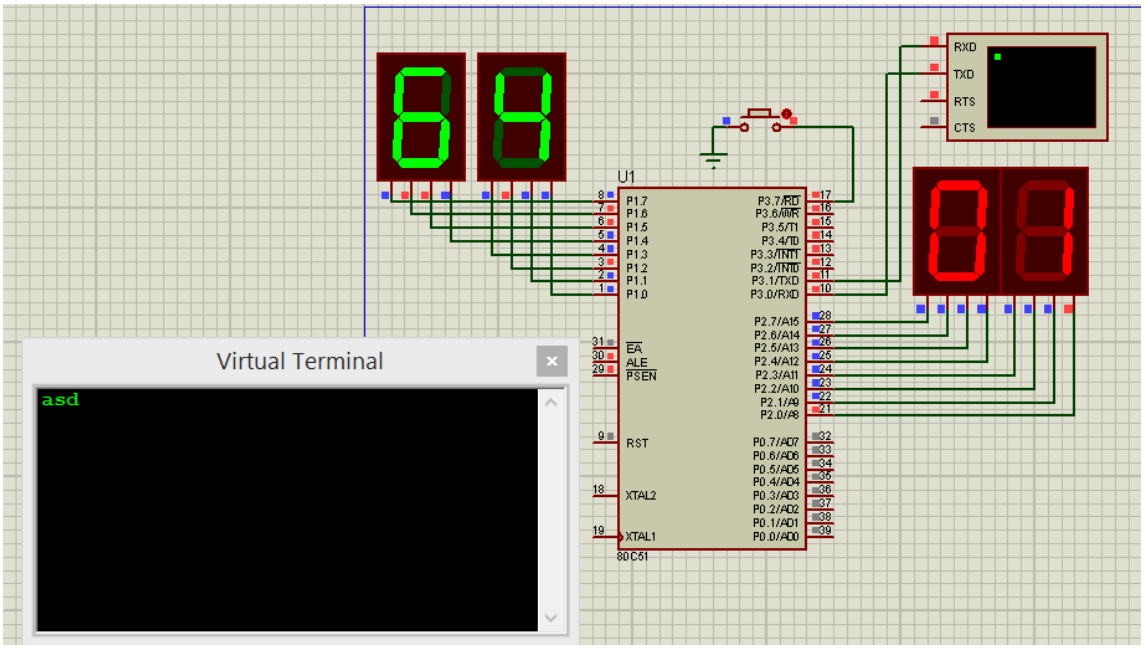
Problem 21



In this problem, we implement a serial cyclic buffer

Each character entered in the terminal will be saved in this buffer

P2 displays the number of character available in the buffer

A button is used to read the data from the buffer and display it on P1 as will as sending it to the terminal

The buffer size is chosen to be 16, so program will not accept any character if the buffer is full. Also it will not execute read from buffer (when pushing the button) if the buffer is empty.

Variables

```
1  SERIAL_START EQU 40H            ; keyboard buffer at 40h - 4Fh (local)
2  SERIAL_END EQU 50H          ;
3  WRITE_POINTER EQU 30h    ; keybd write pointer (local)
4  READ_POINTER EQU 31h        ; keybd buff read pntr (local)
5  COUNT EQU 32H
6  READ_BUTTON EQU P3.7
```

1,2 → The starting and end address of the buffer

3,4 → write pointer and read pointer variables

5→ count holds the available character in the buffer

6→ The port pin used to read the "read button"

Main code

```
8     ORG 00H
9     JMP START
10  ;=========================================================
11  ORG 23H
12     PUSH PSW
13     JMP SERIAL_INT
14  ;=========================================================
15  START:
16     CALL INIT_SERIAL
17     CALL RESET_BUFFER
18     MOV P1,#0
19  START2:
20     MOV P2,COUNT
21     JNB READ_BUTTON, READ_SERIAL
22     JMP START2
23  READ_SERIAL:
24     CALL GET_KEY
25     MOV P1, A
26     CALL SEND_CHAR
27     MOV R5,#3
28     CALL DELAY_100MS
29     JMP START2
```

We use the serial interrupt→ will be triggered when a character is sent/received- but we will use it on character reception.

The interrupt vector of the serial interrupt is at address 23h, so we save the processor status word (12), and then jump to actual interrupt handler (13)

At start, we initialize the serial as before (16), then we reset the buffer pointer to buffer_start (17)

The main loop, will first display the character count (20), then test the read_button to do read operation (21). If the read_button is pressed we call the function "GET_KEY" to read from serial buffer (24), then display it on P1 (25), send it serially (26), wait for 300ms to allow for key release/debounce(27,28).

Functions

1- Serial_INT

```
31  SERIAL_INT:
32     JBC RI,GOT_RI       ; check for character in buffer
33     POP PSW            ; restore the flags
34  RETI
35
36  GOT_RI:
37     PUSH ACC             ; save the acc
38     PUSH 00h
39     MOV A,WRITE_POINTER ; get the write pointer value
40     INC A               ; see if right behind read pointer
41     CJNE A, #SERIAL_END, ROLL_OK
42     MOV A, #SERIAL_START
43  ROLL_OK:
44     CJNE A, READ_POINTER,BUFF_OK ; if so then do not accept
45     JMP SERIAL_EXIT
46          ;
47  BUFF_OK:
48     CALL INC_BCD
49     MOV R0,WRITE_POINTER    ; load the keyboard pointer
50     MOV A,SBUF              ; get the character waiting
51     MOV @R0,A               ; save the character
52     INC WRITE_POINTER       ; increment the write pointer
53     MOV A,WRITE_POINTER
54     CJNE A, #SERIAL_END,SERIAL_EXIT ; if write not at end of buffer then ok
55     MOV WRITE_POINTER,#SERIAL_START ; else roll write (keybd buff 10h-1fh)
56  SERIAL_EXIT:
```

It is the main routine that will read any character received and save it in the buffer if it is not full.

32 → jbc "jump if bit set and clear it" → will test for the RI bit and if it is set (character received), it will proceed to 37 where we push used register (37,38)

Now before saving the character into the buffer, we check if the write_pointer reaches the serial_end by incrementing the write_pointer and if it is equal serial_end, we roll it back to serial_start (39-42). If write_pointer becomes = read_pointer, this means that the buffer is full, so we end the read process (45,46).

If buffer is not full, we increment the count in bcd (48), then we save the new character in the buffer (49-51). Then we increment the write_pointer and roll it to start if it reaches buffer_end(52-55)

2- Get_key

```
62  GET_KEY:
63          MOV A,READ_POINTER              ; GET READ POINTER
64          CJNE A, WRITE_POINTER,GET_CHAR ; COMPARE TO WRITE POINTER
65          JMP GET_KEY                     ; IF EQUAL (BUFFER EMPTY), WAIT
66  GET_CHAR:
67          PUSH 00H
68          MOV R0,READ_POINTER             ; LOAD READ POINTER
69          MOV A,@R0                       ; GET CHARACTER
70          PUSH ACC                        ; SAVE IN STACK
71          INC READ_POINTER
72          CALL DEC_BCD
73          MOV A,READ_POINTER
74          CJNE A, #SERIAL_END,BUFFER_OK          ; CHEACK FOR BUFFER END
75          MOV READ_POINTER,SERIAL_START  ; RESET POINTER TO BUFFER START
76  BUFFER_OK:
77          POP ACC
78          POP 00H
79          CLR C
80          RET
```

This will read a character from the buffer. First, we wait until buffer is not empty. This is done by comparing read_pointer with write_pointer. If they are equal, this means that buffer is empty(63-65)

If not empty, we read the current character (68-70), then increment the read_pointer(71), decrement the count(72). After decrementing the incrementing the read_pointer, we check for roll over (73-75).

3- init_serial

```
94  INIT_SERIAL:
95     MOV SCON,#50H        ;Asynchronous mode, 8-bit data and 1-stop bit
96     MOV TMOD,#20H        ;Timer1 in Mode2.
97     MOV TH1,#253         ; // Load timer value for baudrate generation = 256 - (12000000)/(32*12*baudrate)
98     MOV TL1,#253
99     SETB TR1     ;      //Turn ON the timer for Baud rate generation
100    SETB ES
101    SETB EA
102    MOV COUNT,#0
103 RET
```

It is the same as serial initialization before, but we enable serial interrupt (100,101)

4-INC_BCD, DEC_BCD

```
118 INC_BCD:
119    PUSH ACC
120    MOV A,COUNT
121    ADD A, #1
122    DA A
123    MOV COUNT,A
124    POP ACC
125 RET
126 ;===============================================================
127 DEC_BCD:
128    PUSH ACC
129    MOV A,COUNT
130    ADD A, #99H
131    DA A
132    MOV COUNT,A
133    POP ACC
134 RET
135 ;===============================================================
136 END
```

INC_BCD is as previous examples

DEC_BCD is done as INC_BCD, but with the addition of 99H instead of 1 (130)

Adding 99h to a BCD number is the same as adding "-1"