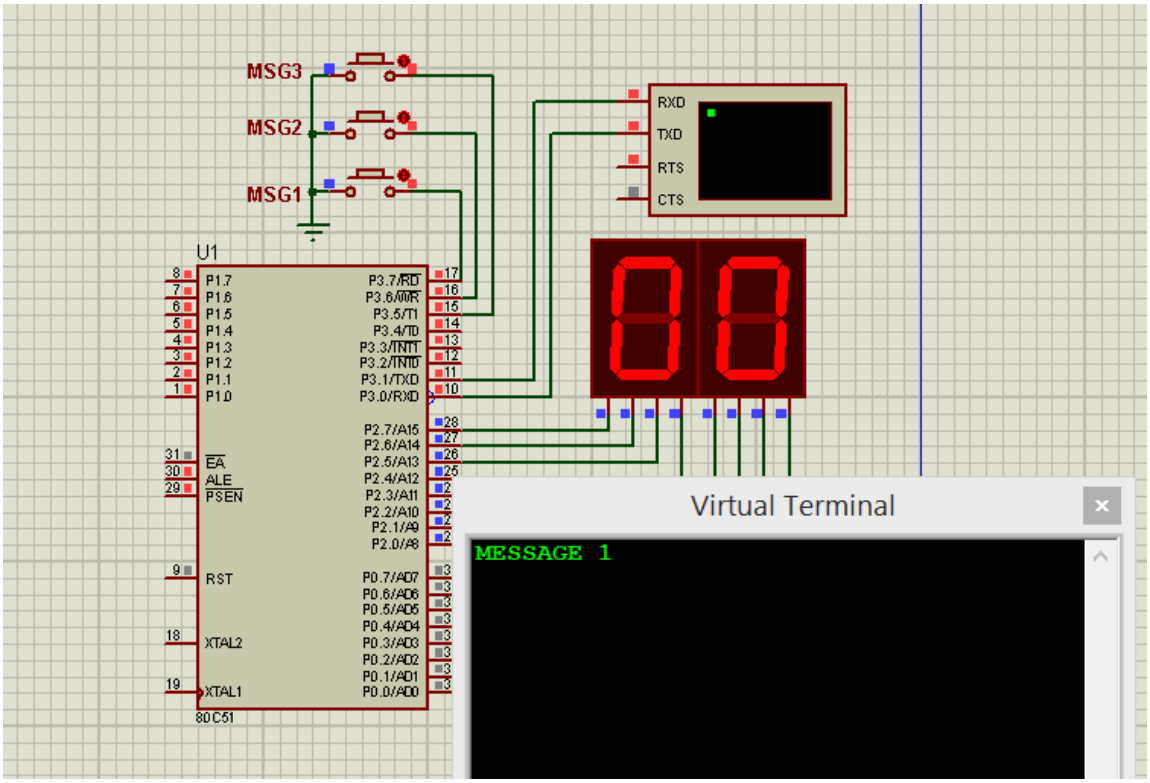
Problem 23



In this problem, send one of 3 message according of the button pressed, but with a cyclic buffer implementation for the transmission.

So, we use a buffer to put the message on it, and then using serial interrupt to send all the characters in the buffer till it becomes empty.

We use the same technique for problem 21 but with the exchange between transmission and reception.

Variables

```
1  NULL EQU 00H
2
3  MSG1_BUTTON EQU P3.7
4  MSG2_BUTTON EQU P3.6
5  MSG3_BUTTON EQU P3.5
6
7
8  SERIAL_START EQU 40H            ; keyboard buffer at 40h - 5Fh (local)
9  SERIAL_END EQU 60H        ;
10 WRITE_POINTER EQU    30h        ; keybd write pointer (local)
11 READ_POINTER EQU    31h        ; keybd buff read pntr (local)
12 COUNT EQU 32H
13
14 BUFFER_EMPTY EQU 00H
15 BUFFER_FULL EQU 01H
16
```

Null→ character that defines the message end

3-5 → send buttons

8-9→ buffer start/end address (as before)

10-11→ read/write pointer (as before)

12→ count → number of characters

14,15 → flags to indicate that the buffer is full/empty

Main code

```
27 START:
28    CALL INIT_SERIAL
29    CALL RESET_BUFFER
30
31 START2:
32    MOV P2,COUNT
33    JNB MSG1_BUTTON,LOAD_MSG1
34    JNB MSG2_BUTTON,LOAD_MSG2
35    JNB MSG3_BUTTON,LOAD_MSG3
36    JMP START2
37 LOAD_MSG1:
38    MOV DPTR,#MSG1
39    JMP DO_SEND
40 LOAD_MSG2:
41    MOV DPTR,#MSG2
42    JMP DO_SEND
43 LOAD_MSG3:
44    MOV DPTR,#MSG3
45 DO_SEND:
46    CALL SEND_MSG
47    MOV R5,#3
48    CALL DELAY_100MS
49    JMP START2
```

28 → initialize the serial at 9600 baud rate with interrupt enable

29 → reset the buffer pointer to buffer_start

33-35→ test for which button is pressed and jump to load DPTR with the corresponding message address(38-44), then execute message_send(46); then pause for 300ms (47-48)

Functions

1-Serial_INT

```
52  SERIAL_INT:
53     JBC TI,GOT_TI        ; check for character in buffer
54     POP PSW              ; restore the flags
55  RETI
56
57  GOT_TI:
58     CALL READ_BUFFER
59     JB BUFFER_EMPTY,END_INT
60     MOV SBUF,A
61  END_INT:
62     POP PSW
63  RETI
```

The same as before, but we will activate the function only for transmission only. So we test for TI (53), and if it is "1" → means last byte has been transmitted, we continue (58) to send another character.

At 58, we call the function read_buffer to read a byte from the buffer and if it is empty (59), we end the interrupt function, else we put it in the SBUF to start transmitting.

2-READ_BUFFER

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

First, we check buffer empty by comparing read_pointer to write_pointer(68); if they are equal → buffer empty → set empty flag and return(69,70)

If not empty → get_char:→ read byte by read_pointer (72,73), then increment read_pointer(75);compare it to serial_end; to roll it back to serial_start if they are equal(78);

3-Write_Buffer:

```
84  WRITE_BUFFER:
85     PUSH ACC
86     CLR BUFFER_FULL
87     MOV A,WRITE_POINTER ; get the write pointer value
88     INC A               ; see if right behind read pointer
89     CJNE A, #SERIAL_END, ROLL_OK
90     MOV A, #SERIAL_START
91  ROLL_OK:
92     CJNE A, READ_POINTER,BUFF_OK ; if so then do not accept
93     SETB BUFFER_FULL
94     JMP SERIAL_EXIT
95          ;
96  BUFF_OK:
97     CALL INC_BCD
98     MOV R0,WRITE_POINTER    ; load the keyboard pointer
99     POP ACC                 ; get the character waiting
100    MOV @R0,A               ; save the character
101    INC WRITE_POINTER       ; increment the write pointer
102    MOV A,WRITE_POINTER
103    CJNE A, #SERIAL_END,SERIAL_EXIT ; if write not at end of buffer then ok
104    MOV WRITE_POINTER,#SERIAL_START ; else roll write (keybd buff 10h-1fh)
105 SERIAL_EXIT:
106 RET
```

This function will load the message into buffer

First, we increment write_pointer and make the roll over check (87-90)

2nd, check for buffer full (read_pointer = write_pointer) (92-94)

If it is no full, restore the byte to be stored (99), then write it to the buffer at the address pointed to by write_pointer (98-100). Then check for roll-over(101-104)

4-SEND_MSG:

```
113  SEND_MSG:
114     CLR A
115     MOVC A,@A + DPTR
116     INC DPTR
117     CJNE A, #NULL, NOT_MSG_END
118     MOV R5,#20
119     CALL DELAY_100MS
120     SETB TI
121     RET
122  NOT_MSG_END:
123     CALL WRITE_BUFFER
124     JMP SEND_MSG
```

It reads message characters until it finds the null character (114-117); then write each character to the buffer using "WRITE_BUFFER" (123).
We impose a delay between character sending to catch up the display (118-119)