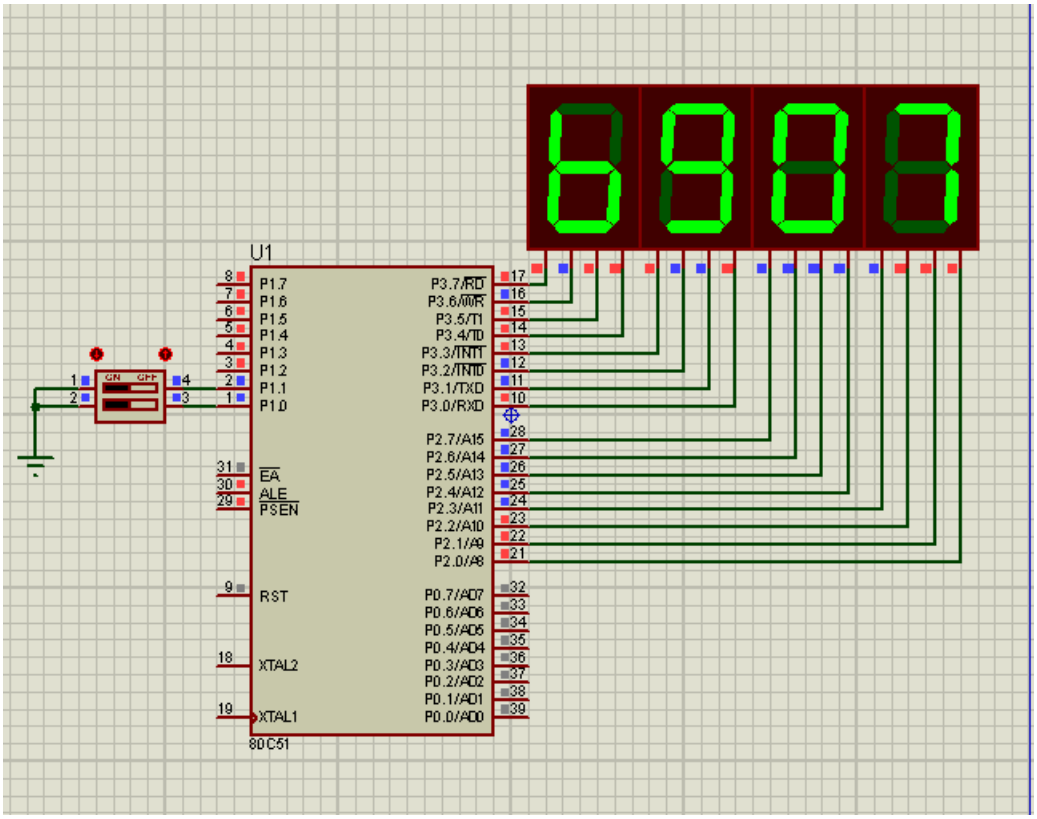
Problem 27



In this problem we add two number with a given length and base

The number base is determined by the dip switch connected to port P1

It has 4 state

00 → hex 2 digits/byte

01→ hex 1 digit/byte

10→ bcd 1 digit/byte

11→ bcd 2 digits/byte

The number are stored internally as follows (for example)

```
194  ;NUM1: DB 12H,75H
195  ;NUM2: DB 14H,16H
196
197  ;NUM1: DB 9H,4H,2H,5H
198  ;NUM2: DB 7H,3H,9H,3H
199
200  NUM1: DB 94H,25H
201  NUM2: DB 73H,93H
```

197,198 → for 1 digit hex or 1 digit BCD

200,201 → for 2 digits hex or bcd

Variables

```
1  LEN EQU 2
2
3  TEMP1 EQU 30H
4  TEMP2 EQU 34H
5
6  RESULT EQU 38H
7  TEMP EQU 40H
8
9  RC EQU 00H
```

1→ number of bytes for a number limited here for 4 bytes

3,4→ temporary storage for num1 and num2

6→ store the result of addition

9→ temp storage of the generated carry or ripple carry

Main code

```
12 START:
13    CALL LOAD_NUMBERS
14    MOV A,P1
15    ANL A,#00000011B
16    CJNE A,#0,NOT0
17    CALL BIN_BASE
18    JMP START
19 NOT0:
20    CJNE A,#1,NOT1
21    CALL ONE_HEX_BASE
22    JMP START
23 NOT1:
24    CJNE A,#2,NOT2
25    CALL ONE_BCD_BASE
26    JMP START
27 NOT2:
28    CJNE A,#3,NOT3
29    CALL TWO_BCD_BASE
30    JMP START
31 NOT3:
32    MOV P2,#0
33    MOV P3,#0
34
35    JMP START
```

First we load numbers into RAM (13) → temp1, temp2

14,15 read first two bits of P1

16,20,24,28 → determine if it is 0 (bin base) or 1 (one hex) or 2 (one BCD) or 3 (two BCD) else; we make both displays 00

Functions:

1-BIN_BASE

```
37  BIN_BASE:
38     MOV R7,#LEN
39     MOV R0,#TEMP1
40     MOV R1,#TEMP2
41     MOV B,#RESULT
42     CLR RC
43  ALL1:
44     MOV A,@R0
45     MOV C,RC
46     ADDC A,@R1
47     MOV RC,C
48     PUSH 01H
49     MOV R1,B
50     MOV @R1,A
51     POP 01H
52     INC R0
53     INC R1
54     INC B
55     DJNZ R7,ALL1
56
57     CALL DISP_RESULT
58  RET
```

It will add byte by byte in binary form → same as two hex/byte

R7 → number of bytes (38)

R0→ points to first number

R1→ points to 2nd number

B → points to result

We make a loop with the number of byte (43,55)

In this loop we read a byte of the first number (44)

add it with previous carry to a byte of the 2nd number (45,46)

preserve the carry in CR (47)

preserve R1 (48) where we need it to point to the result;

store A (addition result) into current byte location in the RESULT (49-50)

restore R1 (51)

52-54 → increment all pointers by 1

After finishing the addition of all bytes, we display the result as hex digit (57)

2- ONE_HEX_BASE

```
61      MOV R7,#LEN
62      MOV R0,#TEMP1
63      MOV R1,#TEMP2
64      MOV B,#RESULT
65      CLR RC
66  ALL2:
67      MOV A,@R0
68      ANL A,#0FH
69      MOV TEMP,A
70      MOV A,@R1
71      ANL A,#0FH
72      MOV C,RC
73      ADDC A,TEMP
74      ANL A,#0FH
75      JNB PSW.6,SKIP1
76      SETB RC
77  SKIP1:
78      PUSH 01H
79      MOV R1,B
80      MOV @R1,A
81      POP 01H
82      INC R0
83      INC R1
84      INC B
85      DJNZ R7,ALL2
86
87      CALL DISP_RESULT_HEX1
```

This function will use one hex digit/byte

61-65 pointer initialization as before

The difference here, is that we ensure that upper nibble is zero by using the ANL instruction (67,71); and hence we need to store the value of the first number before getting the 2nd (69). Also the higher nibble of the result is cleared (74).

But now to store the carry → in one digit/byte → carry will be from first nibble to 2nd nibble→ fortunately this is called Auxiliary carry and is stored in the PSW bit number 6, hence we check this bit (75) and sets RC if it is 1. Then we continue as before by storing the result byte and updating the pointers(78-84).

Finally we use disp_result_hex1 → this function will convert result to two digit/byte for proper display.

3-ONE_BCD_BASE

```
90  ONE_BCD_BASE:
91     MOV R7,#LEN
92     MOV R0,#TEMP1
93     MOV R1,#TEMP2
94     MOV B,#RESULT
95     CLR RC
96  ALL3:
97     MOV A,@R0
98     ANL A,#0FH
99     MOV TEMP,A
100    MOV A,@R1
101    ANL A,#0FH
102    MOV C,RC
103    ADDC A,TEMP
104    DA A
105    CLR C
106    PUSH ACC
107    SUBB A,#10
108    POP ACC
109    JC BCD_OK
110    SETB RC
111    ANL A,#0FH
112    JMP SKIP3
```

```
113 BCD_OK:
114    CLR RC
115 SKIP3:
116    PUSH 01H
117    MOV R1,B
118    MOV @R1,A
119    POP 01H
120    INC R0
121    INC R1
122    INC B
123    DJNZ R7,ALL3
124
125    CALL DISP_RESULT_HEX1
126 RET
```

It is very similar to one_hex_base except for the following

Using decimal adjust after addition (104)

Carry must be produced if the result > 9; so we subtract 10 from the result and test if a carry occurs → means result<10 → no adjustment needed (105-109)

If no carry occurs → result >=10 we need to preserve the carry (CR=1) and clear the upper nibble (110-111)

The remaining code is the same as one_hex_base

4-TWO_BCD_BASE

```
128 TWO_BCD_BASE:
129    MOV R7,#LEN
130    MOV R0,#TEMP1
131    MOV R1,#TEMP2
132    MOV B,#RESULT
133    CLR RC
134 ALL4:
135    MOV A,@R0
136    MOV C,RC
137    ADDC A,@R1
138    MOV RC,C
139    DA A
140    PUSH 01H
141    MOV R1,B
142    MOV @R1,A
143    POP 01H
144    INC R0
145    INC R1
146    INC B
147    DJNZ R7,ALL4
148
149    CALL DISP_RESULT
150 RET
```

It is the same as binary/two_hex-base except that we make use of the DA instruction to convert result to BCD format (139)

5-display routines

```
181 DISP_RESULT:
182    MOV P2,RESULT
183    MOV P3,RESULT+1
184 RET
185 ;=========================================
186 DISP_RESULT_HEX1:
187    MOV A,RESULT+1
188    SWAP A
189    ADD A,RESULT
190    MOV P2,A
191
192    MOV A,RESULT+3
193    SWAP A
194    ADD A,RESULT+2
195    MOV P3,A
196 RET
197 ;=========================================
```

We have two display routine → one for two digits/byte and it just send each byte of the result to P2 and P3 (182-183)

The other routine will deal with one digit/byte→ it will concatenate both digit again into one byte just for display

First byte will contain [result+1]:[result]

2nd byte will have [result+3]:[result+2]