INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
ELECTRICAL ENGINEERING DEPARTMENT

Class Test-1

Thursday Feb. 11, 2021	**EE 309: Microprocessors** Spring Semester 2021	Time: 10:40-11:25 Marks: 10

Q–1 **a)** Bit 7 of port 1 in an 8051 is initially at '0'. We want to toggle this bit 32 times to create 16 pulses. We write the following code fragment to do this:

```
L1:  CLR   A
     INC   A
     CPL   P1.7
     CJNE  A, 32, L2
L2:  ...   ...
```

Soln. 1-a) This code needs several corrections:

1. Loop counter should be initialized outside the loop.
2. Immediate numbers should use the hash sign (#).
3. CJNE should jump to L1 and not to L2 in order to form the loop.

The corrected code is:

```
     CLR   A
L1:  CPL   P1.7
     INC   A
     CJNE  A, #32, L1
L2:  ...   ...
```

Code using DJNZ is:

```
     MOV   A, #32
L1:  CPL   P1.7
     DJNZ  A, L1
L2:  ...   ...
```

For loops with a fixed count, DJNZ gives shorter code.

b) In an 8051, the PSW register contains the following bits:

Bit:	7	6	5	4	3	2	1	0
Flag:	Cy	AC	F0	RS1	RS0	OV	F1	P

Assume that initially all these bits are '0' and the following sequence of instructions is executed:

```
MOV   A, #69H
ADD   A, #48H
DA    A
RRC   A
DA    A
```

What will be the values in A and PSW after executing each of the above instructions? (The values should be specified in binary – not as hex or decimal numbers).

Soln. 1-b) The values of A and PSW after each line of the given code is executed are given by:

Line	Instr	Args	A value	PSW value
1	MOV	A, #69H	0110 1001	0000 0000
2	ADD	A, #48H	1011 0001	0100 0100
3	DA	A	0001 0111	1100 0100
4	RRC	A	1000 1011	1100 0100
5	DA	A	1111 0001	1100 0101

Explanation for each instruction:

1. A now contains 69H. MOV instruction does not directly affect any flags. Since the number of 1's in A is 4 (even), parity flag contains '0'. (The parity flag always indicates the status of A).
2. Binary addition 69H + 48H produces B1H in A.
   The auxiliary carry is set because there was a carry from bit 3 to bit 4. There is no carry out from the most significant bit, so carry bit is cleared. Since there was carry out from bit 6 but not from bit 7, the overflow flag will be set.(Overflow flag is set as XOR of carry out of bits 6 and 7). The overflow flag warns that if we consider the sum as a signed number, it will be interpreted as a negative number resulting from the addition of two positive numbers.
   The number of 1's in A is 4 (even), so parity flag is 0.
3. Since the auxiliary carry is set, 06 is added to the lower nibble, producing B7H in A. Since the upper nibble is > 9, 6 is added to it as well, producing 17H and setting the carry flag. Thus, A contains 17H with carry flag set. This is correct for decimal addition of 69 and 48 which should give 117.
   The number of 1's in A is still 4 (even), so the parity flag is 0. DA does not alter any other flags – so the OV flag remains set.
4. This rotates A with carry by one position to the right, producing 8B in A and setting the carry flag. There are four 1's in A, so P=0. Other flags are unchanged.
5. DA should normally follow ADD or ADDC, but it is interesting to see what happens if we use if after RRC.
   Since AC is set, 06 is added to the lower nibble, producing 91 in A. Since Cy is also set, 6 is added to the upper nibble, producing F1 in A. Now there are five 1's in A (odd), so parity flag is set. Other flags remain unchanged.

Q–2 We want to use external interrupt 1 in an 8051 with edge triggering and high priority. A handler for this interrupt will be loaded at the address 0050H and it will use bank 2 of the registers R0 to R7 for its exclusive use.

a) Give all the initialization code required to enable this interrupt and to run this handler when the interrupt occurs. The interrupt vector for external interrupt 1 is at 0013H.

Soln. 2-a) The code should be:

```
0000    ljmp        INIT
        ORG         0013H
0013    ljmp        Handler
        ORG         0050H
0050    Handler: ...
        ORG         0100H
INIT:   MOV         SP, #17H    ;Avoid overlapping stack with banks 0,1 and 2
        SETB        IT1         ;TCON.2: make ext interrupt 1 edge type.
        SETB        PX1         ;IP.2: make it a high priority interrupt
        SETB        EX1         ;IE.2: Enable external interrupt 1
        SETB        IE          ;Global interrupt enable
        ...
```

b) The interrupt handler needs to use the accumulator and registers R0 to R7. We intend to reserve bank 0 and bank 1 of Rn registers for the main program and use the registers in bank 2 for the interrupt handler. Give the assembly code to be included at the start and at the end of the handler routine for saving and restoring registers and for bank switching.

Soln. 2-b) The handler code should include the following at the beginning and at the end:

```
Handler:  PUSH   ACC     ;Save the A register
          PUSH   PSW     ;Save flags and bank select bits
          SETB   PSW.4   ;Select Bank 2 for registers
          CLR    PSW.3   ;
          ...            ;code using A and R registers
          ...
Done:     POP    PSW
          POP    ACC
          RETI
```

c) Give the complete sequence of events which will take place when this interrupt occurs and when its handler terminates.

Soln. 2-c) When the interrupt occurs through a negative edge on line P3.3

1. The current instruction is completed.
2. All other interrupts are disabled since this is a high priority interrupt.
3. The value of updated PC is put on the stack, lower byte first.
4. The vector 0013H is loaded into PC. So the next instruction fetched and executed is from this address – which is: ljmp Handler. Thus the address of the handler routine (0050H) is loaded into PC.

5. Instructions from this address are now executed, beginning with PUSH ACC and PUSH PSW and setting the bank select bits to select bank 2 for R0-R7.
6. Now the handler code is executed and at the end, PSW and ACC are popped, which restores the flags, register bank selection and the value in ACC.
7. The program now returns through RETI, which restores interrupt enable bits for other interrupts and pops the return address from the stack which is put into PC.
8. The program resumes from the instruction next to the one where interrupt had occurred.