Indian Institute of Engineering Science and Technology, Shibpur

B. Tech (CS) 5th Semester Examinations, January 2021

Microprocessor based design: CS 503

[Answer should be Machine/Handprinted on the space provided. Drawing, if required, should be done separately and imported on the space provided in this question paper. Also, import your signature at the end of the paper. Otherwise, you may write on separate page(s), sign and send the PDF]

Full Marks: 70 Time: 90 Minutes

Answer the following questions in short

1. a) Common RISC characteristics are [6]

Sl. No.	Description
1.	Less number of instructions and addressing modes
2.	Fixed length instruction format leading to easier decoding and ease of pipelining operations
3.	Hardwired control unit instead of microprogramming
4.	Load store architecture
5.	More on-chip register leading to register based parameter passing in certain cases
6.	More reliance on code optimization by the compiler

b) Using Berkeley RISC-I instruction load 15H to Register R5 and copy Register R5 to R10 [4]

As R0 is hardwired to contain always the value 0 – add instruction may be used to accomplish the load/move operation.

```
ADD     R0, #15H, R5
ADD     R0, R5, R10
```

2. a) *int addArrayElements(int *a, int n)* is a C function. Assume that the function is converted to X86 assembly and for parameter passing a stack frame is used. The function for some reason pushes three registers (BX, CX and DX) on to the stack and creates space for three 16-bit word local variables. And then gets 1st and 2nd operands from the stack in BX and CX registers. What are the standard instructions used to accomplish moving the 1st and 2nd operands? [5]

The parameters are picked-up from the stack using BP and offset. Now as a standard practice for stack-based parameter passing in X86 – push BP and then copying SP to BP is always done. It may also be noted that for a far call after the parameters CS and IP were pushed. And BP is pushed next. So, starting from the current position of BP the last parameter (here it is the second) is available at BP+6 and the first is at BP+8 – and so on for more parameters (if any)

So, the instructions are i) MOV CX, [BP+6] and

ii) MOV BX, [BP+8] assuming that the count is loaded in CX and address in BX.

b) Suppose an array of N integers (each 16-bit) is stored starting at an odd address. Processing this array in 8086 based PC could take more time compared to a similar array that starts at even address – Justify. [5]

X86 allows word (16-bit) variables memory to start from the odd address (No consideration for alignment). So, each word stored in the memory is not aligned; i.e., the lower byte of the word data is in odd address (X86 follows little endian system) and the upper byte of the same data item is stored on the even address of next higher address. Now as per the hardware connection with the single read you get b15 to b0 of a single word if it starts with the even address. Thus, this misalignment would cause two (word) read from two successive memory locations – discarding two extra bytes that have been read—join the remaining two bytes to form a single word to be processed. So, you need double time just to get the data for processing.

3. a) For the following program segments (X86 assembly language) [6]

   i) PUSH BX ii) MOV AX, [BX+10]

   determine the physical address (in HEX) to be generated by the 8086 processor to accomplish the task described in the instructions. Assume that the initial values as shown below.

   CS: 1234H; DS: 5678H; SS: 9ABCH; BX: 0BF9EH; SP: 10F2H

i) SS * 16 + SP = 9ABC0 + 10F2 = CBCB2
ii) DS * 16 + BX + 10 = 56780 + 0BF9 + 0010 =567389

b) Determine the address of the ISR (X86 assembly) for the instruction

INT 81H -- assume that the upper half of the first 1K address space is loaded entirely with the same byte 55H. [4]

ISR address is picked up from starting location 81H * 4 = 10 0000 0100 (in binary) = 204H – is an address on the upper half of the 1st 1 K. So, the ISR address would be 55550 + 5555 = 5AAA5H as all the locations are loaded with 55H. So, CS = 5555H and IP = 5555H and the physical address generated for ISR is CS * 16 + IP. Note that the ISR is stored in 4-consecutive bytes 2 for IP and 2 – for CS in the vector table.

4. Using a MCS 48 microcontroller suppose that you need to generate 8-square waves through its built in Port 1. The frequency of the square waves using bit 0 is 128, bit 1 produces a frequency of 64 … in this order; so, bit 7 is producing a frequency of 1 unit. Suppose a count of 255 in a register when decremented to 0 produces a delay of 1 unit. Write the routine. [12]

```
L0:     MOV   R0, #0 ;

        MOV   R1, #128

L1:     MOVP P0, R0

        CALL  DELAY1UNIT

        INC   R0

        DJNZ  R1, L1
```

```
                JMP     L0
DELA1UNIT:      MOV     R2, #255
    D1:         DJNZ    R2, D1
                RET
```

Note that bit 0 is changing with each new count. Bit 1 changes at a frequency of 1/2 as that of bit 0. Bit 2 changes at a frequency of 1/2 as that of bit 1 … and so on.

5. Using a suitable example show the use of ARM's conditional arithmetic operation to reduce the code size.

[8]

Conditional arithmetic operation is helpful in reducing the code as seen from the example given below to find out the GCD of two integers 'a' and 'b'. We write the C -code followed by ARM assemble i) without using conditional execution and ii) with conditional execution.

```
---------------------------------------------------------------
While ( a != b)
  if (a >b)     a = a - b;

  else          b = b - a;

---------------------------------------------
gcd     cmp     r1, r2                  ; ARM 1st version
        beq             done
        blt             lessthan
        sub             r1, r1, r2
        b               gcd
Lessthan
        sub             r2, r2, r1
        b               gcd
done
-----------------------------------
gcd             cmp     r1, r2          ; ARM 2nd version with conditional execution
                subgt   r1, r1, r2      ; subtract if r1 > r2 (r1 ← r1 – r2)
                sublt   r2, r2, r1      ; subtract if r2 > r1 (r2 ← r2 –r1)
                bne     gcd
done
```

6. Write subroutines in assembly language of 8085. [5+5+10]

a) Swaps the upper and lower nibbles of Accumulator.

b) ; swapNIBBLES – this subroutine Swaps the upper and lower nibbles of Accumulator.

```
swapNIBBLES: push   b
             mvi    c, 4
        l1:  rlc           ; rotate left (rotation – b7 goes to b0; b0 to b1)
             dcr    c
             jnz    l1
             pop    b
             ret
```

c) Converts a Binary number in Accumulator to equivalent Gray value.

```
Bin2GRAY:   ; this routine converts a Binary no. to GRAY

        PUSH  B

        MOV   B, A    ; KEEP A COPY

        STC

        CMC           ; SET c= 0

        RAR   ;       A = 0 B7 B6 B5 B4 B3 B2 B1

        XRA   B       ; GRAY IN ACC

        POP   B

        RET
```

d) Assuming that a 16-bit address is stored (in ASCII-hex) starting from location ADDBUFF (most significant) to ADDBUF+3 (least significant). Write a routine that transfers execution control to the code specified by this address.

```
gotoADDRESS:   /* this routine gets the address of the code wherefrom execution starts */
    LXI    H, ADDBUF

    CALL  GETBYTE     ; USE THE STRING BUFFER TO GET THE 1ST BYTE

                      ; i.e. UPPER BYTE OF THE ADDRESS

    MOV   D, A
    INX    H

    CALL   GETBYTE    ; USE THE STRING BUFFER TO GET THE 1ST BYTE

                      ; i.e. LOWER BYTE OF THE ADDRESS
```

```
          MOV   E, A

          XCHG
          PCHL

     GETBYTE:
          MOV   A, M
          CALL  ASCII2HEX
          MOV   B, A
          INX   H
          MOV   A, M
          CALL  ASCII2HEX
          MOV   C, A
          CALL  PACK
          RET


:ASCII2HEX:

          CPI   A, 10H
          JC    ZERO2NINE
          SUI    ‘A’ + 10H
          JMP   DONE

   ZERO2ONE:

          SUI   ‘0’

DONE:     RET


PACK:    MOV   A, B

          RLC

          RLC

          RLC

          RLC

          ORA   C

          RET
```