MARIE (chapter 4) sample problems

Provide the full RTN for a new MARIE instruction, Inc X. This instruction fetches datum X from memory, increments it and stores it back to memory.

Fetch:	MAR ← PC	
	MBR ← M[MAR]	// divided into two steps
	IR ← MBR	
	PC ← PC + 1	
Decode:	Decode IR[15..12]	// reversed the order of these two steps
	MAR ← IR[11..0]	
Get op:	MBR ← M[MAR]	
Execute:	AC ← MBR + 1	// increment X
	MBR ← AC	// move X+1 back to MBR
	M[MAR] ← MBR	// store back to memory location X

Provide the full RTN for a new MARIE instruction, JZ X. This instruction sets the AC to see if it is currently 0 and if so, branches to memory location X.

Fetch:	MAR ← PC	
	MBR ← M[MAR]	
	IR ← MBR	
	PC ← PC + 1	
Decode:	Decode IR[15..12]	
	MAR ← IR[11..0]	// this step is not needed
Get op:	not needed	
Execute:	If AC = 0 then PC ← IR[11..0] // if we do the above step of	
	// MAR ← IR[11..0], we could do PC ← MAR here	

We decide to expand MARIE to have 4 registers R0, R1, R2, R3, instead of using the AC. Our instructions now must include the register being referenced as in Load R1, X or Add R3, Y. What impact will this change have on all of MARIE? Be as complete as possible (note: there are two ways you could go with this answer).

Both answers are based on the fact that to denote the register, we need 2 bits.

Answer 1: We add 2 bits to our instruction so that all instructions are now 18 bits long. This will require that the word size expand from 16 bits to 18 bits so that memory is now 4Kx18 instead of 4Kx16. It will require an 18 bit data bus and 18 bit sized registers for the MBR, IR and the 4 data registers. It will also require expanding the ALU to operate on 18 bits.

Answer 2: We remove 2 bits from the operand specification meaning that instead of 12 bits for a memory address, we only use 10. This causes our address space to shrink from

4K to 1K. We would also reduce the address bus from 12 bits to 10 and the MAR and PC from 12 bits to 10.

Provide the MARIE code for the corresponding pseudocode below (written in a Java-like format except for input and output).

```
input(x, y, z);
if(x>y&&y>z)
        output(x);
else if(y!=z) output(y);
else output(0);
```

```
        Input
        Store X
        Input
        Store Y
        Input
        Store Z
        Load X
        Subt Y
        Skipcond 10
        Jump elseif
        Load Y
        Subt Z
        Skipcond 10
        Jump elseif
        Load X
        Output
        Jump done
elseif: Load Y
        Subt Z
        Skipcond 01     // true if y==z
        Jump outy       // we reach here if the above condition is false, thus we want to output y
        Load #0
        Output
        Jump done
outy:   Load Y
        Output
done:   …
```

Write MARIE code to compute z = x / y. Assume its ok to destroy the contents of x in doing so.

```
        Clear
        Store z
```

```
top:    Load x
        Subt y
        Skipcond 10   // having subtracted y from x, is x still positive?
        Jump done
        Store x       // if so, we continue
        Load z
        Add #1
        Store z
        Jump top
done:   …             // note:  z is the quotient and x is the remainder
```

Write the following pseudocode into MARIE code:

```
        Input x
        While(x>=y)
                If(x%2==0) y++;
                Input x
        Output y
```

NOTE: assume we have a subroutine called mod2 which computes x%2 and sets the variable temp to 1 if the result is equal to 0.

```
        Input
        Store x
top:    Load x
        Subt y
        Skipcond 00   // we do not have a test for >=, so I'm using <
        Jump body     // if x >= y, we go here, so we jump to the load body
        Jump done     // otherwise we skip to here and we want to skip the loop body
        Clear
        Store temp
        Jns mod2
        Load temp
        Subt #1       // does temp == 1?
        Skipcond 01
        Jump bottom   // since temp != 1, x%2 != 0, go to the bottom of the loop body
        Load y
        Add #1
        Store y
bottom: Input
        Store x
        Jump top
done:   Load y
        Output
```

Hand-compile into hexadecimal the program in example 4.3 on page 253. NOTE: Skipcond 400 should be Skipcond 01.

100	Load X	110C
101	Subt Y	410D
102	Skipcond 01	8400
103	Jump Else	9108
104	Load X	110C
105	Add X	310C
106	Store X	210C
107	Jump Endif	910B
108	Load Y	110D
109	Subt X	410C
10A	Store Y	210D
10B	Halt	7000