

The Essentials of Computer Organization and Architecture 4th Edition

Linda Null and Julia Lobur
Jones and Bartlett Publishers © 2015

Chapter 4 Instructor's Manual

Chapter Objectives

Chapter 4, MARIE: An Introduction to a Simple Computer, illustrates basic computer organization and introduces many fundamental concepts, including the fetch-decode-execute cycle, the data path, clocks and buses, register transfer notation, and of course, the CPU. A very simple architecture, MARIE, and its ISA are presented to allow the reader to gain a full understanding of the basic architectural organization involved in program execution. MARIE exhibits the classical von Neumann design, and includes a program counter, an accumulator, an instruction register, 4096 bytes of memory, and two addressing modes. Assembly language programming is introduced to reinforce the concepts of instruction format, instruction mode, data format, and control that are presented earlier. This is not an assembly language textbook and was not designed to provide a practical course in assembly language programming. The primary objective in introducing assembly is to further the understanding of computer architecture in general. However, a simulator for MARIE is provided so assembly language programs can be written, assembled, and run on the MARIE architecture. The two methods of control, hardwiring and microprogramming, are introduced and compared in this chapter. Finally, Intel and MIPS architectures are compared to reinforce the concepts in the chapter.

This chapter should be covered before Chapters 5.

Lectures should focus on the following points:

- **CPU basics and organization.** To understand how computers work, one must become familiar with the components and how they are organized. The CPU is a good component to start with as its operation is very easy to understand.
- **Datapath.** The datapath is a network of storage units and arithmetic and logic units connected by buses. It is very important to understand the flow of information in a computer system, and studying the datapath will help to understand this flow.

- **Registers.** Registers are used widely in computer systems as places to store a wide variety of data, such as addresses, program counters, or data necessary for program execution.
- **ALU.** The ALU carries out the logic operations (for example, comparisons) and arithmetic operations (such as add or multiply) required by the instructions being executed by the computer system. A simple ALU was introduced in Chapter 3. In Chapter 4, the focus is on integrating the ALU into the entire system.
- **Control Unit.** The control unit is responsible for extracting instructions from memory, decoding these instructions, making sure data is in the right place at the right time, telling the ALU which registers are to be used, servicing interrupts, and turning on the correct circuitry in the ALU for the execution of the desired operation.
- **Buses.** Buses are the devices that allow various components in the system to communicate. In particular, address buses and data buses are important.
- **Clocks.** Every computer contains an internal clock that regulates how quickly instructions can be executed. The clock also synchronizes all of the components in the system. Clock frequency and clock cycle time determine how quickly a computer can function.
- **Input/Output Subsystem.** Although I/O is covered in depth in Chapter 7, the basic operation of I/O subsystems is introduced and tied in with the rest of the computer system.
- **Memory organization and addressing.** Understanding how a computer function requires not only an understanding of how memory is built, but also how it is laid out and addressed.
- **Interrupts.** Interrupts are events that alter the normal flow of execution in a computer system. They are used for I/O, error handling, and other miscellaneous events, and the system cannot function properly without them.
- **The MARIE architecture.** This is a simple architecture consisting of a very small memory and a simplified CPU. This architecture ties together the concepts from Chapters 2 and 3, and applies this knowledge. It allows coverage of an architecture in depth without the often messy details of a real architecture.
- **Instruction processing.** The fetch-decode-execute cycle represents the steps that a computer follows to run a program. By this point, the ideas of how a system can be built and the necessary components to build it have been covered. Discussing instruction processing allows a deeper understanding of how the system actually works.
- **Register transfer notation.** This symbolic notation can be used to describe how instructions execute at a very low level.
- **Assemblers.** An assembler's job is to convert assembly language (using mnemonics) into machine language (which consists entirely of binary values, or strings of zeros and ones). Assemblers take a programmer's assembly language program, which is really a symbolic representation of the binary numbers, and convert it into binary instructions, or the machine code equivalent. MARIE's assembly language, combined with the MARIE simulator, allow programs to be written and run on the MARIE architecture.
- **Hardwired control versus microprogramming.** Control signals assert lines on various digital components allowing the CPU to execute a sequence of steps correctly. This control can be hardwired and built from digital components, or can use a software program (microprogram). Focus should be on the differences between these two methods.

- **Case studies of real architectures.** Case studies of the Intel and MIPS architectures, with a discussion on those concepts relevant to Chapters 2, 3, and 4, helps reinforce why it is important to understand these ideas. Focus is on register sets, CPU speed, and instruction set architectures. Although MARIE is a very simple architecture, these case studies help confirm that MARIE's design is quite similar to real-world architectures in many aspects.

Required Lecture Time

The important concepts in Chapter 4 can typically be covered in 6 lecture hours. However, if a teacher wants the students to have a mastery of all topics in Chapter 4, 8 lecture hours are more reasonable. If lecture time is limited, we suggest that the focus be on ISAs and instruction processing, as well as writing programs in MARIE's assembly language.

Lecture Tips

The material in this chapter is not intended to be a thorough coverage of assembly language programming. Our intent is to provide a simple architecture with a simple language so students understand the basics of how the architecture and the language are connected.

Regarding potential problem areas for students, there are several. First, the I/O subsystem tends to be unfamiliar territory for most students, so we suggest that instructors spend enough time on this topic to be sure students understand I/O interrupts and the process of I/O itself. Students also seem to have problems with the concepts of byte-addressable and word-addressable. Many mistakenly believe a word to be 32 bits. It is important to stress that the word length is whatever the architecture specifies, and that many machines have words of more than 8 bits, but are still byte-addressable machines. Students also have problems differentiating an instruction in memory from data in memory (e.g., an instruction may have a 12-bit word address so they believe all words in memory must be 12 bits in length when this actually implies this architecture can only address 2^{12} words regardless of their length).

An organization and architecture class is typically the first place students encounter assembly language. It is often difficult for them to understand the "simplicity" of assembly language programming and to recognize that many of the nice features (looping, IF statements, etc.) of higher-level languages often don't exist. Instructors need to emphasize that programming in assembly language (whether it be MARIE's or any other assembly language) requires significantly more intimate knowledge about the architecture and the datapath. In addition, instructors should mention that, although students probably won't be doing much assembly language programming, understanding how to program in assembly will make them better higher-level programmers.

A note about RTN: In the program trace in Figure 4.14, the changes to the registers are shown during the steps of the fetch-decode-execute cycle. Note that for `Load 104`, the steps are:

Load 104

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		100	-----	-----	-----	-----
Fetch	MAR \leftarrow PC	100	-----	100	-----	-----
	IR \leftarrow M[MAR]	100	1104	100	-----	-----
	PC \leftarrow PC + 1	101	1104	100	-----	-----
Decode	MAR \leftarrow IR[11-0]	101	1104	104	-----	-----
	(decode IR[15-12])	101	1104	104	-----	-----
Get operand	MBR \leftarrow M[MAR]	101	1104	104	0023	-----
Execute	AC \leftarrow MBR	101	1104	104	0023	0023

However, when listing the actual RTN for Load, we provide the following:

Load X	MAR \leftarrow X MBR \leftarrow M[MAR], AC \leftarrow MBR
--------	--

The RTN is a clarified version of what is going on in the registers. For example, MAR \leftarrow X is really MAR \leftarrow IR[11-0]. However, we prefer to use X instead of IR[11-0] to give an overall view of what is happening. Instructors should point this out to students.

In addition, for the JnS instruction, we have indicated the following steps:

JnS X	MBR \leftarrow PC MAR \leftarrow X M[MAR] \leftarrow MBR MBR \leftarrow X AC \leftarrow 1 AC \leftarrow AC + MBR PC \leftarrow AC
-------	---

The MAR \leftarrow X (or MAR \leftarrow IR[11-0]) is actually not necessary as the MAR should contain the value of X from the instruction fetch. However, we have included it to remind readers of how this instruction actually works.

A note on the SkipCond instruction: To be consistent with the hexadecimal representation of the instructions, we use Skipcond 400 (which, in hex, is 8400, or 1000 0100 0000 0000), Skipcond 000 (1000 0000 0000 0000), and Skipcond 800 (1000 1000 0000 0000).

Students typically have problems writing the programs, so we encourage instructors to assign multiple programming assignments using MarieSim.

Answers to Exercises

1. What are the main functions of the CPU?

Ans.

The CPU is responsible for fetching program instructions, decoding each instruction that is fetched and performing the indicated sequence of operations on the correct data.

2. How is the ALU related to the CPU? What are its main functions?

Ans.

The ALU is part of the CPU. It carries out arithmetic operations (typically only integer arithmetic) and can carry out logical operations such as AND, OR, and XOR, as well as shift operations.

3. Explain what the CPU should do when an interrupt occurs. Include in your answer the method the CPU uses to detect an interrupt, how it is handled, and what happens when the interrupt has been serviced.

Ans.

The CPU checks, at the beginning of the fetch-decode-execute cycle to see if an interrupt is pending. (This is often done via a special status or flag register.) If so, an interrupt handling routine is dispatched, which itself follows the fetch-decode-execute cycle to process the handler's instructions. When the routine is finished, normal execution of the program continues.

◆ 4. How many bits would you need to address a $2\text{M} \times 32$ memory if

- a) The memory is byte-addressable?
- b) The memory is word-addressable?

Ans.

- a) There are $2\text{M} \times 4$ bytes which equals $2 \times 2^{20} \times 2^2 = 2^{23}$ total bytes, so 23 bits are needed for an address.
 - b) There are 2M words which equals $2 \times 2^{20} = 2^{21}$, so 21 bits are required for an address.
-

5. How many bits are required to address a $4\text{M} \times 16$ main memory if

- a) Main memory is byte-addressable?
- b) Main memory is word-addressable?

Ans.

- a) There are $4M \times 2$ bytes which equals $2^2 \times 2^{20} \times 2 = 2^{23}$ total bytes, so 23 bits are needed for an address
 - b) There are $4M$ words which equals $2^2 \times 2^{20} = 2^{22}$, so 22 bits are required for an address
-

- 6) How many bits are required to address a $1M \times 8$ main memory if
- a) Main memory is byte-addressable?
 - b) Main memory is word-addressable?

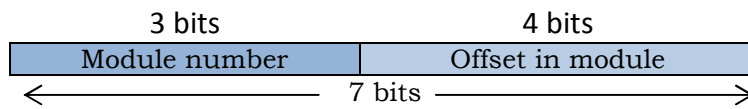
Ans)

- a) There are $1M \times 1$ bytes, or 2^{20} total bytes, so 20 bits are needed for an address
 - b) There are $1M$ words, or 2^{20} total words, so 20 bits are required for an address
-

7. Redo Example 4.1 using high-order interleaving instead of low-order interleaving.

Ans.

Because we have 128 words, we need 7 bits for each address ($128=2^7$). Therefore, an address in this memory has the following structure:



8. Suppose we have 4 memory modules instead of 8 in Figures 4.6 and 4.7. Draw the memory modules with the addresses they contain using: a) High-order Interleaving and b) Low-order interleaving.

Ans.

- a) High-order interleaving

Module 0	Module 1	Module 2	Module 3
0	8	16	24
1	9	17	25
2	10	18	26
3	11	19	27
4	12	20	28
5	13	21	29
6	14	22	30
7	15	23	31

b) Low-order interleaving

Module 0	Module 1	Module 2	Module 3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31

9. How many 256x8 RAM chips are needed to provide a memory capacity of 4096 bytes?

- a) How many bits will each address contain?
- b) How many lines must go to each chip?
- c) How many lines must be decoded for the chip select inputs? Specify the size of the decoder.

Ans.

- a) 12 ($4096 = 2^{12}$)
- b) 8 ($256 = 2^8$)
- c) We have 16 chips so we need 4 lines to select the chip.

◆10. Suppose that a 2M x 16 main memory is built using 256K x 8 RAM chips and memory is word-addressable.

- ◆a) How many RAM chips are necessary?
- ◆b) If we were accessing one full word, how many chips would be involved?
- ◆c) How many address bits are needed for each RAM chip?
- ◆d) How many banks will this memory have?
- ◆e) How many address bits are needed for all of memory?
- ◆f) If high-order interleaving is used, where would address 14 (which is E in hex) be located?
- ◆g) Repeat Exercise 9f for low-order interleaving.

Ans.

- a) 16 (8 rows of 2 columns)
- b) 2
- c) $256K = 2^{18}$, so 18 bits
- d) 8
- e) $2M = 2^{21}$, so 21 bits
- f) Bank 0 (000)
- g) Bank 6 (110) if counting from 0, Bank 7 if counting from 1

11. Redo Exercise 10 assuming a $16\text{M} \times 16$ memory built using $512\text{K} \times 8$ RAM chips.

Ans.

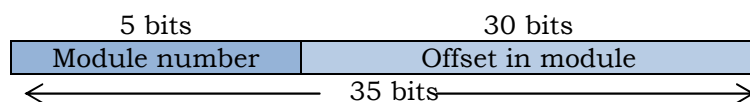
- a) 64 (32 rows of 2 columns)
 - b) 2
 - c) $512\text{K} = 2^{19}$, so 19 bits
 - d) 32
 - e) $16\text{M} = 2^{24}$, so 24 bits
 - f) Bank 0 (000)
 - g) Bank 14 if counting from 0, Bank 15 if counting from 1.
-

12. Suppose we have $1\text{G} \times 16$ RAM chips that make up a $32\text{G} \times 64$ memory that uses high interleaving. (Note: This means that each word is 64 bits in size and there are 32G of these words.)

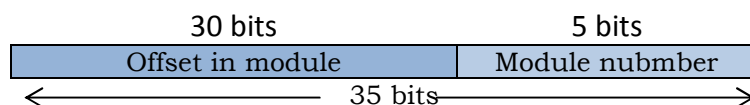
- a) How many RAM chips are necessary?
- b) Assuming 4 chips per bank, how many banks are required?
- c) How many lines must go to each chip?
- d) How many bits are needed for a memory address, assuming it is word addressable?
- e) For the bits in part d), draw a diagram indicating how many and which bits are used for chip select, and how many and which bits are used for the address on the chip.
- f) Redo this problem assuming low order interleaving is being used instead.

Ans.

- a) $32\text{G} \times 64 / 1\text{G} \times 16 = 32 \times 4 = 128$ chips
- b) $128 / 4 = 32$ banks or modules
- c) Each chip has to address 1G of "things" so needs 30 lines
- d) Memory has 32G addresses, or 2^{30} , so needs 30 bits
- e)



- f) Nothing changes except part e:



13. A digital computer has a memory unit with 24 bits per word. The instruction set consists of 150 different operations. All instructions have an operation code part (opcode) and an address part (allowing for only one address). Each instruction is stored in one word of memory.

- a) How many bits are needed for the opcode?
- b) How many bits are left for the address part of the instruction?
- c) What is the maximum allowable size for memory?
- d) What is the largest unsigned binary number that can be accommodated in one word of memory?

Ans.

- a) 8 b) 16 c) 2^{16} d) $2^{24} - 1$
-

14. A digital computer has a memory unit with 32 bits per word. The instruction set consists of 110 different operations. All instructions have an operation code part (opcode) and two address fields: one for a memory address and one for a register address. This particular system includes eight general-purpose, user-addressable registers. Registers may be loaded directly from memory, and memory may be updated directly from the registers. Direct memory-to-memory data movement operations are not supported. Each instruction stored in one word of memory.

- a) How many bits are needed for the opcode?
- b) How many bits are needed to specify the register?
- c) How many bits are left for the memory address part of the instruction?
- d) What is the maximum allowable size for memory?
- e) What is the largest unsigned binary number that can be accommodated in one word of memory?

Ans.

- a) 7 bits for opcode
 - b) 25 bits
 - c) 3 bits to address a register $\Rightarrow 32 - 10 = 22$ bits for address \Rightarrow max memory is 2^{22} or 2MB.
 - d) Largest unsigned binary number is $2^{32} - 1$
-

15. Assume a 2^{20} byte memory:

- ◆ a) What are the lowest and highest addresses if memory is byte-addressable?
- ◆ b) What are the lowest and highest addresses if memory is word-addressable, assuming a 16-bit word?
- c) What are the lowest and highest addresses if memory is word-addressable, assuming a 32-bit word?

Ans.

- a) There are 2^{20} bytes, which can all be addressed using addresses 0 through $2^{20}-1$ with 20-bit addresses
 - b) There are only 2^{19} words and addressing each requires using addresses 0 through $2^{19}-1$
 - c) There are only 2^{18} words and addressing each requires using addresses 0 through $2^{18}-1$
-

16. Suppose the RAM for a certain computer has 256M words, where each word is 16 bits long.

- a) What is the capacity of this memory expressed in bytes?
- b) If this RAM is byte-addressable, how many bits must an address contain?
- c) If this RAM is word addressable, how many bits must an address contain?

Ans.

- a) $256M \times 2 = 2^8 \times 2^{20} \times 2 = 2^{29}$
 - b) 29
 - c) 28
-

17. You and a colleague are designing a brand new microprocessor architecture. Your colleague wants the processor to support 509 different instructions. You do not agree, and would like to have many fewer instructions. Outline the argument for a position paper to present to the management team that will make the final decision. Try to anticipate the argument that could be made to support the opposing viewpoint.

Answer guide:

The point of this problem is to encourage the student to think about tradeoffs. Any good argument along this vein will meet this objective. A clear argument could be made around the design of the control unit: It would be larger, costlier, and more complex than necessary. If it is microprogrammed, it would be slower and more complex to program at the microprogramming level. An argument could be made in favor of a large instruction set stating that the programmers who write applications for the ship will need to write fewer lines of code, thus reducing the chance for error.

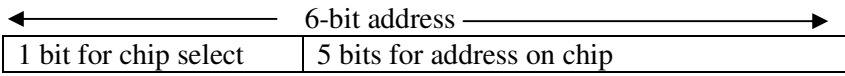
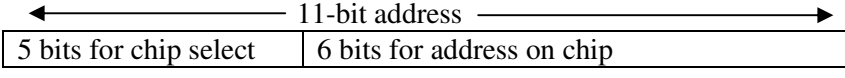
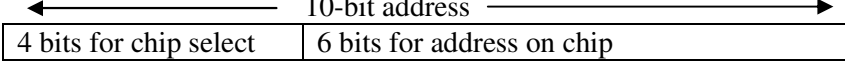
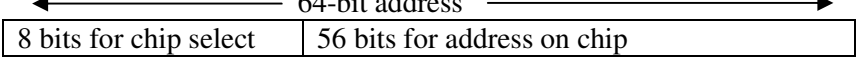
18. Given a memory of 2048 bytes consisting of several 64 Byte x 8 RAM chips. Assuming byte-addressable memory, which of the following seven diagrams indicates the correct way to use the address bits? Explain your answer.

- a)

← 2 bits for chip select	10-bit address → 8 bits for address on chip
--------------------------	--
- b)

← 16 bits for chip select	64-bit address → 48 bits for address on chip
---------------------------	---
- c)

← 6 bits for chip select	11-bit address → 5 bits for address on chip
--------------------------	--

- d) 
- e) 
- f) 
- g) 

Ans.

The correct answer is e.

19. Explain the steps of the fetch-decode-execute cycle. Your explanation should include what is happening in the various registers.

Ans.

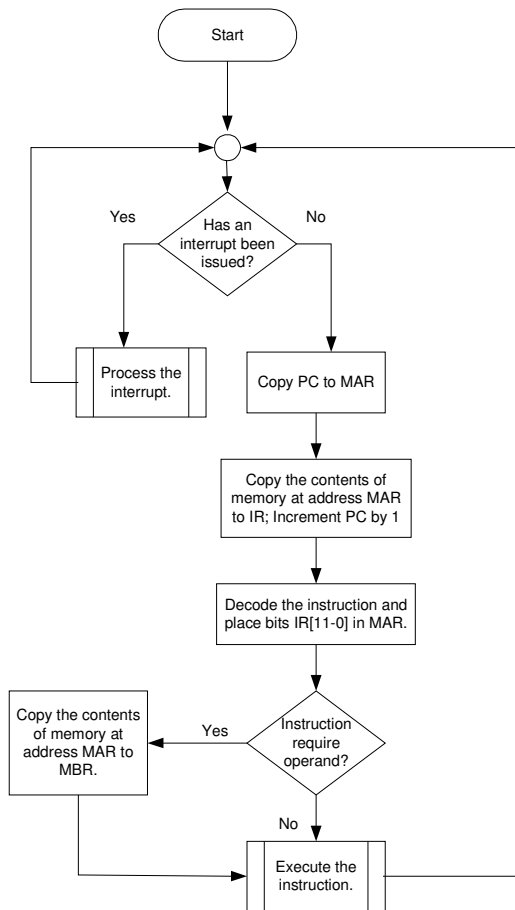
Fetch: Load the PC into the MAR; fetch the instruction and place it into the IR; increment PC by 1;

Decode: Decode the instruction using IR[15-12]; if necessary, place IR[11-0] into MAR and fetch operand, placing result into MBR;

Execute: Execute instruction

20. Combine the flowcharts that appear in Figures 4.11 and 4.12 so that the interrupt checking appears at a suitable place.

Ans.



21. Explain why, in MARIE, the MAR is only 12 bits wide while the AC is 16 bits wide.

Hint: Consider the difference between data and addresses

Ans.

MARIE can handle 16-bit data, so the AC must be 16 bits wide. However, MARIE's memory is limited to 4096 address locations, so the MAR only needs to be 12 bits wide to hold the largest address.

22. List the hexadecimal code for the following program (hand assemble it).

Hex

Address Label Instruction

100		Load A
101		Add One
102		Jump S1
103	S2,	Add One
104		Store A
105		Halt
106	S1,	Add A
107		Jump S2
108	A,	HEX 0023
109	One,	HEX 0001

Ans.

1108
3109
9106
3109
2108
7000
3108
9103
0023
0001

◆23. What are the contents of the symbol table for the preceding program?

Ans.

A	108
One	109
S1	106
S2	103

24. Consider the MARIE program below.

- List the hexadecimal code for each instruction.
- Draw the symbol table.
- What is the value stored in the AC when the program terminates?

Hex Address	Label	Instruction
100	Start,	LOAD A
101		ADD B
102		STORE D
103		CLEAR
104		OUTPUT
105		ADDI D
106		STORE B
107		HALT
108	A,	HEX 00FC

109	B,	DEC 14
10A	C,	HEX 0108
10B	D,	HEX 0000

Ans.

a)

Address	Hex	
100	1108	Start, LOAD A
101	3109	ADD B
102	210B	STORE D
103	A000	CLEAR
104	6000	OUTPUT
105	B10B	ADDI D
106	2109	STORE B
107	7000	HALT
108	00FC	A, HEX 00FC
109	000E	B, DEC 14
10A	0108	C, HEX 0108
10B	0000	D, HEX 0000

b)

SYMBOL TABLE		
Symbol		Location
A		108
B		109
C		10A
D		10B
Start		100

c) AC = 0108 upon termination.

25. Consider the MARIE program below.

- List the hexadecimal code for each instruction.
- Draw the symbol table.
- What is the value stored in the AC when the program terminates?

Hex Address	Label	Instruction
200	Begin,	LOAD Base
201		ADD Offs
202	Loop,	SUBT One
203		STORE Addr
204		SKIPCOND 800
205		JUMP Done
206		JUMPI Addr
207		CLEAR
208	Done,	HALT
209	Base,	HEX 200

20A	Offs,	DEC 9
20B	One,	HEX 0001
20C	Addr,	HEX 0000

Ans.

a)

Address Hex

200	1209		Begin	LOAD Base
201	320A			ADD Offs
202	420B		Loop	SUBT One
203	220C			STORE Addr
204	8800			SKIPCOND 800
205	9208			JUMP Done
206	C20C			JUMPI Addr
207	A000			CLEAR
208	7000		Done	HALT
209	0200		Base	HEX 200
20A	0009		Offs	DEC 9
20B	0001		One	HEX 0001
20C	0000		Addr	HEX 0000

b)

Symbol	Location
Addr	20C
Base	209
Begin	200
Done	208
Loop	202
Offs	20A
One	20B

c) AC = 208 upon termination.

26. Given the instruction set for MARIE in this chapter, decipher the following MARIE machine language instructions. (Write the assembly language equivalent.)

- ◆ a) 0010000000000111
- b) 1001000000001011
- c) 0011000000001001

Ans.

- a) Store 007
- b) Jump 00B
- c) Add 009

27. Write the assembly language equivalent of the following MARIE machine language instructions:

- a) 0111000000000000
- b) 1011001100110000
- c) 0100111101001111

Ans.

- a) Halt
 - b) AddI 330
 - c) Subt F4F
-

28. Write the assembly language equivalent of the following MARIE machine language instructions:

- a) 0000010111000000
- b) 0001101110010010
- c) 1100100101101011

Ans.

- a) JnS 5C0
 - b) Load B92
 - c) JumpI 96B
-

29. Write the following code segment in MARIE's assembly language:

```
if X > 1 then
    Y = X + X;
    X = 0;
endif;
Y = Y + 1;
```

Ans.

```
ORG 100
If,    Load    X        /Load X
       Subt     One      /Subtract 1, store result in AC
       Skipcond 800      /If AC>0 (X>1), skip the next instruction
       Jump     Endif    /Jump to Endif if X is not greater than 1
Then,  Load    X        /Reload X so it can be doubled
       Add      X        /Double X
       Store    Y        /Y= X + X
       Clear    Y        /Move 0 into AC
       Store    X        /Set X to 0
Endif, Load    Y        /Load Y into AC
       Add      One      /Add 1 to Y
       Store    Y        /Y = Y + 1
       Halt     /Terminate program
X,     Dec      ?        /X has starting value, not given in problem
Y,     Dec      ?        /Y has starting value, not given in problem
One,   Dec      1        /Use as a constant
```

30. Write the following code segment in MARIE's assembly language:

```

if x <= y then
    y = y + 1;
else if x != z
    then y = y - 1;
else z = z + 1;

```

Ans.

```

ORG 100
If1,  Load    X        /Load X
      Subt     Y        /Subtract Y, store result in AC
      Skipcond 800      /If AC>0 (X>Y), skip the next instruction
      Jump     Then1    /Jump to Then1 because X<=Y
Else1, Load    X        /End up here if X>Y; need to test if X=Z
      Subt     Z        /Compare X to Z
      Skipcond 400      /If AC=0 (X=Z), skip the next instruction
      Jump     Then2    /Jump to Else2
      Load     Z        /Execute Z=Z+1
      Add      One
      Store    Z
      Jump     Done     /Done with if statement
Then1, Load    Y        /End up here if X<=Y, so execute Y=Y+1
      Add      One
      Store    Y
      Jump     Done     /Done with if statement
Then2, Load    Y
      Subt     One
      Store    Y
Done,  Halt                /terminate program

X,     Dec      ?        /X has starting value, not given in problem
Y,     Dec      ?        /Y has starting value, not given in problem
One,   Dec      1        /Use as a constant

```

31. What are the potential problems (perhaps more than one) with the following assembly language code fragment (implementing a subroutine) written to run on MARIE? The subroutine assumes the parameter to be passed is in the AC and should double this value. The Main part of the program includes a sample call to the subroutine. You can assume this fragment is part of a larger program.

```

Main,  Load    X
      Jump     Sub1
Sret,  Store    X
      ...
Sub1,  Add      X
      Jump     Sret

```

Ans.

First, this subroutine works only for the parameter X (no other variable could be used as X is explicitly added in the subroutine). Second, this subroutine cannot be called from anywhere, as it always returns to Sret.

32. Write a MARIE program to evaluate the expression $A \times B + C \times D$.

Ans.

	ORG	100	
	Load	A	
	Store	X	/Store A in first parameter
	Load	B	
	Store	Y	/Store B in second parameter
	JnS	Mul	/Jump to multiplication subroutine
	Load	Sum	/Get result
	Store	E	/E = A x B
	Load	C	
	Store	X	/Store C in first parameter
	Load	D	
	Store	Y	/Store D in second parameter
	JnS	Mul	/Jump to multiplication subroutine
	Load	Sum	/Get result
	Store	F	/F = C x D
	Load	E	/Get first result
	Add	F	/AC now contains sum of A X B + C X D
	Halt		/Terminate program
A,	Dec	?	/Initial values of A,B,C,D not given in problem
B,	Dec	?	/ (give values before assembling and running)
C,	Dec	?	/
D,	Dec	?	/
X,	Dec	0	/First parameter
Y,	Dec	0	/Second parameter
Ctr,	Dec	0	/Counter for looping
One,	Dec	1	/Constant with value 1
E,	Dec	0	/Temp storage
F,	Dec	0	/Temp storage
Sum,	Dec	0	
Mul,	Hex	0	/Store return address here
	Load	Y	/Load second parameter to be used as counter
	Store	Ctr	/Store as counter
	Clear		/Clear sum
	Store	Sum	/Zero out the sum to begin
Loop,	Load	Sum	/Load the sum
	Add	X	/Add first parameter
	Store	Sum	/Store result in Sum
	Load	Ctr	
	Subt	One	/Decrement counter
	Store	Ctr	/Store counter
	SkipCond	400	/If counter = 0 finish subroutine
	Jump	Loop	/Continue subroutine loop
	JumpI	Mul	/Done with subroutine, return to main
	END		

33. Write the following code segment in MARIE assembly language:

```
X = 1;
while X < 10 do
    X = X + 1;
endwhile;
```

Ans.

```

                ORG 100
                Load    One           /Initialize X
                Store    X
Loop,           Load    X             /Load loop constant
                Subt     Ten           /Compare X to 10
                SkipCond 000          /If AC<0 (X is less than 10), continue loop
                Jump     Endloop       /If X is not less than 10, terminate loop
                Load    X             /Begin body of loop
                Add      One          /Add 1 to X
                Store    X            /Store new value in X
                Jump     Loop         /Continue loop
Endloop,       Halt                    /Terminate program
X,             Dec      0             /Storage for X
One,           Dec      1             /The constant value 1
Ten,           Dec      10           /The loop constant
```

34. Write the following code segment in MARIE assembly language. (Hint: Turn the for loop into a while loop):

```
Sum = 0;
for X = 1 to 10 do
    Sum = Sum + X;
```

Ans.

```

                ORG      100
                Load     One           /Load constant
                Store     X             /Initialize loop control variable X
Loop,           Load     X             /Load X
                Subt      Ten           /Compare X to 10
                SkipCond  000          /If AC<0 (X is less than 10), continue loop
                Jump      Endloop       /If X is not less than 10, terminate loop
                Load     Sum           /Load Sum
                Add       X             /Add X to Sum
                Store     Sum          /Store result in Sum
                Load     X             /Load X
                Add       One          /Increment X
                Store     X
                Jump      Loop
Endloop,       Load     Sum           /Load Sum
                Output    Sum          /Print Sum
                Halt      /terminate program
Sum,           Dec      0             /Storage for Sum
X,             Dec      0             /Storage for X
One,           Dec      1             /The constant value 1
Ten,           Dec      10           /The loop constant
                END
```

35. Write a MARIE program using a loop that multiplies two positive numbers by using repeated addition. For example, to multiple 3 x 6, the program would add 3 six times, or 3+3+3+3+3+3.

Ans.

	ORG	100	
	Load	Y	/Load second value to be used as counter
	Store	Ctr	/Store as counter
Loop,	Load	Sum	/Load the sum
	Add	X	/Add X to Sum
	Store	Sum	/Store result in Sum
	Load	Ctr	
	Subt	One	/Decrement counter
	Store	Ctr	/Store counter
	SkipCond	400	/If AC=0 (Ctr = 0), discontinue looping
	Jump	Loop	/If AC not 0, continue looping
Endloop,	Load	Sum	
	Output		/Print product
	Halt		/Sum contains the product of X and Y
Ctr,	Dec	0	
X,	Dec	?	/Initial value of X (could also be input)
Y,	Dec	?	/Initial value of Y (could also be input)
Sum,	Dec	0	/Initial value of Sum
One,	Dec	1	/The constant value 1
	END		

36. Write a MARIE subroutine to subtract two numbers.

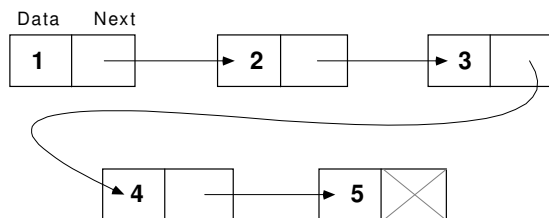
Ans.

Assume the formal parameters are X and Y, and we are subtracting Y from X. Assume also that the actual parameters are A and B. These values could be input or declared in the program. This program tests the subroutine with two sets of values.

	ORG	100	
	Load	A	/Load the first number
	Store	X	/Let X be the first parameter
	Load	B	/Load the second number
	Store	Y	/Let Y be the second parameter
	JnS	Subr	/Store return address, jump to procedure
	Load	X	/Load the result
	Output		/Output the first difference
	Load	C	
	Store	X	
	Load	D	
	Store	Y	
	JnS	Subr	
	Load	X	
	Output		/Output the second difference
	Halt		/Terminate program
X,	Dec	0	/These could also be input
Y,	Dec	0	
A,	Dec	8	/A and B could be input or declared
B,	Dec	4	
C,	Dec	10	

D,	Dec	2	
Subr,	Hex	0	/Store return address here
	Load	X	/Load the first number
	Subt	Y	/Subtract the second number
	Store	X	/Store result in first parameter
	JumpI	Subr	
	END		

37. A linked list is a linear data structure consisting of a set of nodes, where each one except the last one points to the next node in the list. (Appendix A provides more information about linked lists.) Suppose we have the set of 5 nodes shown in the illustration below. These nodes have been scrambled up and placed in a MARIE program as shown below. Write a MARIE program to traverse the list and print the data in order as stored in each node.



MARIE program fragment:

Address			
(Hex)			
00D	Addr,	Hex ????	/ Top of list pointer:
			/ You fill in the address of Node1.
00E	Node2,	Hex 0032	/ Node's data is the character "2."
00F		Hex ????	/ Address of Node3.
010	Node4,	Hex 0034	/ Character "4."
011		Hex ????	
012	Node1,	Hex 0031	/ Character "1"
013		Hex ????	
014	Node3,	Hex 0033	/ Character "3"
015		Hex ????	
016	Node5,	Hex 0035	/ Character "5"
017		Hex 0000	/ Indicates terminal node.

Ans.

/ This program traverses a simple linked list. As each node is visited,
 / its "data" is printed. The last node in the list has the
 / sentinel value 0000 in its next-node pointer.

000	Load	Addr	/ Get the address of the first node.
001	Visit,	Skipcond 400	/ If it is zero, we're done.
002		Jump Getit	
003		Halt	
004	Getit,	LoadI Addr	/ Load the data stored in the node...
005		Output	/ ... and print it.
006		Load Addr	/ Retrieve node's address.
007		Add One	/ Get the pointer to the next node.

```

008 |          Store Addr      / Store the address of the address of next node.
009 |          LoadI Addr     / Get the next node's address...
00A |          Store Addr      / ... and store it.
00B |          Jump Visit
00C | One,      Hex 0001
00D | Addr,     Hex 0012      / Top of list pointer
00E | Node2,    Hex 0032      / Node's data is the character "2."
00F |          Hex 0014      / Address of Node3.
010 | Node4,    Hex 0034      / Character "4."
011 |          Hex 0016
012 | Node1,    Hex 0031      / "1"
013 |          Hex 000e
014 | Node3,    Hex 0033      / "3"
015 |          Hex 0010
016 | Node5,    Hex 0035      / "5"
017 |          Hex 0000
    |          END

```

38. More registers appears to be a good thing, in terms of reducing the total number of memory accesses a program might require. Give an arithmetic example to support this statement. First, determine the number of memory accesses necessary using MARIE and the two registers for holding memory data values (AC and MBR). Then perform the same arithmetic computation for a processor that has more than three registers to hold memory data values.

Ans.

Consider the statement $\text{Sum} = (A + B) - (C + D)$. In MARIE, this would require:

```

Load  A
Add    B
Store Temp1
Load  C
Add    D
Store Temp2
Load  Temp1
Subt   Temp2
Store Sum

```

for a total of 9 memory accesses. (If C+D is executed first, this can be done with 7 memory accesses.)

If an architecture has 4 registers (call them R1, R2, R3 and R4), then we could:

```

Load R1,A
Load R2,B
Add  R1,R2
Load R3,C
Load R4,D
Add  R3,R4    /no memory accesses required for this operation
Subt R1,R4    /no memory accesses required for this operation
Store Sum

```

for a total of 5 memory accesses.

41. Provide a trace (similar to the one in Figure 4.14) for Example 4.3.

Ans. The trace will present the statements in execution order.

```

100 If,      Load    X      /Load the first value
101         Subt     Y      /Subtract the value of Y, store result in AC
102         Skipcond 400    /If AC=0 (X=Y), skip the next instruction
103         Jump     Else    /Jump to Else part if AC is not equal to 0
104 Then,    Load    X      /Reload X so it can be doubled
105         Add      X      /Double X
106         Store    X      /Store the new value
107         Jump     Endif   /Skip over the false, or else, part to end of if
108 Else,    Load    Y      /Start the else part by loading Y
109         Subt     X      /Subtract X from Y
10A        Store    Y      /Store Y-X in Y
10B Endif,   Halt      /Terminate program (it doesn't do much!)
10C X,      Dec      12     /Assume these values for X and Y
10D Y,      Dec      20

```

Load X

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		100	-----	-----	-----	-----
Fetch	MAR ← PC	100	-----	100	-----	-----
	IR ← M[MAR]	100	110C	100	-----	-----
	PC ← PC + 1	101	110C	100	-----	-----
Decode	MAR ← IR[11-0]	101	110C	10C	-----	-----
	(decode IR[15-12])	101	110C	10C	-----	-----
Get operand	MBR ← M[MAR]	101	110C	10C	000C	-----
Execute	AC ← MBR	101	110C	10C	000C	000C

Subt Y

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		101	110C	10C	000C	000C
Fetch	MAR ← PC	101	110C	101	000C	000C
	IR ← M[MAR]	101	410D	101	000C	000C
	PC ← PC + 1	102	410D	101	000C	000C
Decode	MAR ← IR[11-0]	102	410D	10D	000C	000C
	(decode IR[15-12])	102	410D	10D	000C	000C
Get operand	MBR ← M[MAR]	102	410D	10D	0014	000C
Execute	AC ← AC - MBR	102	410D	10D	0014	FFF8

Skipcond 400

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		102	410D	10D	0014	FFF8
Fetch	MAR ← PC	102	410D	102	0014	FFF8
	IR ← M[MAR]	102	8400	102	0014	FFF8
	PC ← PC + 1	103	8400	102	0014	FFF8
Decode	MAR ← IR[11-0]	103	8400	400	0014	FFF8
	(decode IR[15-12])	103	8400	400	0014	FFF8
Get operand	(not necessary)	103	8400	400	0014	FFF8
Execute	do nothing (AC < 0)	103	8400	400	0014	FFF8

Jump Else

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		103	8400	400	0014	FFF8
Fetch	$MAR \leftarrow PC$	103	8400	103	0014	FFF8
	$IR \leftarrow M[MAR]$	103	9108	103	0014	FFF8
	$PC \leftarrow PC + 1$	104	9108	103	0014	FFF8
Decode	$MAR \leftarrow IR[11-0]$	104	9108	108	0014	FFF8
	(decode IR[15-12])	104	9108	108	0014	FFF8
Get operand	(not necessary)	104	9108	108	0014	FFF8
Execute	$PC \leftarrow IR[11-0]$	108	9108	108	000C	FFF8

Load Y

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		108	9108	108	000C	FFF8
Fetch	$MAR \leftarrow PC$	108	9108	108	000C	FFF8
	$IR \leftarrow M[MAR]$	108	110D	108	000C	FFF8
	$PC \leftarrow PC + 1$	109	110D	108	000C	FFF8
Decode	$MAR \leftarrow IR[11-0]$	109	110D	10D	000C	FFF8
	(decode IR[15-12])	109	110D	10D	000C	FFF8
Get operand	$MBR \leftarrow M[MAR]$	109	110D	10D	0014	FFF8
Execute	$AC \leftarrow MBR$	109	110D	10D	0014	0014

Subt X

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		109	110D	10D	0014	0014
Fetch	$MAR \leftarrow PC$	109	110D	109	0014	0014
	$IR \leftarrow M[MAR]$	109	410C	109	0014	0014
	$PC \leftarrow PC + 1$	10A	410C	109	0014	0014
Decode	$MAR \leftarrow IR[11-0]$	10A	410C	10C	0014	0014
	(decode IR[15-12])	10A	410C	10C	0014	0014
Get operand	$MBR \leftarrow M[MAR]$	10A	410C	10C	000C	0014
Execute	$AC \leftarrow AC - MBR$	10A	410C	10C	000C	0008

Store Y

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		10A	410C	10C	000C	0008
Fetch	$MAR \leftarrow PC$	10A	410C	10A	000C	0008
	$IR \leftarrow M[MAR]$	10A	210D	10A	000C	0008
	$PC \leftarrow PC + 1$	10B	210D	10A	000C	0008
Decode	$MAR \leftarrow IR[11-0]$	10B	210D	10D	000C	0008
	(decode IR[15-12])	10B	210D	10D	000C	0008
Get operand	not necessary	10B	210D	10D	000C	0008
Execute	$MBR \leftarrow AC$	10B	210D	10D	0008	0008
(changes Y)	$M[MAR] \leftarrow MBR$	10B	210D	10D	0008	0008

Halt

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		10B	210D	10D	0008	0008
Fetch	$MAR \leftarrow PC$	10B	210D	10B	0008	0008
	$IR \leftarrow M[MAR]$	10B	7000	10B	0008	0008
	$PC \leftarrow PC + 1$	10C	7000	10B	0008	0008
Decode	$MAR \leftarrow IR[11-0]$	10C	7000	000	0008	0008
	(decode IR[15-12])	10C	7000	000	0008	0008
Get operand	not necessary	10C	7000	000	0008	0008
Execute	terminate program	10C	7000	000	0008	0008

42. Provide a trace (similar to the one in Figure 4.14) for Example 4.4.

Ans. The trace will present the statements in execution order.

```

100      Load      X          /Load the first number to be doubled
101      Store     Temp       /Use Temp as a parameter to pass value to Subr
102      JnS       Subr       /Store return address, jump to procedure
103      Store     X          /Store first number, doubled
104      Load     Y          /Load the second number to be doubled
105      Store     Temp       /Use Temp as a parameter to pass value to Subr
106      JnS       Subr       /Store return address, jump to procedure
107      Store     Y          /Store second number, doubled
108      Halt      /End program
109  X,   Dec      20
10A  Y,   Dec      48
10B  Temp, Dec     0
10C  Subr, Hex     0          /Store return address here
10D      Load     Temp       /Actual subroutine to double numbers
10E      Add       Temp       /AC now hold double the value of Temp
10F      JumpI     Subr       /Return to calling code
      END

```

Load X

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		100	-----	-----	-----	-----
Fetch	$MAR \leftarrow PC$	100	-----	100	-----	-----
	$IR \leftarrow M[MAR]$	100	1109	100	-----	-----
	$PC \leftarrow PC + 1$	101	1109	100	-----	-----
Decode	$MAR \leftarrow IR[11-0]$	101	1109	109	-----	-----
	(decode IR[15-12])	101	1109	109	-----	-----
Get operand	$MBR \leftarrow M[MAR]$	101	1109	109	0014	-----
Execute	$AC \leftarrow MBR$	101	1109	109	0014	0014

Store Temp

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		101	1109	109	0014	0014
Fetch	MAR \leftarrow PC	101	1109	101	0014	0014
	IR \leftarrow M[MAR]	101	210B	101	0014	0014
	PC \leftarrow PC + 1	102	210B	101	0014	0014
Decode	MAR \leftarrow IR[11-0]	102	210B	10B	0014	0014
	(decode IR[15-12])	102	210B	10B	0014	0014
Get operand	not necessary	102	210B	10B	0014	0014
Execute	MBR \leftarrow AC	102	210B	10B	0014	0014
(changes Temp)	M[MAR] \leftarrow MBR	102	210B	10B	0014	0014

JnS Subr

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		102	210B	10B	0014	0014
Fetch	MAR \leftarrow PC	102	210B	102	0014	0014
	IR \leftarrow M[MAR]	102	010C	102	0014	0014
	PC \leftarrow PC + 1	103	010C	102	0014	0014
Decode	MAR \leftarrow IR[11-0]	103	010C	10C	0014	0014
	(decode IR[15-12])	103	010C	10C	0014	0014
Get operand	not necessary	103	010C	10C	0014	0014
Execute	MBR \leftarrow PC	103	010C	10C	0103	0014
	MAR \leftarrow IR[11-0]	103	010C	10C	0103	0014
(changes Subr)	M[MAR] \leftarrow MBR	103	010C	10C	0103	0014
	MBR \leftarrow IR[11-0]	103	010C	10C	010C	0014
	AC \leftarrow 1	103	010C	10C	010C	0001
	AC \leftarrow AC + MBR	103	010C	10C	010C	010D
	PC \leftarrow AC	10D	010C	10C	010C	010D

Load Temp

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		10D	A000	000	010C	0000
Fetch	MAR \leftarrow PC	10D	A000	10D	010C	0000
	IR \leftarrow M[MAR]	10D	110B	10D	010C	0000
	PC \leftarrow PC + 1	10E	110B	10D	010C	0000
Decode	MAR \leftarrow IR[11-0]	10E	110B	10B	010C	0000
	(decode IR[15-12])	10E	110B	10B	010C	0000
Get operand	MBR \leftarrow M[MAR]	10E	110B	10B	0014	0000
Execute	AC \leftarrow MBR	10E	110B	10B	0014	0014

Add Temp

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		10E	110B	10B	0014	0014
Fetch	MAR \leftarrow PC	10E	110B	10E	0014	0014
	IR \leftarrow M[MAR]	10E	310B	10E	0014	0014
	PC \leftarrow PC + 1	10F	310B	10E	0014	0014
Decode	MAR \leftarrow IR[11-0]	10F	310B	10B	0014	0014
	(decode IR[15-12])	10F	310B	10B	0014	0014
Get operand	MBR \leftarrow M[MAR]	10F	310B	10B	0014	0014
Execute	AC \leftarrow AC + MBR	10F	310B	10B	0014	0028

JumpI Subr

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		10F	310B	10B	0014	0028
Fetch	MAR \leftarrow PC	10F	310B	10F	0014	0028
	IR \leftarrow M[MAR]	10F	C10C	10F	0014	0028
	PC \leftarrow PC + 1	110	C10C	10F	0014	0028
Decode	MAR \leftarrow IR[11-0]	110	C10C	10C	0014	0028
	(decode IR[15-12])	110	C10C	10C	0014	0028
Get operand	MBR \leftarrow M[MAR]	110	C10C	10C	0103	0028
Execute	PC \leftarrow MBR	103	C10C	10C	0103	0028

Store X

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		103	C10C	10C	0103	0028
Fetch	MAR \leftarrow PC	103	C10C	103	0103	0028
	IR \leftarrow M[MAR]	103	2109	103	0103	0028
	PC \leftarrow PC + 1	104	2109	103	0103	0028
Decode	MAR \leftarrow IR[11-0]	104	2109	109	0103	0028
	(decode IR[15-12])	104	2109	109	0103	0028
Get operand	not necessary	104	2109	109	0103	0028
Execute	MBR \leftarrow AC	104	2109	109	0028	0028
(changes X)	M[MAR] \leftarrow MBR	104	2109	109	0028	0028

Load Y

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		104	2109	109	0028	0028
Fetch	MAR \leftarrow PC	104	2109	104	0028	0028
	IR \leftarrow M[MAR]	104	110A	104	0028	0028
	PC \leftarrow PC + 1	105	110A	104	0028	0028
Decode	MAR \leftarrow IR[11-0]	105	110A	10A	0028	0028
	(decode IR[15-12])	105	110A	10A	0028	0028
Get operand	MBR \leftarrow M[MAR]	105	110A	10A	0030	0028
Execute	AC \leftarrow MBR	105	110A	10A	0030	0030

Store Temp

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		105	110A	10A	0030	0030
Fetch	MAR \leftarrow PC	105	110A	105	0030	0030
	IR \leftarrow M[MAR]	105	210B	10A	0030	0030
	PC \leftarrow PC + 1	106	210B	10A	0030	0030
Decode	MAR \leftarrow IR[11-0]	106	210B	10B	0030	0030
	(decode IR[15-12])	106	210B	10B	0030	0030
Get operand	not necessary	106	210B	10B	0030	0030
Execute	MBR \leftarrow AC	106	210B	10B	0030	0030
(changes Temp)	M[MAR] \leftarrow MBR	106	210B	10B	0030	0030

JnS Subr

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		106	210B	10B	0030	0030
Fetch	MAR \leftarrow PC	106	210B	106	0030	0030
	IR \leftarrow M[MAR]	106	010C	106	0030	0030
	PC \leftarrow PC + 1	107	010C	106	0030	0030
Decode	MAR \leftarrow IR[11-0]	107	010C	10C	0014	0014
	(decode IR[15-12])	107	010C	10C	0014	0014
Get operand	not necessary	107	010C	10C	0014	0014
Execute	MBR \leftarrow PC	107	010C	10C	0107	0014
	MAR \leftarrow IR[11-0]	107	010C	10C	0107	0014
(changes Subr)	M[MAR] \leftarrow MBR	107	010C	10C	0107	0014
	MBR \leftarrow IR[11-0]	107	010C	10C	010C	0014
	AC \leftarrow 1	107	010C	10C	010C	0001
	AC \leftarrow AC + MBR	107	010C	10C	010C	010D
	PC \leftarrow AC	10D	010C	10C	010C	010D

Load Temp

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		10D	A000	000	010C	0000
Fetch	MAR \leftarrow PC	10D	A000	10D	010C	0000
	IR \leftarrow M[MAR]	10D	110B	10D	010C	0000
	PC \leftarrow PC + 1	10E	110B	10D	010C	0000
Decode	MAR \leftarrow IR[11-0]	10E	110B	10B	010C	0000
	(decode IR[15-12])	10E	110B	10B	010C	0000
Get operand	MBR \leftarrow M[MAR]	10E	110B	10B	0030	0000
Execute	AC \leftarrow MBR	10E	110B	10B	0030	0030

Add Temp

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		10E	110B	10B	0030	0030
Fetch	MAR \leftarrow PC	10E	110B	10E	0030	0030
	IR \leftarrow M[MAR]	10E	310B	10F	0030	0030
	PC \leftarrow PC + 1	10F	310B	10F	0030	0030
Decode	MAR \leftarrow IR[11-0]	10F	310B	10B	0030	0030
	(decode IR[15-12])	10F	310B	10B	0030	0030
Get operand	MBR \leftarrow M[MAR]	10F	310B	10B	0030	0030
Execute	AC \leftarrow AC + MBR	10F	310B	10B	0030	0060

JumpI Subr

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		10F	310B	10B	0030	0060
Fetch	MAR \leftarrow PC	10F	310B	10F	0030	0060
	IR \leftarrow M[MAR]	10F	C10C	110F	0030	0060
	PC \leftarrow PC + 1	100	C10C	10F	0030	0060
Decode	MAR \leftarrow IR[11-0]	100	C10C	10C	0030	0060
	(decode IR[15-12])	100	C10C	10C	0030	0060
Get operand	MBR \leftarrow M[MAR]	100	C10C	10C	0107	0060
Execute	PC \leftarrow MBR	107	C10C	10C	0107	0060

Store Y

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		107	C10C	10C	0107	0060
Fetch	$MAR \leftarrow PC$	107	C10C	107	0107	0060
	$IR \leftarrow M[MAR]$	107	210A	107	0107	0060
	$PC \leftarrow PC + 1$	108	210A	107	0107	0060
Decode	$MAR \leftarrow IR[11-0]$	108	210A	10A	0107	0060
	(decode IR[15-12])	108	210A	10A	0107	0060
Get operand	not necessary	108	210A	10A	0107	0060
Execute	$MBR \leftarrow AC$	108	210A	10A	0060	0060
(changes Y)	$M[MAR] \leftarrow MBR$	108	210A	10A	0060	0060

Halt

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		108	210A	10A	0060	0060
Fetch	$MAR \leftarrow PC$	108	210D	108	0060	0060
	$IR \leftarrow M[MAR]$	108	7000	108	0060	0060
	$PC \leftarrow PC + 1$	109	7000	108	0060	0060
Decode	$MAR \leftarrow IR[11-0]$	109	7000	000	0060	0060
	(decode IR[15-12])	109	7000	000	0060	0060
Get operand	not necessary	109	7000	000	0060	0060
Execute	terminate program	109	7000	000	0060	0060

43. Suppose we add the following instruction to MARIE's ISA:

IncSZ Operand

This instruction increments the value with effective address "Operand," and if this newly incremented value is equal to 0, the program counter is incremented by 1. Basically, we are incrementing the operand, and if this new value is equal to 0, we skip the next instruction. Show how this instruction would be executed using RTN.

Ans.

```

MAR ← Operand
MBR ← M[MAR]
AC ← 1
AC ← AC + MBR
M[MAR] ← AC
If AC = 0 then PC ← PC + 1

```

44. Suppose we add the following instruction to MARIE's ISA:

JumpOffset X

This instruction will jump to the address calculated by adding the given address, X, to the contents of the accumulator. Show how this instruction would be executed using RTN.

Ans.

```
MBR ← IR[11..0]
AC ← AC + MBR
PC ← AC
```

45. Suppose we add the following instruction to MARIE's ISA:

JumpOffset X

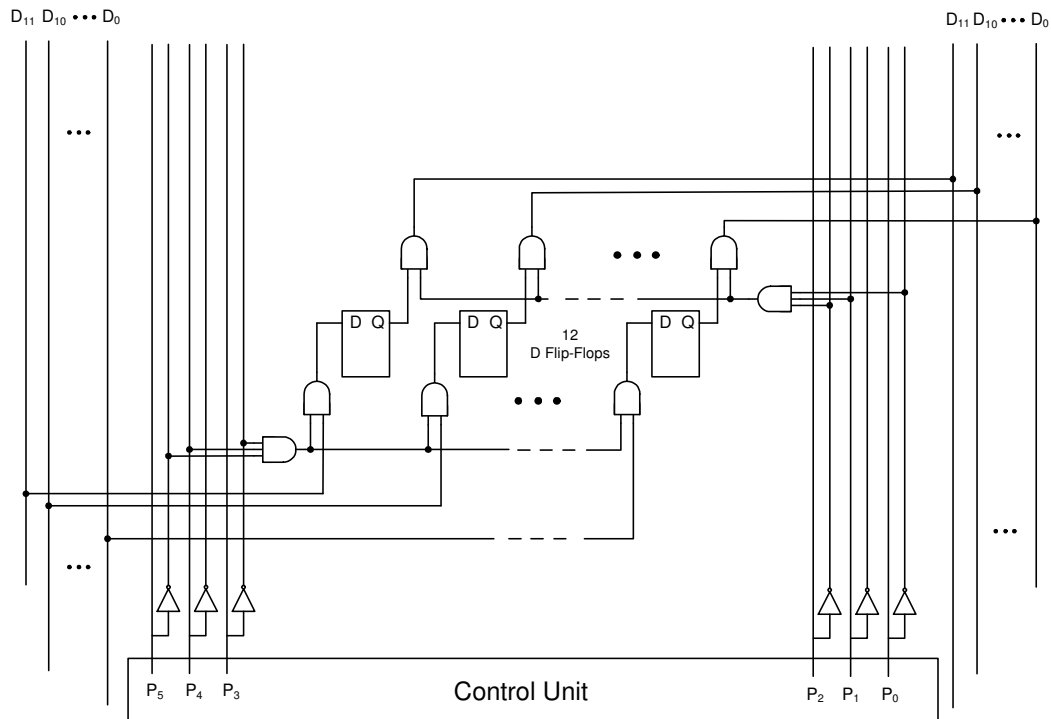
This instruction will jump to the address calculated by going to address X, then adding the value found there to the value in the AC. Show how this instruction would be executed using RTN.

Ans.

```
MAR ← IR[11..00]
MBR ← M[MAR]
AC ← AC + MBR
PC ← AC
```

46. Draw the connection of MARIE's PC to the datapath using the format shown in Figure 4.15.

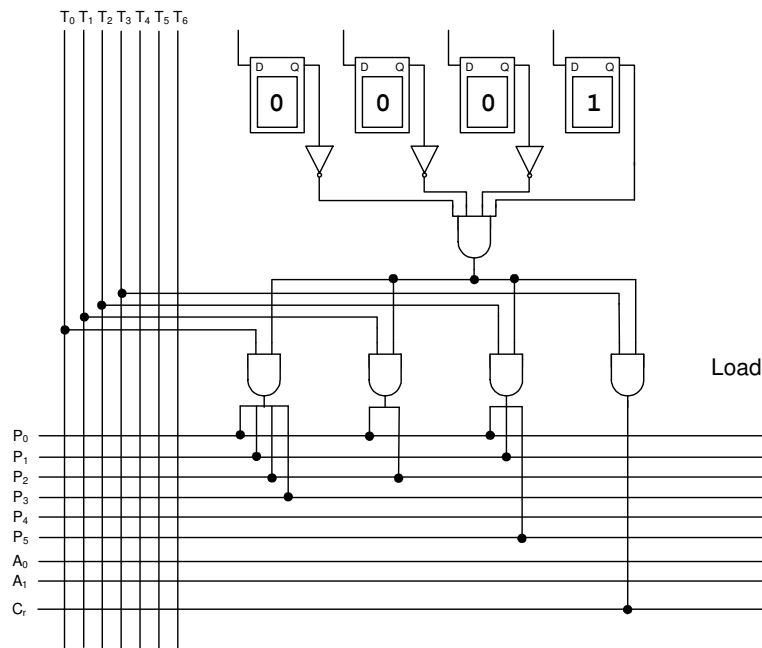
Ans.



47. The table below provides a summary of MARIE's datapath control signals. Using this information, Table 4-9, and Figure 4.20 as guides draw the control logic for MARIE's Load instruction.

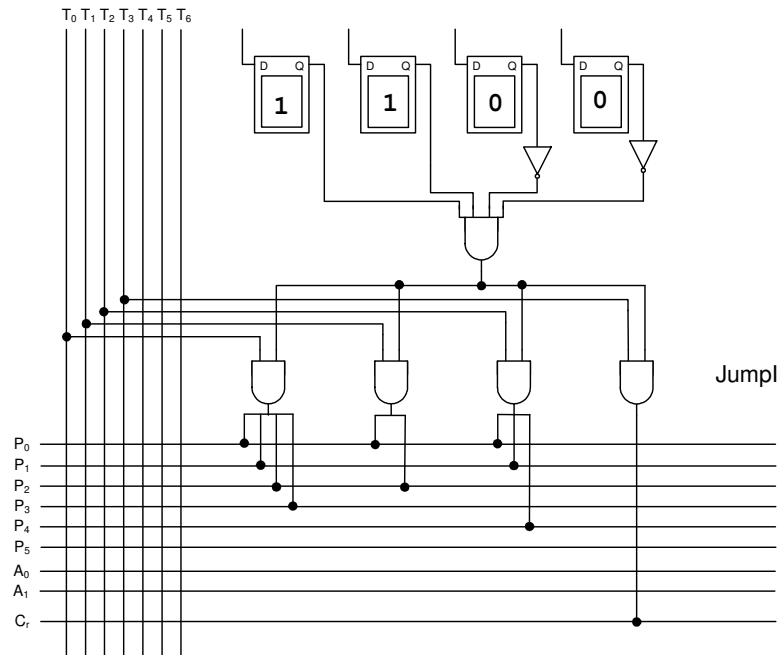
Register	Memory	MAR	PC	MBR	AC	IN	OUT	IR
Signals								
$P_2P_1P_0$ (Read)	000	001	010	011	100	101	110	111
$P_5P_4P_3$ (Write)								

Ans.



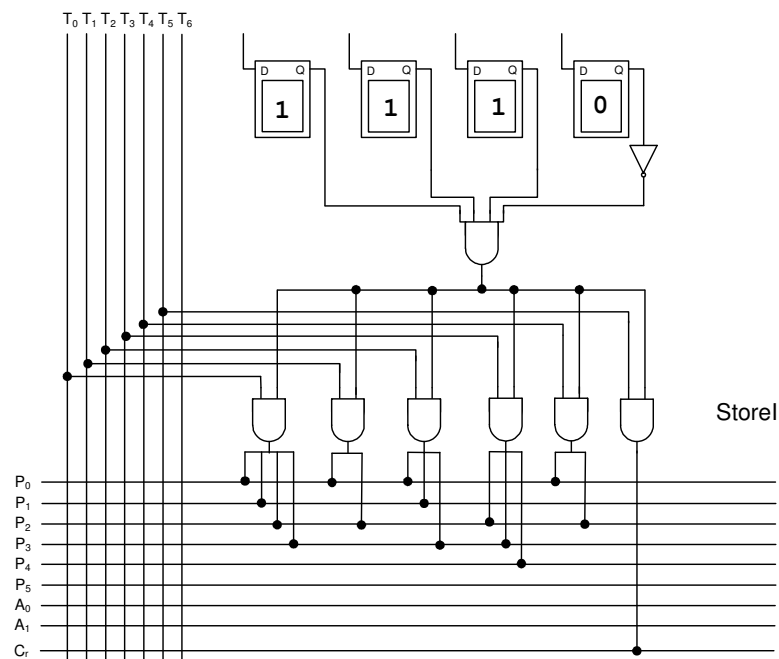
48. The table in Problem 47 provides a summary of MARIE's datapath control signals. Using this information, Table 4.9, and Figure 4.20 as guides draw the control logic for MARIE's JumpI instruction.

Ans.



49. The table in Problem 47 provides a summary of MARIE's datapath control signals. Using this information, Table 4.9, and Figure 4.20 as guides draw the control logic for MARIE's StoreI instruction.

Ans.



50. Suppose some hypothetical system's control unit has a ring (cycle) counter consisting of some number of D flip-flops. This system runs at 1 GHz and has a maximum of 10 microoperations/instruction.

- a) What is the maximum frequency of the output (number of signal pulses) output by each flip-flop?
- b) How long does it take to execute an instruction that requires only 4 microoperations?

Ans.

- a) The output frequency of each flip-flop is $[1 \div (\text{number of flip-flops})] \times \text{clock frequency}$.
Thus: $10^{-1} \times 10^9 = 10^8$ pulses/sec.
 - b) Each microoperation requires 10^{-8} seconds to execute. Hence a 4-microoperation instruction needs 4×10^{-8} seconds to execute, or 40 nanoseconds.
-

51. Suppose you are designing a hardwired control unit for a very small computerized device. This system is so revolutionary that the system designers have devised an entirely new ISA for it. Because everything is so new, you are contemplating including one or two extra flip-flops and signal outputs in the cycle counter. Why would you want to do this? Why would you not want to do this? Discuss the tradeoffs.

Ans.

You would want to include extra flip flops in the cycle counter because the architecture is in such a state of flux. If it turns out that an instruction needs more cycles to execute than originally thought, a great deal of redesign work may have to be done to accommodate the extra required signals. (Often this kind of thing happens toward the end of the project, when everyone is under a great deal of deadline pressure. Hasty work can cause even bigger problems.)

You may decide against the extra flip-flops because you would like to do only as much work as is needed, and you want to keep the cost and size of your control unit to an absolute minimum. Changes to the microoperations of the instructions will probably require changes to the control matrix, anyway. Adding a couple of extra flip-flops at that point, may not be as big a deal as it sounds.

52. Building on the idea presented in Problem 51, suppose that MARIE has a hardwired control unit and we decide to add a new instruction that requires 8 clock cycles to execute. (This is one cycle longer than the longest instruction, **Jns**.) Briefly discuss the changes that we would need to make to accommodate this new instruction.

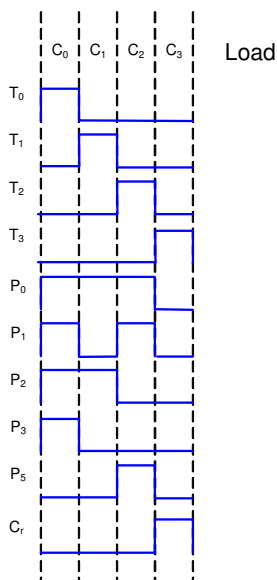
Ans.

There are three essential changes that we need to make to the control unit.

1. First, with the new instruction requiring more clock cycles than the cycle counter can count, we need to add another flip-flop and signal line. Fortunately, it's not necessary to connect this signal line to any of the existing combinational circuits in the control matrix-- only to the new instruction that needs it.
2. Second, we would certainly need to add the logic to the control unit for the new instruction.
3. Third, the instruction decoder would need new combinational logic along with a sixteenth signal output for the new instruction.

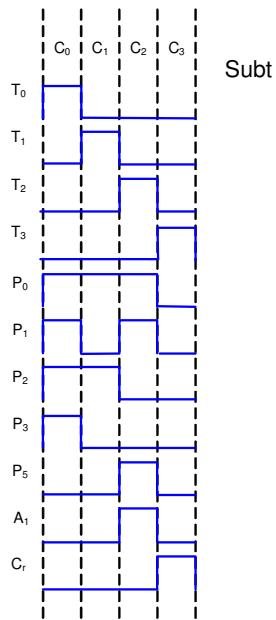
53. Draw the timing diagram for MARIE's Load instruction using the format of Figure 4.16.

Ans.



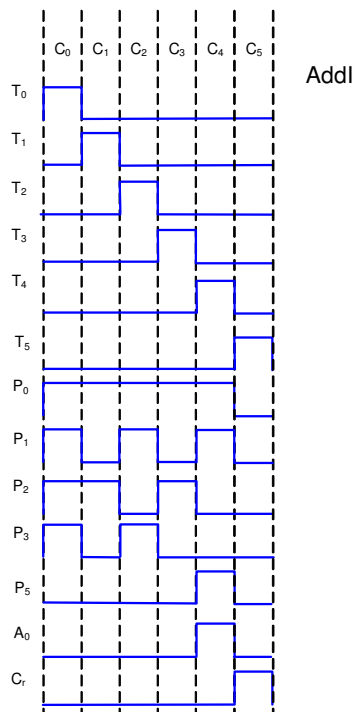
54. Draw the timing diagram for MARIE's Subt instruction using the format of Figure 4.16.

Ans.



55. Draw the timing diagram for MARIE's Addl instruction using the format of Figure 4.16.

Ans.



56. Using the coding given in Table 4.9, translate into binary the mnemonic microcode instructions given in Figure 4.23 for the fetch-decode cycle (the first nine lines of the table).

Ans.

Address	MicroOp 1	MicroOp 2	Jump	Dest
0000000	01010	00000	0	0000000
0000001	00110	00000	0	0000000
0000010	10001	00000	0	0000000
0000011	01111	00000	0	0000000
0000100	11000	00000	1	0100000
0000101	11000	00010	1	0100111
0000110	11000	00100	1	0101010
0000111	11000	00110	1	0101100
0001000	11000	01000	1	0101111

57. Continuing from Exercise 56, write the microcode for the jump table for the MARIE instructions for Jump X, Clear, and AddI X. (Use all 1s for the Destination value.)

Ans.

Address	MicroOp 1	MicroOp 2	Jump	Dest
0001000	11000	01001	1	1111111
0001001	11000	01010	1	1111111
0001010	11000	01011	1	1111111

58. Using Figure 4.23 as a guide, write the binary microcode for MARIE's Load instruction. Assume that the microcode begins at instruction line number **0110000₂**.

Ans.

Address	MicroOp 1	MicroOp 2	Jump	Dest
0110000	01011	00000	0	0000000
0110001	01101	00000	0	0000000
0110010	00010	00000	1	0000000

59. Using Figure 4.23 as a guide, write the binary microcode for MARIE's Add instruction. Assume that the microcode begins at instruction line number **0110100₂**.

Ans.

Address	MicroOp 1	MicroOp 2	Jump	Dest
0110100	01011	00000	0	0000000
0110101	01101	00000	0	0000000

0110110	00100	00000	1	0000000

60. Would you recommend a synchronous bus or an asynchronous bus for use between the CPU and the memory? Explain your answer.

Ans.

Whereas I/O buses are typically asynchronous, the CPU-memory bus is almost always synchronous. Synchronous buses are fast and run with a fixed rate. Every device on a synchronous bus must run at the same clock rate, but this works well with CPU-memory buses since the buses can be matched to the memory system to maximize memory-CPU bandwidth.

Since little or no logic is required to decide what to do next, a synchronous bus is both fast (offers better performance) and inexpensive. Due to clock-skew, the bus cannot be long (but this works fine for a CPU-memory bus). Asynchronous buses have overhead associated with synchronizing the bus but work well for longer buses.

*61. Pick an architecture (other than those covered in this chapter). Do research to find out how your architecture deals with the concepts introduced in this chapter, as was done for Intel and MIPS.

Ans.

None given.

62. Which control signals should contain a 1 for each step in executing the JUMPI instruction?

Ans.

Step	RTN	Time	P ₅	P ₄	P ₃	P ₂	P ₁	P ₀	C _r	IncrPC	M _R	M _W	L _{ALT}
Fetch	MAR ← PC	T ₀	0	0	1	0	1	0	0	0	0	0	0
	IR ← M[MAR]	T ₁	1	1	1	0	0	0	0	0	1	0	0
Decode IR[15-12]	PC ← PC + 1	T ₂	0	0	0	0	0	0	0	1	0	0	0
Get operand	MAR ← IR[11-0]	T ₃	0	0	1	1	1	1	0	0	0	0	0
Execute	MBR ← M[MAR]	T ₄	0	1	1	0	0	0	0	0	1	0	0
	PC ← MBR	T ₅	0	1	0	0	1	1	1	0	0	0	0

63. Which control signals should contain a 1 for each step in executing the STOREI instruction?

Ans.

Step	RTN	Time	P ₅	P ₄	P ₃	P ₂	P ₁	P ₀	C _r	IncrPC	M _R	M _W	L _{ALT}
Fetch	MAR ← PC	T ₀	0	0	1	0	1	0	0	0	0	0	0
	IR ← M[MAR]	T ₁	1	1	1	0	0	0	0	0	1	0	0
Decode IR[15-12]	PC ← PC + 1	T ₂	0	0	0	0	0	0	0	1	0	0	0
Get operand	MAR ← IR[11-0]	T ₃	0	0	1	1	1	1	0	0	0	0	0
Execute	MBR ← M[MAR]	T ₄	0	1	1	0	0	0	0	0	1	0	0
	MAR ← MBR	T ₅	0	0	1	0	1	1	0	0	0	0	0
	MBR ← AC	T ₆	0	1	1	1	0	0	1	0	0	0	1
	M[MAR] ← MBR	T ₇	0	0	0	0	1	1	1	0	1	1	0

64. The $PC \leftarrow PC + 1$ microoperation is executed at the end of every fetch cycle (to prepare for the next instruction to be fetched). However, if we execute a JUMP or a JUMPI instruction, the PC overwrites the value in the PC with a new one, thus voiding out the microoperation that incremented the PC. Explain how the microprogram for MARIE might be modified to be more efficient in this regard.

Ans.

The instruction to increment the PC could be moved to the start of the execution of an instruction if it is needed.

TRUE or FALSE

- _____ 1. If a computer uses hardwired control, the microprogram determines the instruction set for the machine. This instruction set can never be changed unless the architecture is redesigned.
- _____ 2. A branch instruction changes the flow of information by changing the PC.
- _____ 3. Registers are storage locations within the CPU itself.
- _____ 4. A two pass assembler generally creates a symbol table during the first pass and finishes the complete translation from assembly language to machine instructions on the second.

- _____ 5. The MAR, MBR, PC and IR registers in MARIE can be used to hold arbitrary data values.
- _____ 6. MARIE has a common bus scheme, which means a number of entities share the bus.
- _____ 7. An assembler is a program that accepts a symbolic language program and produces the binary machine language equivalent, resulting in a one-to-one correspondence between the assembly language source program and the machine language object program.
- _____ 8. If a computer uses microprogrammed control, the microprogram determines the instruction set for the machine.
- _____ 9. The length of a word determines the number of bits necessary in a memory address.
- _____ 10. If memory is 16-way interleaved, it means memory is implemented with 4 banks (since $2^4 = 16$).

Ans.

- | | | |
|------|------|-------|
| 1. F | 5. F | 9. F |
| 2. T | 6. T | 10. F |
| 3. T | 7. T | |
| 4. T | 8. T | |

Additional MARIE Programming Exercises

1. First, write a subroutine that, when sent a value, will triple that value. Then write a complete assembly program (using that subroutine) to allow the user to enter four grades. Assume that the last grade is a final exam grade and counts three times as much as the first three. Display the overall sum (including the last grade three times) and the average (sum/6).
2. Write a MARIE subroutine that will multiply two numbers by using repeated addition. For example, to multiply 2 x 8, the program would add 8 to itself twice (if the larger of the two numbers is chosen to be added to itself, the subroutine will run more quickly).
3. Write a MARIE program to raise an integer to a given power, using the subroutine written in problem 2 above.
4. Write a MARIE program to sum the numbers $1 + 2 + 3 + 4 + \dots + N$, where the user enters N.

5. Write a MARIE program that will sum numbers as the user enters them, which prints the running sum and the final sum. The user should enter a zero to terminate the program.
6. Write a MARIE program to calculate the sum: $1^2 + 2^2 + 3^2 + 4^2 + \dots + N^2$, where the user inputs the value N.
7. Write a MARIE program to calculate $N! = 1 \times 2 \times 3 \times 4 \times \dots \times N$, where the user enters N.
8. The Fibonacci numbers are defined as the sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ..., where, beginning with the third number, each number is the sum of the two prior numbers. We define $\text{Fib}(0)=0$, $\text{Fib}(1)=1$, $\text{Fib}(2)=1$, $\text{Fib}(3)=2$, $\text{Fib}(4)=3$, etc. Write a MARIE program to calculate $\text{Fib}(n)$, where the user inputs n.