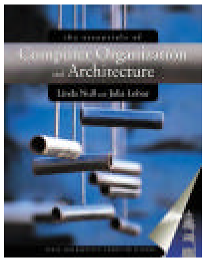


The Essentials of Computer Organization and Architecture

Linda Null and Julia Lobur
Jones and Bartlett Publishers, 2003

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, 10 lecture hours are more reasonable. If lecture time is limited, we suggest that the focus be on MARIE and understanding the components, 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.

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 emphasis 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 4.13, 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 ← PC	100	------	100	------	------
	IR ← M[MAR]	100	1104	100	------	------
	PC ← PC + 1	101	1104	100	------	------
Decode	MAR ← IR[11-0]	101	1104	104	------	------
	(decode IR[15-12]	101	1104	104	------	------
Get operand	MBR ← M[MAR]	101	1104	104	0023	------
Execute	AC ← MBR	101	1104	104	0023	0023

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

Load X	MAR ← X
	MBR ← M[MAR], AC ← MBR

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

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

JnS X	MBR ← PC
	MAR ← X
	M[MAR] ← MBR
	MBR ← X
	AC ← 1
	AC ← AC + MBR
	PC ← AC

The MAR ← X (or MAR ← 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. Originally we had intended to use only the two bits that indicated the branching condition in the instruction. For example, Skipcond 01 was to be the assembly language instruction for skipping the next instruction if the AC is equal to 0. However, to be consistent with the hexadecimal representation of the instructions, we use Skipcond 400 (which, in hex, is 8400, or 1000 0100 0000 0000). Please note that on Page 176, Example 4.1 and page 178, Example 4.2, the Skipcond instructions need to be changed (please see errata) to follow this format.

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. 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 of the program continues.

♦ 3. How many bits would you need to address a 2M × 32 memory if

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

Ans.

a. There are 2M × 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

4. How many bits are required to address a 4M $\times$ 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

---

5. 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 which equals 2^{20} total bytes, so 20 bits are needed for an address

b. There are 1M words which equals 2^{20}, so 20 bits are required for an address

---

♦ 6. Suppose that a 2M x 16 main memory is built using 256K $\times$ 8 RAM chips and memory is word-addressable.

♦ a. How many RAM chips are necessary?
♦ b. How many RAM chips are there per memory word?
♦ 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 6f 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

---

7. Redo Exercise 6 assuming a 16M $\times$ 16 memory built using 512K $\times$ 8 RAM chips.

Ans.

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

---

8. 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^{16} - 1

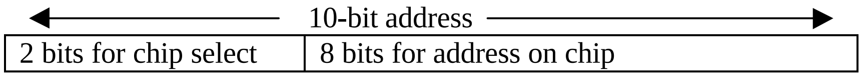
---

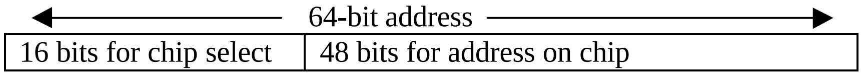
9. Assume a 2^{20} byte memory:
   ♦ a. What are the lowest and highest addresses if memory is byte-addressable?
   ♦ b. What are the lowest and highest addresses if memory is word-addressable, assuming a 16-bit word?
   c. What are the lowest and highest addresses if memory is word-addressable, assuming a 32-bit word?

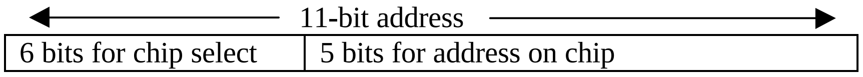
Ans.
   a. There are 2^{20} bytes, which can all be addressed using addresses 0 through 2^{20}-1 with 20 bit addresses
   b. There are only 2^{19} words and addressing each requires using addresses 0 through 2^{19} -1
   c. There are only 2^{18} words and addressing each requires using addresses 0 through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

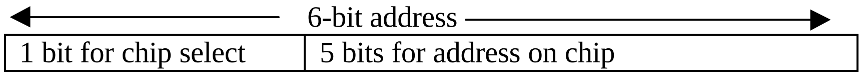
---

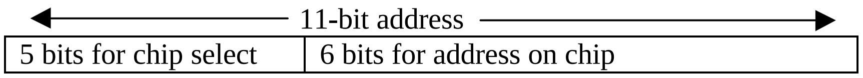
10. Given a memory of 2048 bytes consisting of several 64 Byte x 8 RAM chips. Assuming byte-addressable memory, which of the following seven diagrams indicates the correct way to use the address bits? Explain your answer.

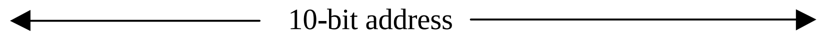
a.


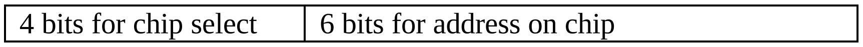
b.


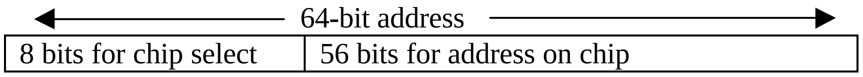
c.


d.


e.


f.




g. 

Ans.
The correct answer is e.

---

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

Ans.
The PC points to the next instruction to be fetched. When this instruction is fetched, it is placed into the IR, and the PC is incremented by 1. The decode cycle looks at the instruction in the IR to determine if data must be fetched. If so, the operand portion of the instruction is placed in the MAR, and the data is fetched and placed in the MBR. The instruction is then executed.

---

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

Only this hint was given to the students:
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.

---

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

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

Ans.
```
1108
3109
9106
3109
2108
7000
3108
9103
0023
```

```
0001
```

♦14. What are the contents of the symbol table for the preceding program?
Ans.

A	108
One	109
S1	106
S2	103

---

15. Given the instruction set for MARIE in this chapter:

a. Decipher the following MARIE machine language instructions (write the assembly language equivalent):

♦i) `0010000000000111`
ii) `1001000000001011`
iii) `0011000000001001`

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

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

c. 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.

a. i) `Store 007`
ii) `Jump 00B`
iii) `Add 009`

b.

```
If,     100    Load      X      /Load X
        101    Subt      One    /Subtract 1, store result in AC
        102    Skipcond  800    /If AC>0 (X>1), skip the next instruction
        103    Jump      Endif  /Jump to Endif if X is not greater than 1
Then,   104    Load      X      /Reload X so it can be doubled
        105    Add       X      /Double X
        106    Store     Y      /Y:= X + X
        107    Clear            /Move 0 into AC
```

```
       108    Store     X      /Set X to 0
Endif, 109    Load      Y      /Load Y into AC
       10A    Add       One    /Add 1 to Y
       10B    Store     Y      /Y := Y + 1
       10C    Halt             /Terminate program
X,     10D    Dec       ?      /X has starting value, not given in problem
Y,     10E    Dec       ?      /Y has starting value, not given in problem
One,   110    Dec       1      /Use as a constant
```

c. 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.

---

16. Write a MARIE program to evaluate the expression A x B + C x D.

Ans. Note: The line numbers as shown in the book are NOT necessary. They are included in the book to help students see the correlation between the assembly language instructions and the MARIE instructions.

```
        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
```

---

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

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

Ans.

```
         Load      One
         Store     X        /Initialize 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
```

♦ 18. Write the following code segment in MARIE assembly language:
(A hint, not a solution, was given to students on this problem.)

```
   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
         Add       X        /Add X to Sum
         Store     Sum      /Store result in Sum
         Load      X
         Add       One      /Increment X
         Store     X
         Jump      Loop
Endloop, Load      Sum
         Output             /Print Sum
         Halt               /terminate program
Sum,     Dec       0
X,       Dec       0        /Storage for X
One,     Dec       1        /The constant value 1
Ten,     Dec       10       /The loop constant
         END
```

---

19. 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
```

---

20. 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
```

---

21. 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 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
StoreSum
```

for a total of 5 memory accesses.

◆ 22. MARIE saves the return address for a subroutine in memory, at a location designated by the jump-and-store instruction. In some architectures, this address is stored in a register, and in many it is stored on a stack. Which of these methods would best handle recursion? Explain your answer.

Ans.

A stack would handle recursion more efficiently. The stack could grow as large as necessary to accommodate multiple calls to the subroutine. If there were only one register or one memory location, multiple calls to the subroutine from within the subroutine (i.e. recursion) would not be possible.

---

23. Provide a trace (similar to the one in Figure 4.13) for Example 4.2.

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

```
If,    100 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
                           /Note: This is Skipcond 01 in the book
       103 Jump     Else   /Jump to Else part if AC is not equal to 0
Then,  104 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
Else,  108 Load     Y      /Start the else part by loading Y
       109 Subt     X      /Subtract X from Y
       10A Store    Y      /Store Y-X in Y
Endif, 10B Halt            /Terminate program (it doesn't do much!)
X,     10C Dec      12     /Assume these values for X and Y
Y,     10D 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 (Skipcond 01 in book)
```

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 ← PC	103	8400	103	0014	FFF8
	IR ← M[MAR]	103	9108	103	0014	FFF8
	PC ← PC + 1	104	9108	103	0014	FFF8
Decode	MAR ← 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 ← 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 ← PC	108	9108	108	000C	FFF8
	IR ← M[MAR]	108	110D	108	000C	FFF8
	PC ← PC + 1	109	110D	108	000C	FFF8
Decode	MAR ← IR[11-0]	109	110D	10D	000C	FFF8
	(decode IR[15-12]	109	110D	10D	000C	FFF8
Get operand	MBR ← M[MAR]	109	110D	10D	0014	FFF8
Execute	AC ← MBR	109	110D	10D	0014	0014

```
Subt X
```

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		109	110D	10D	0014	0014
Fetch	MAR ← PC	109	110D	109	0014	0014
	IR ← M[MAR]	109	410C	109	0014	0014
	PC ← PC + 1	10A	410C	109	0014	0014
Decode	MAR ← IR[11-0]	10A	410C	10C	0014	0014
	(decode IR[15-12]	10A	410C	10C	0014	0014
Get operand	MBR ← M[MAR]	10A	410C	10C	000C	0014
Execute	AC ← 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 ← PC	10A	410C	10A	000C	0008
	IR ← M[MAR]	10A	210D	10A	000C	0008
	PC ← PC + 1	10B	210D	10A	000C	0008
Decode	MAR ← 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 ← AC	10B	210D	10D	0008	0008
(changes Y)	M[MAR] ← MBR	10B	210D	10D	0008	0008

Halt

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		10B	210D	10D	0008	0008
Fetch	MAR ← PC	10B	210D	10B	0008	0008
	IR ← M[MAR]	10B	7000	10B	0008	0008
	PC ← PC + 1	10C	7000	10B	0008	0008
Decode	MAR ← 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

---

24. Provide a trace (similar to the one in Figure 4.13) for Example 4.3.

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
X,    109   Dec         20
Y,    10A   Dec         48
Temp, 10B   Dec         0
Subr, 10C   Hex         0     /Store return address here
      10D   Clear             /Clear AC as it was modified by JnS
      10E   Load        Temp  /Actual subroutine to double numbers
      10F   Add         Temp  /AC now hold double the value of Temp
      110   JumpI       Subr  /Return to calling code
            END
```

`Load X`

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

`Store Temp`

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		101	1109	109	0014	0014
Fetch	MAR ← PC	101	1109	101	0014	0014
	IR ← M[MAR]	101	210B	101	0014	0014
	PC ← PC + 1	102	210B	101	0014	0014
Decode	MAR ← 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 ← AC	102	210B	10B	0014	0014
(changes Temp)	M[MAR] ← MBR	102	210B	10B	0014	0014

JnS Subr

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		102	210B	10B	0014	0014
Fetch	MAR ← PC	102	210B	102	0014	0014
	IR ← M[MAR]	102	010C	102	0014	0014
	PC ← PC + 1	103	010C	102	0014	0014
Decode	MAR ← 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 ← PC	103	010C	10C	0103	0014
	MAR ← IR[11-0]	103	010C	10C	0103	0014
(changes Subr)	M[MAR] ← MBR	103	010C	10C	0103	0014
	MBR ← IR[11-0]	103	010C	10C	010C	0014
	AC ← 1	103	010C	10C	010C	0001
	AC ← AC + MBR	103	010C	10C	010C	010D
	PC ← AC	10D	010C	10C	010C	010D

`Clear`

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		10D	010C	10C	010C	010D
Fetch	MAR ← PC	10D	010C	10D	010C	010D
	IR ← M[MAR]	10D	A000	10D	010C	010D
	PC ← PC + 1	10E	A000	10D	010C	010D
Decode	MAR ← IR[11-0]	10E	A000	000	010C	010D
	(decode IR[15-12]	10E	A000	000	010C	010D
Get operand	(not necessary)	10E	A000	000	010C	010D
Execute	AC ← 0	10E	A000	000	010C	0000

`Load Temp`

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

`Add Temp`

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

`JumpI Subr`

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		110	310B	10B	0014	0028
Fetch	MAR ← PC	110	310B	110	0014	0028
	IR ← M[MAR]	110	C10C	110	0014	0028
	PC ← PC + 1	111	C10C	110	0014	0028
Decode	MAR ← IR[11-0]	111	C10C	10C	0014	0028
	(decode IR[15-12]	111	C10C	10C	0014	0028
Get operand	MBR ← M[MAR]	111	C10C	10C	0103	0028
Execute	PC ← MBR	103	C10C	10C	0103	0028

`Store X`

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		103	C10C	10C	0103	0028
Fetch	MAR ← PC	103	C10C	103	0103	0028
	IR ← M[MAR]	103	2109	103	0103	0028
	PC ← PC + 1	104	2109	103	0103	0028
Decode	MAR ← 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 ← AC	104	2109	109	0028	0028
(changes X)	M[MAR] ← MBR	104	2109	109	0028	0028

`Load Y`

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

`Store Temp`

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		105	110A	10A	0030	0030
Fetch	MAR ← PC	105	110A	105	0030	0030
	IR ← M[MAR]	105	210B	10A	0030	0030
	PC ← PC + 1	106	210B	10A	0030	0030
Decode	MAR ← 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 ← AC	106	210B	10B	0030	0030
(changes Temp)	M[MAR] ← MBR	106	210B	10B	0030	0030

JnS Subr

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		106	210B	10B	0030	0030
Fetch	MAR ← PC	106	210B	106	0030	0030
	IR ← M[MAR]	106	010C	106	0030	0030
	PC ← PC + 1	107	010C	106	0030	0030
Decode	MAR ← 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 ← PC	107	010C	10C	0107	0014
	MAR ← IR[11-0]	107	010C	10C	0107	0014
(changes Subr)	M[MAR] ← MBR	107	010C	10C	0107	0014
	MBR ← IR[11-0]	107	010C	10C	010C	0014
	AC ← 1	107	010C	10C	010C	0001
	AC ← AC + MBR	107	010C	10C	010C	010D
	PC ← AC	10D	010C	10C	010C	010D

`Clear`

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		10D	010C	10C	010C	010D
Fetch	MAR ← PC	10D	010C	10D	010C	010D
	IR ← M[MAR]	10D	A000	10D	010C	010D
	PC ← PC + 1	10E	A000	10D	010C	010D
Decode	MAR ← IR[11-0]	10E	A000	000	010C	010D
	(decode IR[15-12]	10E	A000	000	010C	010D
Get operand	(not necessary)	10E	A000	000	010C	010D
Execute	AC ← 0	10E	A000	000	010C	0000

Load Temp

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

Add Temp

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

JumpI Subr

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		110	310B	10B	0030	0060
Fetch	MAR ← PC	110	310B	110	0030	0060
	IR ← M[MAR]	110	C10C	110	0030	0060
	PC ← PC + 1	111	C10C	110	0030	0060
Decode	MAR ← IR[11-0]	111	C10C	10C	0030	0060
	(decode IR[15-12]	111	C10C	10C	0030	0060
Get operand	MBR ← M[MAR]	111	C10C	10C	0107	0060
Execute	PC ← MBR	107	C10C	10C	0107	0060

Store Y

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		107	C10C	10C	0107	0060
Fetch	MAR ← PC	107	C10C	107	0107	0060
	IR ← M[MAR]	107	210A	107	0107	0060
	PC ← PC + 1	108	210A	107	0107	0060
Decode	MAR ← 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 ← AC	108	210A	10A	0060	0060
(changes Y)	M[MAR] ← MBR	108	210A	10A	0060	0060

```
Halt
```

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		108	210A	10A	0060	0060
Fetch	MAR ← PC	108	210D	108	0060	0060
	IR ← M[MAR]	108	7000	108	0060	0060
	PC ← PC + 1	109	7000	108	0060	0060
Decode	MAR ← 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

---

25. 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 written using RTN.

Ans.

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

---

26. 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.

---

*27. 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.

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. As assembler is a program that accepts a symbolic language program and produces the binary machine language equivalent, resulting in a 1-to-1 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.

Ans.

1. F	5. F
2. T	6. T
3. T	7. T
4. T	8. T

Sample Exam Questions

1. Identify the following register transfer statements as legal or not legal for the datapath used in MARIE. If it is not legal, rewrite it as a sequence of microoperations to perform the indicated task.

   a. IR $\leftarrow$ MAR

   b. MBR $\leftarrow$ M[PC]

   c. AC $\leftarrow$ AC + PC

   d. MAR $\leftarrow$ PC

Ans.

   a. Legal

b. The PC can't be used directly for memory access, so this would need to be rewritten:

   MAR ← PC
   MBR ← M[MAR]

c. No, only the MBR can be added to the AC, so this would need to be rewritten:

   MBR ← PC
   AC ← AC + MBR

d. Legal

---

2. The instruction AddI 085 is stored at memory location 100 and is fetched, decoded, and executed. Give the contents of PC, MAR, IR, MBR and AC as the instruction is processed. Assume memory contains:

Location	Contents (hex)
085	0087
086	9085
087	1086

Give all answers in hexadecimal. Assume the initial values as given in the table below.

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		100	---	---	---	0004
Fetch	MAR ← PC	100	---	100	---	0004
	IR ← M[MAR]	100	B085	100	---	0004
	PC ← PC + 1	101	B085	100	---	0004
Decode	MAR ← IR[11-0]	101	B085	085	---	0004
	(decode IR[15-12]	101	B085	085	---	0004
Get operand	MBR ← M[MAR]	101	B085	085	0087	0004
Execute	MAR ← MBR	101	B085	087	0087	0004
	MBR ← M[MAR]	101	B085	087	1086	0004
	AC ← AC + MBR	101	B085	087	1086	108A

---

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

```
        ORG      000
        Input
        Store    C
Loop,   Clear
        Load     C
        Subt     B
        Output
        Store    C
        Subt     A
        Skipcond 400
        Jump     Loop
        Halt
A,      Dec      32
B,      Dec      1
C,      Dec      0
```

Ans.

```
5000
200D
A000
100D
400C
6000
200D
400B
8400
9002
7000
0020
0001
0000
```

---

4. Decipher the following MARIE machine language instruction (write the assembly language equivalent):

```
001100000000000A
```

Ans.

300A or Add 00A

---

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

```
if X > 1 do
   X := X + 1;
else
   Y := Y + 1;
```

Ans.

```
If,     100    Load      X      /Load X
        101    Subt      One    /Subtract 1, store result in AC
        102    Skipcond  800    /If AC>0 (X>1), skip the next instruction
        103    Jump      Else   /Branch to Else
        104    Load      X      /Load X
Then,   105    Add       One    /Add 1
        106    Store     X      /X:= X + 1
        107    Jump      Endif  /Jump over Else part
Else    108    Load      Y      /Load Y
        109    Add       One    /Add 1
        10A    Store     Y      /Y:= Y + 1
Endif,  10B    Halt             /Terminate program
One,    10C    DEC       1      /Variable One has value 1
X,      10D    DEC       ?
Y,      10E    DEC       ?
```

---

6. How many address lines (bits in the address) and I/O lines (bits in the actual data) are needed for each of the following word-addressable memories?

   a. $2K \times 16$

   b. $16K \times 8$

   c. $4M \times 12$

Ans.

   a. 11 address bits and 16 I/O lines
   b. 14 address bits and 8 I/O lines
   c. 22 address bits and 12 I/O lines

---

7. Match the following:

_____ Holds data just read from memory
_____ Holds data the CPU needs to process
_____ Holds next instruction to be executed
_____ Holds address of next instruction to be executed
_____ Holds memory address of data being referenced
_____ Holds data written from the keyboard
_____ Holds interrupt signals

A. Instruction Register	E. Accumulator
B. Program Counter	F. Memory Buffer Register
C. Input Register	G. Flag Register
D. Memory Address Register	

Ans.

F, E, A, B, D, C, G

---

8. Given the following program:

```
       100    Load   D
       101    Subt   C
       102    Store  D
       103    Add    A
       104    Store  A
       105    JumpI  A
       106    AddI   B
       107    Subt   C
       108    Store  B
       109    Jump   X
X,     10A    Halt
A,     10B    Hex    99
B,     10C    Hex    10F
C,     10D    Hex    1
D,     10E    Hex    8
E,     10F    Hex    3
```

When this program terminates, what values will be in:

a. Memory location 10B
b. Memory location 10C
c. Memory location 10D
d. Memory location 10E
e. Memory location 10F
f. The Accumulator
g. The Program Counter

Ans.

a. 106
b. 10B
c. 1
d. 6
e. 3
f. 10B
g. 10B

Location

	A	B	C	D	E		
Instruction	**10B**	**10C**	**10D**	**10E**	**10F**	**AC**	**PC**
LOAD D	99	10F	1	8	3	8	101
SUBT C	99	10F	1	8	3	7	102
STORE D	99	10F	1	7	3	7	103
ADD A	99	10F	1	7	3	100	104
STORE A	100	10F	1	7	3	100	105
JUMPI A	100	10F	1	7	3	100	100
LOAD D	100	10F	1	7	3	7	101
SUBT C	100	10F	1	7	3	6	102
STORE D	100	10F	1	6	3	6	103
ADD A	100	10F	1	6	3	106	104
STORE A	106	10F	1	6	3	106	105
JUMPI A	106	10F	1	6	3	106	106
ADDI B	106	10F	1	6	3	109	107
SUBT C	106	10F	1	6	3	108	108
STORE B	106	108	1	6	3	10B	109
JUMP X	106	10B	1	6	3	10B	10A
HALT	106	10B	1	6	3	10B	10B

9. Fill in all necessary information in the diagram of the Fetch-Decode-Execute cycle below.