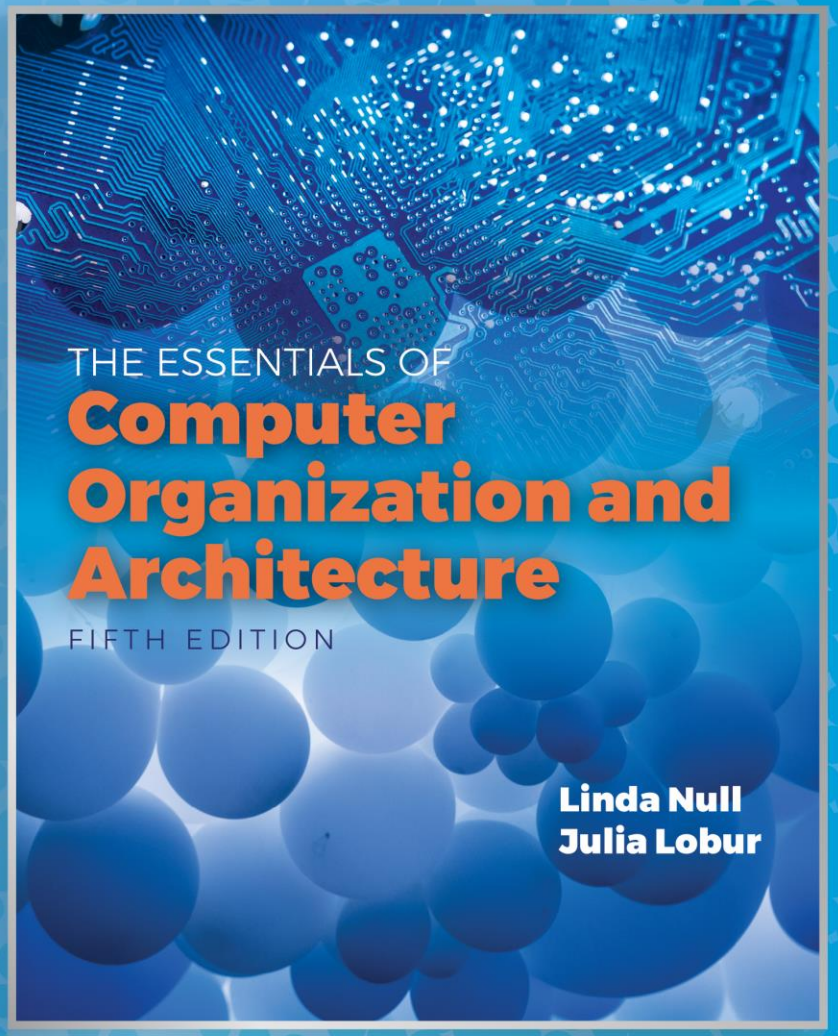
This is the second lecture of Chapter 4

Chapter 4

MARIE: An Introduction to a Simple Computer



4.8 MARIE (1 of 12)

- We can now bring together many of the ideas that we have discussed to this point using a very simple model computer.
- Our model computer, the Machine Architecture that is Really Intuitive and Easy (MARIE) was designed for the singular purpose of illustrating basic computer system concepts.
- While this system is too simple to do anything useful in the real world, a deep understanding of its functions will enable you to comprehend system architectures that are much more complex.

4.8 MARIE (2 of 12)

- The MARIE architecture has the following characteristics:
  - Binary, two's complement data representation.
  - Stored program, fixed word length data and instructions.
  - 4K words of word-addressable main memory.
  - 16-bit data words.
  - 16-bit instructions, 4 for the opcode and 12 for the address.
  - A 16-bit arithmetic logic unit (ALU).
  - Seven registers for control and data movement.

4.8 MARIE (3 of 12)

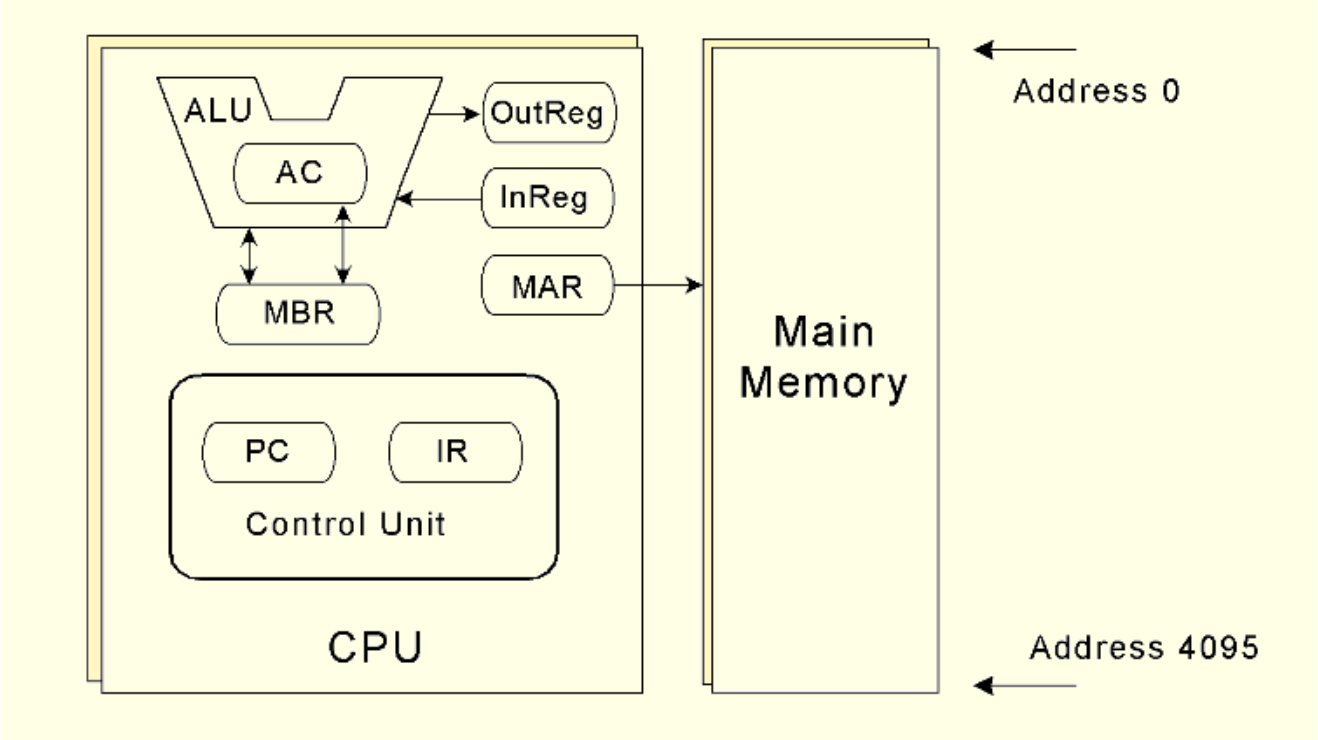
- MARIE’s seven registers are:
  1. Accumulator, AC, a 16-bit register that holds a conditional operator (e.g., "less than") or one operand of a two-operand instruction.
  2. Memory address register, MAR, a 12-bit register that holds the memory address of an instruction or the operand of an instruction.
  3. Memory buffer register, MBR, a 16-bit register that holds the data after its retrieval from, or before its placement in memory.

4.8 MARIE (4 of 12)

4. Program counter, PC, a 12-bit register that holds the address of the next program instruction to be executed.
5. Instruction register, IR, which holds an instruction immediately preceding its execution.
6. Input register, InREG, an 8-bit register that holds data read from an input device.
7. Output register, OutREG, an 8-bit register, that holds data that is ready for the output device.

4.8 MARIE (5 of 12)

- This is the MARIE architecture shown graphically.

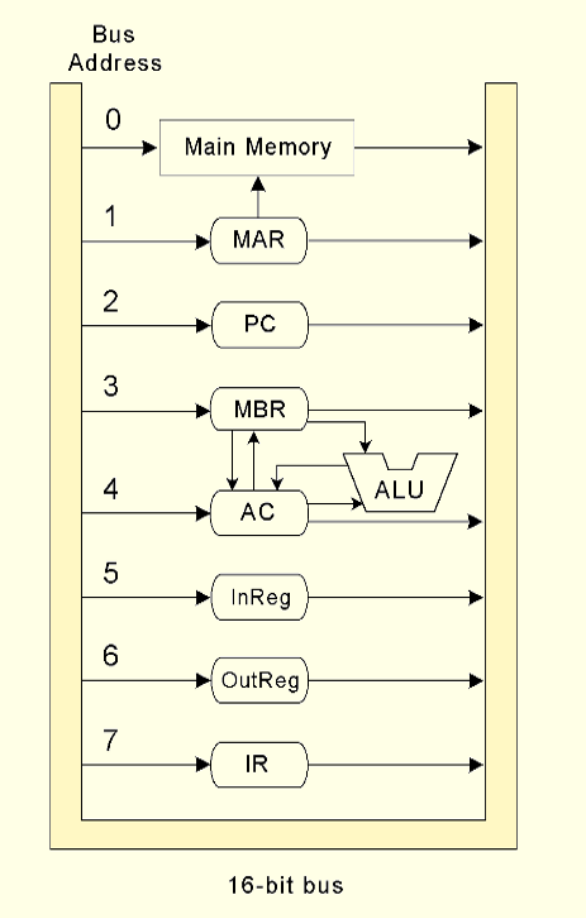


4.8 MARIE (6 of 12)

- The registers are interconnected, and connected with main memory through a common data bus.
- Each device on the bus is identified by a unique number that is set on the control lines whenever that device is required to carry out an operation.
- Separate connections are also provided between the accumulator and the memory buffer register, and the ALU and the accumulator and memory buffer register.
- This permits data transfer between these devices without use of the main data bus.

4.8 MARIE (7 of 12)

- This is the MARIE data path shown graphically.

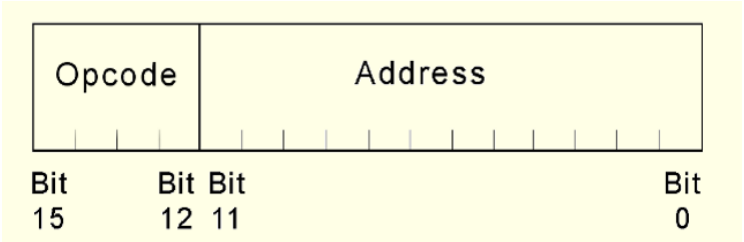


4.8 MARIE (8 of 12)

- A computer's instruction set architecture (ISA) specifies the format of its instructions and the primitive operations that the machine can perform.
- The ISA is an interface between a computer's hardware and its software.
- Some ISAs include hundreds of different instructions for processing data and controlling program execution.
- The MARIE ISA consists of only 13 instructions.

4.8 MARIE (9 of 12)

- This is the format of a MARIE instruction:

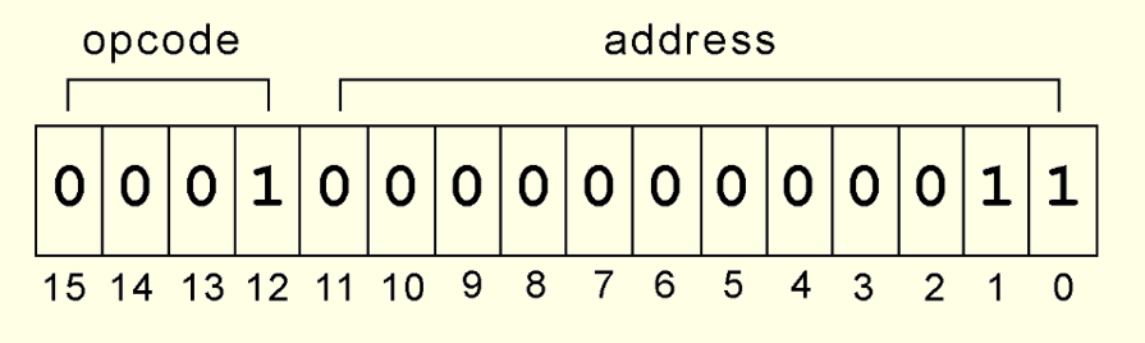


- The fundamental MARIE instructions are:

Instruction Number			
Binary	Hex	Instruction	Meaning
0001	1	Load X	Load contents of address X into AC.
0010	2	Store X	Store the contents of AC at address X.
0011	3	Add X	Add the contents of address X to AC.
0100	4	Subt X	Subtract the contents of address X from AC.
0101	5	Input	Input a value from the keyboard into AC.
0110	6	Output	Output the value in AC to the display.
0111	7	Halt	Terminate program.
1000	8	Skipcond	Skip next instruction on condition.
1001	9	Jump X	Load the value of X into PC.

4.8 MARIE (10 of 12)

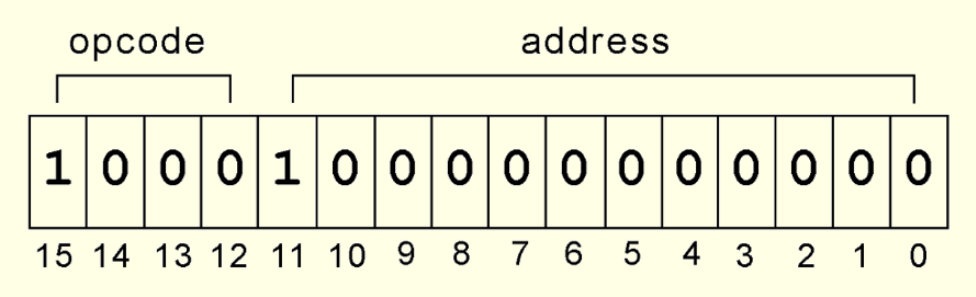
- This is a bit pattern for a **LOAD** instruction as it would appear in the IR:



- We see that the opcode is 1 and the address from which to load the data is 3.

4.8 MARIE (11 of 12)

- This is a bit pattern for a **`SKIPCOND`** instruction as it would appear in the IR:

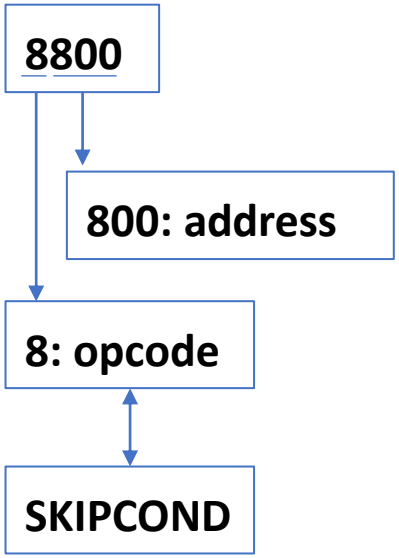


- We see that the opcode is 8 and bits 11 and 10 are 10, meaning that the next instruction will be skipped if the value in the AC is greater than zero.

What is the hexadecimal representation of this instruction?

4.8 MARIE (12 of 12)

What is the hexadecimal representation of this instruction?



Three SKIPCOND Instructions and their meanings

1. SKIPCOND 000
If AC < 0 then PC ← PC + 1
2. SKIPCOND 400
If AC = 0 then PC ← PC + 1
3. SKIPCOND 800
If AC > 0 then PC ← PC + 1

4.11 A Discussion on Assemblers (1 of 4)

- Mnemonic instructions, such as **`LOAD 104`**, are easy for humans to write and understand.
- They are impossible for computers to understand.
- *Assemblers* translate instructions that are comprehensible to humans into the machine language that is comprehensible to computers
  - We note the distinction between an assembler and a compiler: In assembly language, there is a one-to-one correspondence between a mnemonic instruction and its machine code. With compilers, this is not usually the case.

4.11 A Discussion on Assemblers (2 of 4)

- Assemblers create an *object program file* from mnemonic *source code* in two passes.
- During the first pass, the assembler assembles as much of the program as it can, while it builds a *symbol table* that contains memory references for all symbols in the program.
- During the second pass, the instructions are completed using the values from the symbol table.

4.11 A Discussion on Assemblers (3 of 4)

- Consider our example program at the right.
  - Note that we have included two directives **HEX** and **DEC** that specify the radix of the constants.
- The first pass, creates a symbol table and the partially-assembled instructions as shown.

Address		Instruction	
100		Load	X
101		Add	Y
102		Store	Z
103		Halt	
104	X,	DEC	35
105	Y,	DEC	-23
106	Z,	HEX	0000

X	104
Y	105
Z	106

1	X
3	Y
2	Z
7000	

4.11 A Discussion on Assemblers (4 of 4)

- After the second pass, the assembly is complete.

1104
3105
2106
7000
0023
FFE9
0000

X	104
Y	105
Z	106

Address		Instruction	
100		Load	X
101		Add	Y
102		Store	Z
103		Halt	
104	X,	DEC	35
105	Y,	DEC	-23
106	Z,	HEX	0000

4.12 Extending Our Instruction Set (1 of 2)

- So far, all of the MARIE instructions that we have discussed use a *direct addressing mode*.
- This means that the address of the operand is explicitly stated in the instruction.
- It is often useful to employ a *indirect addressing*, where the address of the address of the operand is given in the instruction.
  - If you have ever used pointers in a program, you are already familiar with indirect addressing.

4.12 Extending Our Instruction Set (2 of 2)

- The Four indirect memory access instructions:
  - **`LOADI X`** means AC ← M[M[X]]
  - **`STOREI X`** means M[M[X]] ← AC
  - **`ADDI X`** means AC ← AC + M[M[X]]
  - **`JUMPI X`** means PC ← M[X]
- The jump-and-store instruction
  - **`JNS X`**
    - Save return address at memory location X and jump to subroutine starting at X+1
    - M[X] ← PC and PC ← X+1
- The `clear` instruction
  - CLEAR means AC ← 0

Does JNS permit recursive calls?

MARIE Assembly Program: Example 4.0

Compute Sum = Num1 + Num2 and display Sum

```
        ORG  100          /Example 4.0
        Load  Num1        /Load address of first number to be added
        Add   Num2        /Add the second number
        Store Sum         /Store sum at address Sum
        Output            /Display
        Halt              /Terminate program

Num1,   Hex     0023      /The first number to add
Num2,   Hex     FFE9      /The second number to add
Sum,    Hex     0000      /The sum
```

Note that Num1 and Num2 are 2's complement representation.

Num1 = 35 and Num2 = -23

MARIE Assembly Program: Example 4.1

Get two integers from keyboard and store them in Num1 and Num2, respectively. Then compute Sum = Num1 + Num2 and display Sum

```
        ORG  100            /Example 4.1
        Input               /Get the first integer from keyboard into AC
        Store  Num1         /Save the first integer in Num1
        Input               /Get the second integer from keyboard into AC
        Store  Num2         /Save the second integer in Num1
        Add    Num1         /Add the first integer with the second integer
        Store Sum           /Store sum at address Sum
        Output              /Display
        Halt                /Terminate program

Num1,   Hex      0000       /The first integer to add
Num2,   Hex      0000       /The second integer to add
Sum,    Hex      0000       /The sum
```

Example 4.2 (Cont.)

Using loop to add five numbers in array, save result to Sum, and display Sum

113	Addr,	Hex	119	/Numbers to be summed start at location 119
114	Next,	Hex	0	/A pointer to the next number to add
115	Num,	Dec	5	/The number of values to add
116	Sum,	Dec	0	/The sum
117	Ctr,	Hex	0	/The loop control variable
118	One,	Dec	1	/Used to increment and decrement by 1
119		Dec	10	/The values to be added together
11A		Dec	15	
11B		Dec	20	
11C		Dec	25	
11D		Dec	30	

MARIE Assembly Program: Example 4.2

Using loop to add five numbers in array, save result to Sum, and display Sum

```
100
101
102
103
104
105
106
107
108
109
10A
10B
10C
10D
10E
10F
110
111
112
```

```
            ORG   100          /Example 4.2
            Load  Addr         /Load address of first number to be added
            Store Next         /Store this address is our Next pointer
            Load  Num          /Load the number of items to be added
            Subt  One          /Decrement
            Store Ctr          /Store this value in Ctr to control looping
Loop,       Load  Sum          /Load the Sum into AC
            AddI  Next         /Add the value pointed to by location Next
            Store Sum          /Store this sum
            Load  Next         /Load Next
            Add   One          /Increment by one to point to next address
            Store Next         /Store in our pointer Next
            Load  Ctr          /Load the loop control variable
            Subt  One          /Subtract one from the loop control variable
            Store Ctr          /Store this new value in loop control variable
            Skipcond 000       /If control variable < 0, skip next instruction
            Jump  Loop         /Otherwise, go to Loop
            Load  Sum
            Output             /Display Sum
            Halt               /Terminate program
```

Example 4.2 (Cont.)

Using loop to add five numbers in array, save result to Sum, and display Sum

113	Addr,	Hex	119	/Numbers to be summed start at location 119
114	Next,	Hex	0	/A pointer to the next number to add
115	Num,	Dec	5	/The number of values to add
116	Sum,	Dec	0	/The sum
117	Ctr,	Hex	0	/The loop control variable
118	One,	Dec	1	/Used to increment and decrement by 1
119		Dec	10	/The values to be added together
11A		Dec	15	
11B		Dec	20	
11C		Dec	25	
11D		Dec	30	

MARIE Assembly Program: Example 4.3

```
100
101
102
103
104
105
106
107
108
109
10A
```

```
/ This program traverses a string and outputs each character. The string is terminated with
/ a null character.

            ORG 100
Getch,      LoadI       Chptr       / Load the character found at address chptr.
            Skipcond    400         / If the character is a null, we are done.
            Jump        Outp        / Otherwise, proceed with operation.
            Halt
Outp,       Output                  / Output the character.
            Load        Chptr       / Move pointer to
            Add         One         / next character.
            Store       Chptr
            Jump        Getch
One,        Hex         0001
Chptr,      Hex         10B
```

```
10B
```

```
String,     Dec     072  / H
            Dec     101  / e
            Dec     108  / l
            Dec     108  / l
            Dec     111  / o
            Dec     032  / [space]
            Dec     119  / w
            Dec     111  / o
            Dec     114  / r
            Dec     108  / l
            Dec     100  / d
            Dec     033  / !
            Dec     000  / [null]
```

MARIE Assembly Program: Example 4.4

```
/This example illustrates the use of a simple subroutine to double the value stored at Temp

            ORG     100
            Load    X       / Load the first number to be doubled.
            Store   Temp    / Use Temp as a parameter to pass value to Subr.
            JnS     Subr    / Store the return address, and jump to the procedure.
            Store   X       / Store the first number, doubled
            Load    Y       / Load the second number to be doubled.
            Store   Temp
            JnS     Subr    / Store the return address and jump to the procedure.
            Store   Y       / Store the second number doubled.
            Halt            / End program.

X,          DEC     20
Y,          DEC     48
Temp,       DEC     0
Subr,       HEX     0       / Store return address here.
            Load    Temp    / Actual subroutine to double numbers.
            Add     Temp    / AC now holds double the value of Temp.
            JumpI   Subr    / Return to calling code.
            END
```