WEEK-12

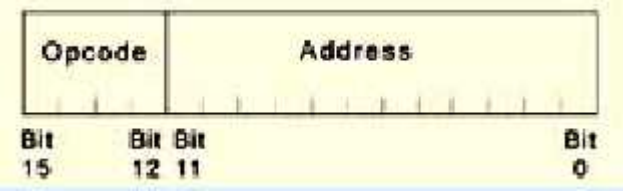
3.6.3 The Instruction Set Architecture.

MARIE has a very simple, yet powerful, instruction set. The *instruction set architecture* (*ISA*) of a machine specifies the instructions that the computer can perform and the format for each instruction. The ISA is essentially an interface between the software and the hardware. Some ISAs include hundreds of instructions.

We mentioned previously that each instruction for MARIE consists of 16 bits. The most significant 4 bits, bits 12–15, make up the *opcode* that specifies the instruction to be executed (which allows for a total of 16 instructions). The least significant 12 bits, bits 0–11, form an address, which allows for a maximum memory size of 2^{12-1}.

• The MARIE ISA consists of only thirteen instructions.

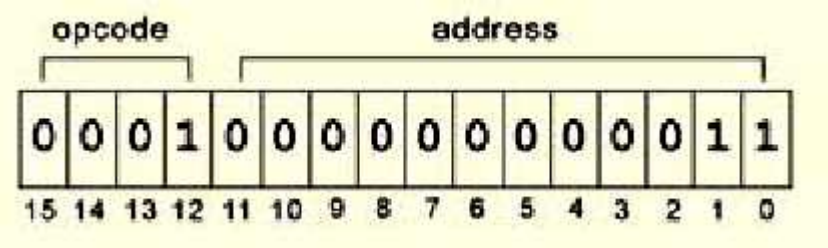
This is the format of a MARIE instruction:



• The fundamental MARIE instructions are:

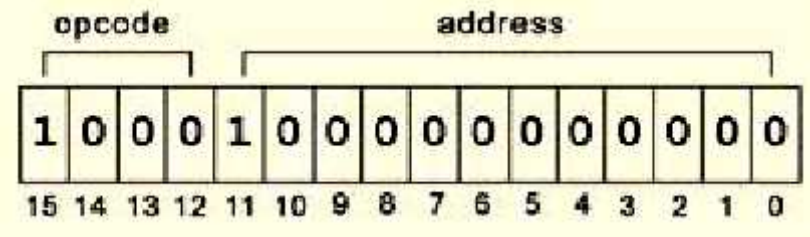
Instruction Number		Instruction	Meaning
Bin	Hex		
0001	1	Load *X*	Load the contents of address *X* into AC.
0010	2	Store *X*	Store the contents of AC at address *X*.
0011	3	Add *X*	Add the contents of address *X* to AC and store the result in AC.
0100	4	Subt *X*	Subtract the contents of address *X* from AC and store the result in 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 the program.
1000	8	Skipcond	Skip the next instruction on condition.
1001	9	Jump *X*	Load the value of *X* into PC.

• 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 000000000011

• 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.

3.6.4 Register Transfer Notation.

We have seen that digital systems consist of many components, including arithmetic logic units, registers, memory, decoders, and control units. These units are interconnected by buses to allow information to flow through the system. The instruction set presented for MARIE in the preceding sections constitutes a set of machine level instructions used by these components to execute a program. Each instruction appears to be very simplistic; however, if you examine what actually happens at the component level, each instruction involves multiple operations.

For example, the Load instruction loads the contents of the given memory location into the AC register. But, if we observe what is happening at the component level, we see that multiple “mini-instructions” are being executed. First, the address from the instruction must be loaded into the MAR. Then the data in memory at this location must be loaded into the MBR. Then the MBR must be loaded into the AC. These mini-instructions are called micro operations and specify the elementary operations that can be performed on data stored in registers.

The symbolic notation used to describe the behavior of micro operations is called ***register transfer notation (RTN)*** or ***register transfer language (RTL).*** We use the notation M[X] to indicate the actual data stored at location X in memory, and ← to indicate a

transfer of information. In reality, a transfer from one register to another always involves a transfer onto the bus from the source register, and then a transfer off the bus into the destination register.

However, for the sake of clarity, we do not include these bus transfers, assuming that you understand that the bus must be used for data transfer.

We now present the register transfer notation for each of the instructions in the ISA for MARIE.

- ***Load X.***

Recall that this instruction loads the contents of memory location *X* into the AC. However, the address *X* must first be placed into the MAR. Then the data at location M[MAR] (or address *X*) is moved into the MBR. Finally, this data is placed in the AC.

$$\mathbf{MAR} \leftarrow \boldsymbol{X}$$

$$\mathbf{MBR} \leftarrow \mathbf{M[MAR]}$$

$$\mathbf{AC} \leftarrow \mathbf{MBR}$$

Because the IR must use the bus to copy the value of *X* into the MAR, before the data at location *X* can be placed into the MBR, this operation requires two bus cycles. Therefore, these two operations are on separate lines to indicate they cannot occur during the same cycle. However, because we have a special connection between the MBR and the AC, the transfer of the data from the MBR to the AC can occur immediately after the data is put into the MBR, without waiting for the bus.

- ***Store X.***

This instruction stores the contents of the AC in memory location *X*:

MAR ← *X*

MBR ← AC

M [MAR] ← MBR

- ***Add X.***

The data value stored at address *X* is added to the AC. This can be accomplished as follows:

MAR ← X

MBR ← M [MAR]

AC ← AC + MBR

- ***Subt. X.***

Similar to *Add*, this instruction subtracts the value stored at address *X* from the accumulator and places the result back in the AC:

MAR ← *X*

MBR ← M [MAR]

AC ← AC – MBR

- ***Input.***

Any input from the input device is first routed into the InREG. Then the data is transferred into the AC.

AC ← InREG

- ***Output***.

This instruction causes the contents of the AC to be placed into the OutREG, where it is eventually sent to the output device.

OutREG ← AC

- ***Halt.***

No operations are performed on registers; the machine simply ceases execution.

- ***Skipcond.***

Recall that this instruction uses the bits in positions 10 and 11 in the address field to determine what comparison to perform on the AC. Depending on this bit combination, the AC is checked to see whether it is negative, equal to zero, or greater than zero. If the given condition is true, then the next instruction is skipped. This is performed by incrementing the PC register by 1.

if IR[11–10] = 00 then {if bits 10 and 11 in the IR are both 0}
If AC < 0 then PC ← PC+1
else If IR[11–10] = 01 then {if bit 11 = 0 and bit 10 = 1}
If AC = 0 then PC ← PC + 1
else If IR[11–10] = 10 then {if bit 11 = 1 and bit 10 = 0}
If AC > 0 then PC ← PC + 1

If the bits in positions ten and eleven are both ones, an error condition results. However, an additional condition could also be defined using these bit values.

- ***Jump X.***

This instruction causes an unconditional branch to the given address *X*. Therefore to execute this instruction, *X* must be loaded into the PC.

$$\mathbf{PC} \leftarrow \boldsymbol{X}$$

In reality the lower or least significant 12 bits of the instruction register (or IR[11–0]) reflect the value of *X*. So this transfer is more accurately depicted as:

$$\mathbf{PC} \leftarrow \mathbf{IR\ [11–0]}$$

However, we feel that the notation **PC** ← ***X*** is easier to understand and relate to the actual instructions, so we use this instead.

Register transfer notation is a symbolic means of expressing what is happening in the system when a given instruction is executing. RTN is sensitive to the data path, in that if multiple micro operations must share the bus, they must be executed in a sequential fashion, one following the other.