

MARIE: An Introduction to a Simple Computer

- In this chapter, we first look at a very simple computer called MARIE: A Machine Architecture that is Really Intuitive and Easy.
- The objective of this chapter is to give you an understanding of how a computer functions.

4.1 CPU Basics and Organization

- A computer must manipulate binary-coded data. Memory is used to store both data and program instructions (also in binary).
- Somehow, the program must be executed and the data must be processed correctly.
- The central processing unit (CPU) is responsible for fetching program instructions, decoding each instruction that is fetched, and performing the indicated sequence of operations on the correct data.
- All computers have a central processing unit. This unit can be divided into two pieces.
- **The first is the data path**, which is a network of storage units (registers) and arithmetic and logic units (for performing various operations on data) connected by buses (capable of moving data from place to place) where the timing is controlled by clocks.
- **The second CPU component is the control unit**: A module responsible for sequencing operations and making sure the correct data is where it needs to be at the correct time.
- Together, these components perform the tasks of the CPU: fetching instructions, decoding them, and finally performing the indicated sequence of operations.

4.1.1 The Registers.

- Registers are used in computer systems as places to store a wide variety of data, such as addresses, program counters, or data necessary for program execution.
- A register is a hardware device that stores binary data.

- Registers are located on the processor so information can be accessed very quickly.
- To build a 16-bit register, we need to connect 16 D flip-flops together.
- At each pulse of the clock, input enters the register and cannot be changed (and thus is stored) until the clock pulses again.
- Data processing on a computer is usually done on fixed size binary words that are stored in registers.
- Common sizes include 16, 32, and 64 bits.
- The number of registers in a machine varies from architecture to architecture, but is typically a power of 2, with 16 and 32 being most common.
- Registers contain data, addresses, or control information.
- Some registers are specified as "special purpose" and may contain only data, only addresses, or only control information.
- Other registers are more generic and may hold data, addresses, and control information at various times.
- In modern computer systems, there are many types of specialized registers: Registers to store information, registers to shift values, registers to compare values, and registers that count.
- There are "scratchpad" registers that store temporary values, index registers to control program looping, stack pointer registers to manage stacks of information for processes, status registers to hold the status or mode of operation (such as overflow, carry, or zero conditions), and general purpose registers that are the registers available to the programmer.

4.1.2 The ALU

- The arithmetic logic unit (ALU) carries out the logic operations (such as comparisons) and arithmetic operations (such as add or multiply) required during the program execution.
- Generally, an ALU has two data inputs and one data output.
- Operations performed in the ALU often affect bits in the status register (bits are set to indicate actions such as whether an overflow has occurred).
- The ALU knows which operations to perform because it is controlled by signals from the control unit.

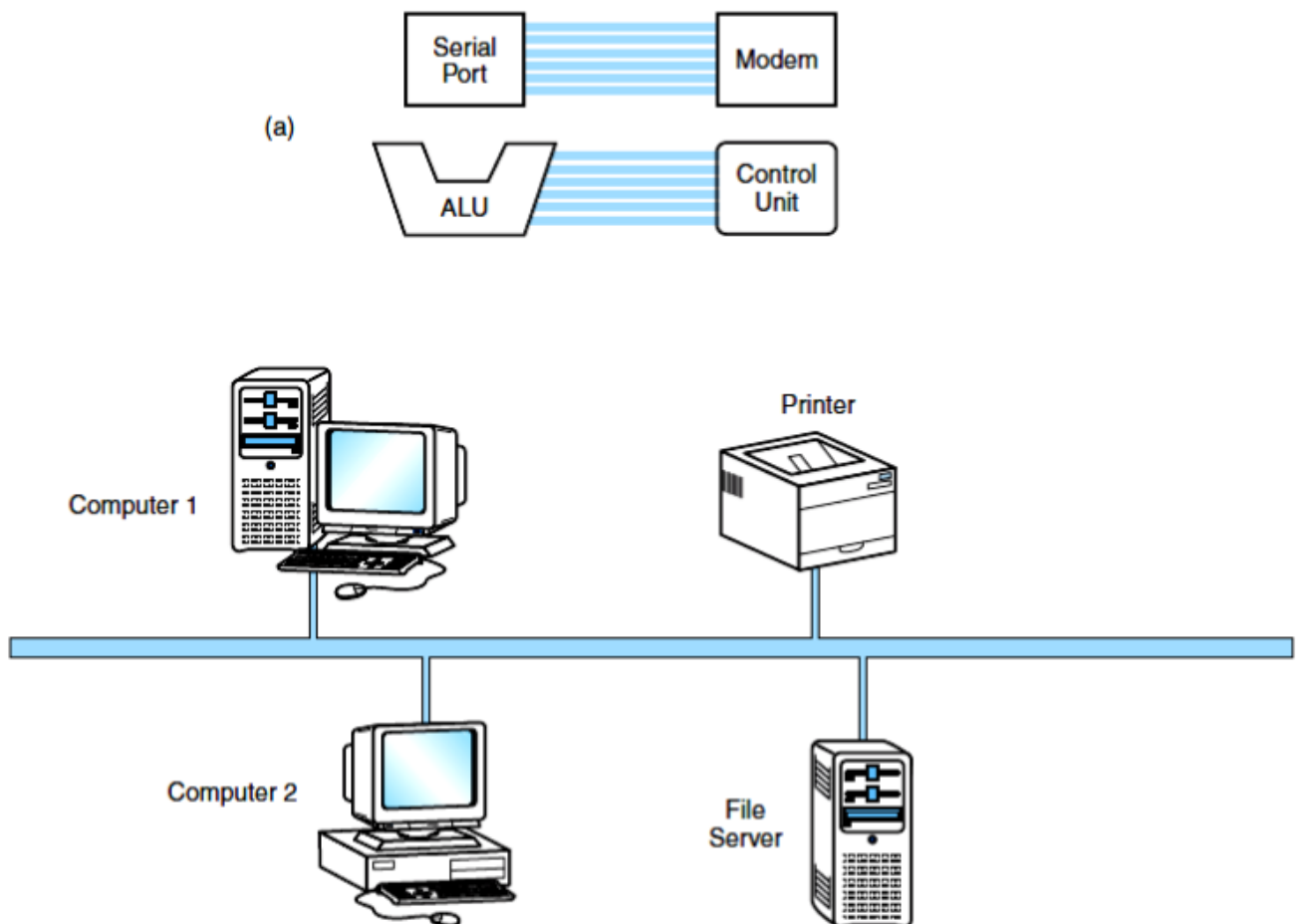
4.1.3 The Control Unit

- The control unit is the "policeman" or "traffic manager" of the CPU.
- It monitors the execution of all instructions and the transfer of all information.
- The control unit extracts instructions from memory, decodes these instructions, making sure data is in the right place at the right time, tells the ALU which registers to use, services interrupts, and turns on the correct circuitry in the ALU for the execution of the desired operation.
- The control unit uses a program counter register to find the next instruction for execution and a status register to keep track of overflows, carries, borrows, and the like.

4.2 The Bus

- The CPU communicates with the other components via a bus.
- A bus is a set of wires that acts as a shared but common data path to connect multiple subsystems within the system.
- It consists of multiple lines, allowing the parallel movement of bits.
- Buses are low cost but very versatile, and they make it easy to connect new devices to each other and to the system.

- At any one time, only one device (be it a register, the ALU, memory, or some other component) may use the bus.
- However, this sharing often results in a communications bottleneck.
- The speed of the bus is affected by its **length** as well as by the **number of devices sharing it**.
- Quite often, devices are divided into master and slave categories, where a master device is one that initiates actions and a slave is one that responds to requests by a master.
- A bus can be **point-to-point**, connecting two specific components (as seen in Figure 3.1a) or it can be a **common pathway** that connects a number of devices, requiring these devices to share the bus (referred to as a multipoint bus and shown in Figure 3.1b).



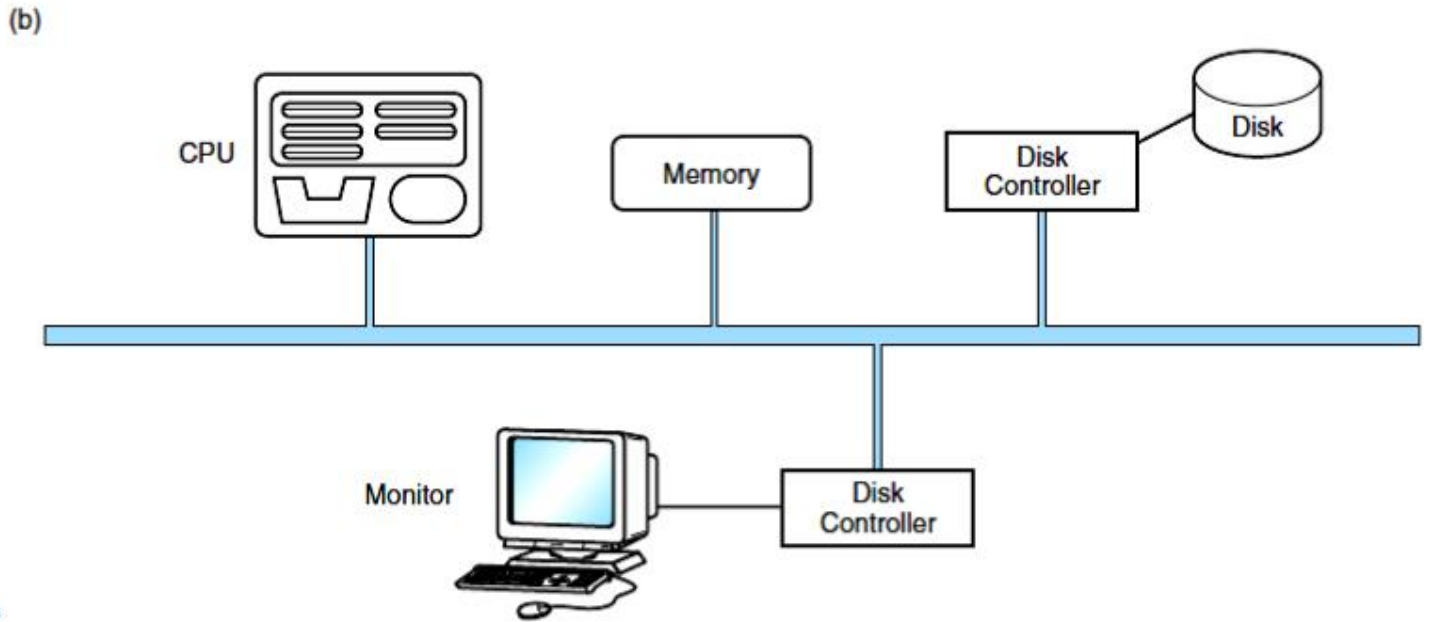


Fig.(3-1) a) Point to Point Buses b) A Multi-Point Buses

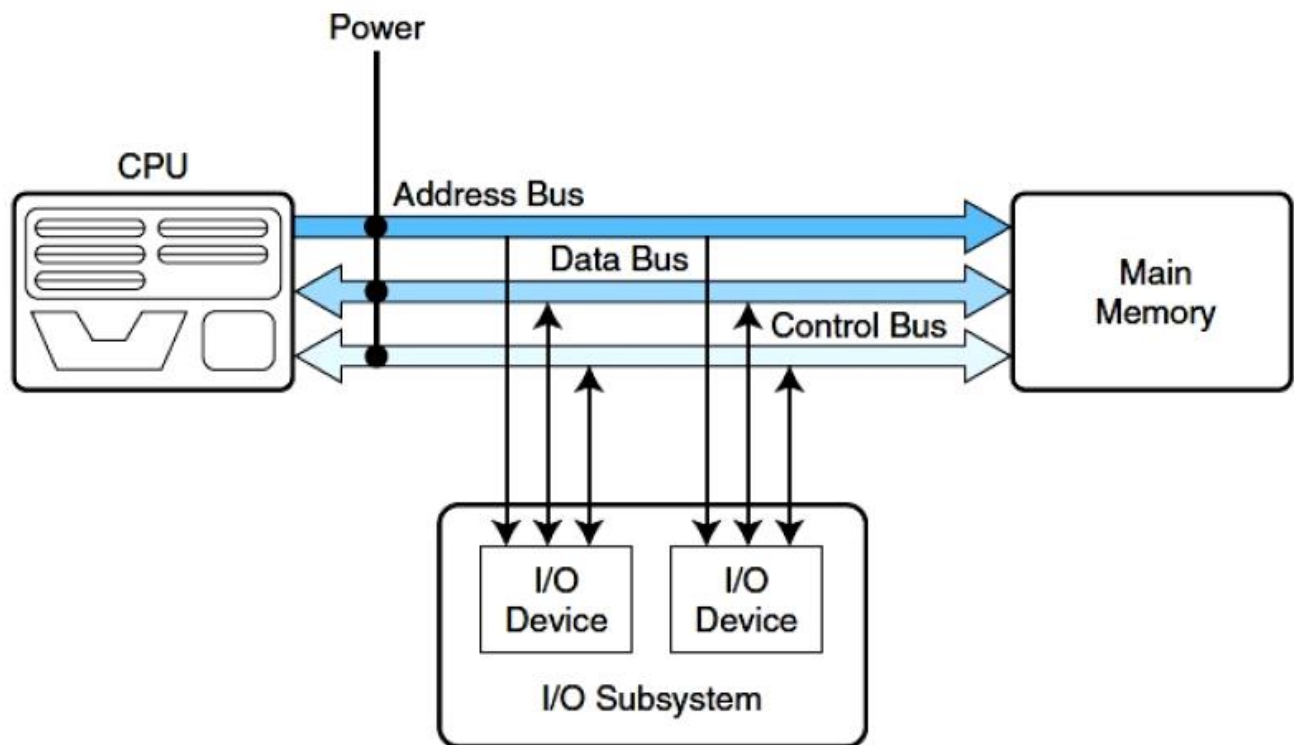


FIGURE 3.2 The Components of a Typical Bus

- Because of this sharing, the bus protocol (set of usage rules) is very important.
- Figure 3.2 shows a typical bus consisting of data lines, address lines, control lines, and power lines.
- Often the lines of a bus dedicated to moving data are called the data bus.
- **Data lines:** Contain the actual information that must be moved from one location to another.
- **Control lines:** Indicate which device has permission to use the bus and for what purpose (reading or writing from memory or from an I/O device, for example).
- Control lines also transfer acknowledgments for bus requests, clock synchronization signals, and interrupts.
- **Address lines:** Indicate the location (in memory, for example) that the data should be either read from or written to.
- **The power lines** provide the electrical power necessary.
- Typical bus transactions include sending an address (for a read or write), transferring data from memory to a register (a memory read), and transferring data to the memory from a register (a memory write).
- In addition, buses are used for I/O reads and writes from peripheral devices.
- Each type of transfer occurs within a bus cycle, the time between two ticks of the bus clock.

4.3 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.
- As the clock ticks, it sets the pace for everything that happens in the system.

- The CPU uses this clock to regulate its progress, checking the otherwise unpredictable speed of the digital logic gates.
- The CPU requires a fixed number of clock ticks to execute each instruction.
- Therefore, instruction performance is often measured in clock cycles—the time between clock ticks—instead of seconds.
- The clock frequency (sometimes called the clock rate or clock speed) is measured in MHz.
- The clock cycle time (or clock period) is simply the reciprocal of the clock frequency.
- For example, an 800MHz machine has a clock cycle time of $1/800,000,000$ or 1.25ns.
- If a machine has a 2ns cycle time, then it is a 500MHz machine.

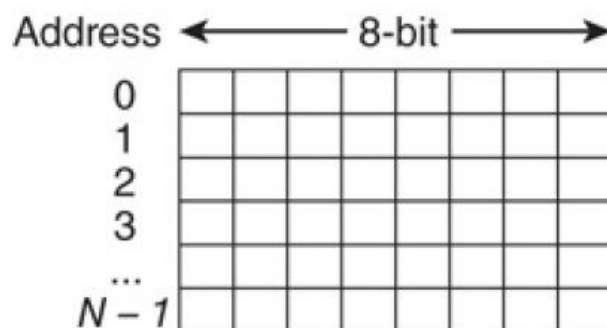
4.4 The Input / Output Subsystem

- Input and output (I/O) devices allow us to communicate with the computer system.
- I/O is the transfer of data between primary memory and various I/O peripherals.
- Input devices such as keyboards, mice, card readers, scanners, voice recognition systems, and touch screens allow us to enter data into the computer.
- Output devices such as monitors, printers, plotters, and speakers allow us to get information from the computer.
- These devices are not connected directly to the CPU. Instead, there is an interface that handles the data transfers.
- This interface converts the system bus signals to and from a format that is acceptable to the given device.

- The CPU communicates to these external devices via input/output registers. This exchange of data is performed in two ways.
- In memorymapped I/O, the registers in the interface appear in the computer's memory map and there is no real difference between accessing memory and accessing an I/O device. Clearly, this is advantageous from the perspective of speed, but it uses up memory space in the system.
- With instruction-based I/O, the CPU has specialized instructions that perform the input and output. Although this does not use memory space, it requires specific I/O instructions, which implies it can be used only by CPUs that can execute these specific instructions.
- Interrupts play a very important part in I/O, because they are an efficient way to notify the CPU that input or output is available for use.

4.5 MEMORY ORGANIZATION

- You can envision memory as a matrix of bits. Each row, implemented by a register, has a length typically equivalent to the addressable unit size of the machine.
- Each register (more commonly referred to as a memory location) has a unique address; memory addresses usually start at zero and progress upward. Figure 4.4 illustrates this concept.



(a)

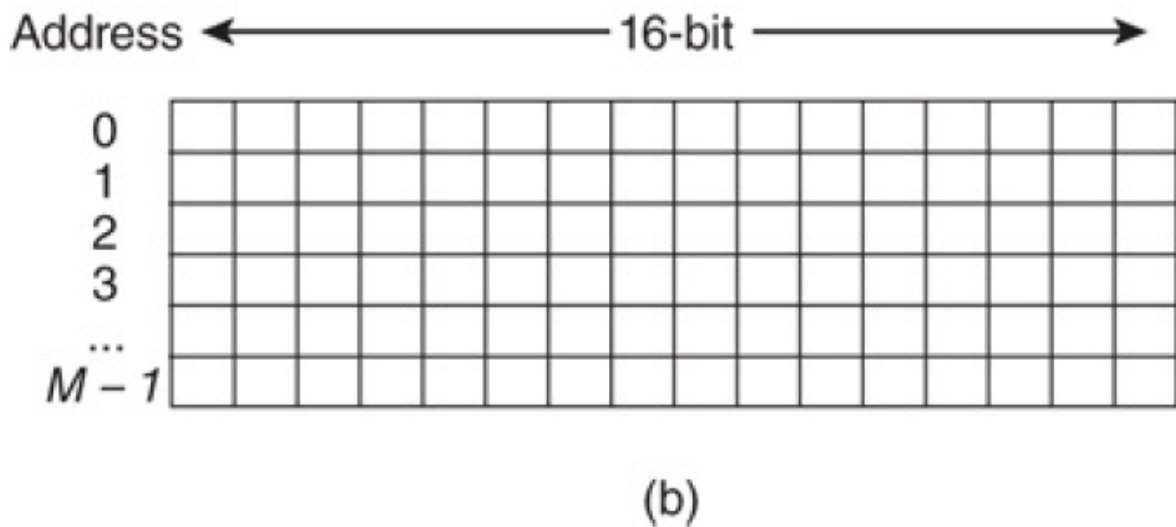


FIGURE 4.4 (a) N 8-Bit Memory Locations
(b) M 16-Bit Memory Locations

- An address is typically represented by an unsigned integer.
- Normally, memory is **byte addressable**, which means that each individual byte has a unique address.
- Some machines may have a word size that is larger than a single byte.
- For example, a computer might handle 32-bit words (which means it can manipulate 32 bits at a time through various instructions and it uses 32-bit registers) but still employ a byte-addressable architecture.
- In this situation, when a word uses multiple bytes, the byte with the lowest address determines the address of the entire word.

4.6 MARIE Architecture

- MARIE, a Machine Architecture that is Really Intuitive and Easy, is a simple architecture consisting of memory (to store programs and data) and a CPU (consisting of an ALU and several registers).

- It has all the functional components necessary to be a real working computer.
- We describe MARIE's architecture in the following sections.

4.6.1 The Architecture.

- MARIE has the following characteristics:
 - Binary, two's complement.
 - Stored program, fixed word length.
 - Word (but not byte) addressable.
 - 4K words of main memory (this implies 12 bits per address).
 - 16-bit data (words have 16 bits).
 - 16-bit instructions, 4 for the opcode and 12 for the address.
 - A 16-bit accumulator (AC).
 - A 16-bit instruction registers (IR).
 - A 16-bit memory buffer register (MBR).
 - A 12-bit program counter (PC).
 - A 12-bit memory address register (MAR).
 - An 8-bit input register.
 - An 8-bit output register.
- We emphasize again that each location in memory has a unique address (represented in binary) and each location can hold a value.
- These notions of the address versus what is actually stored at that address tend to be confusing.
- To help avoid confusion, just visualize a post office.
- There are post office boxes with various "addresses" or numbers. Inside the post office box, there is mail. To get the mail, the number of the post office box must be known. The same is true for data or instructions that need to be fetched from memory.

- The contents of any memory address are manipulated by specifying the address of that memory location.
- We shall see that there are many different ways to specify this address.

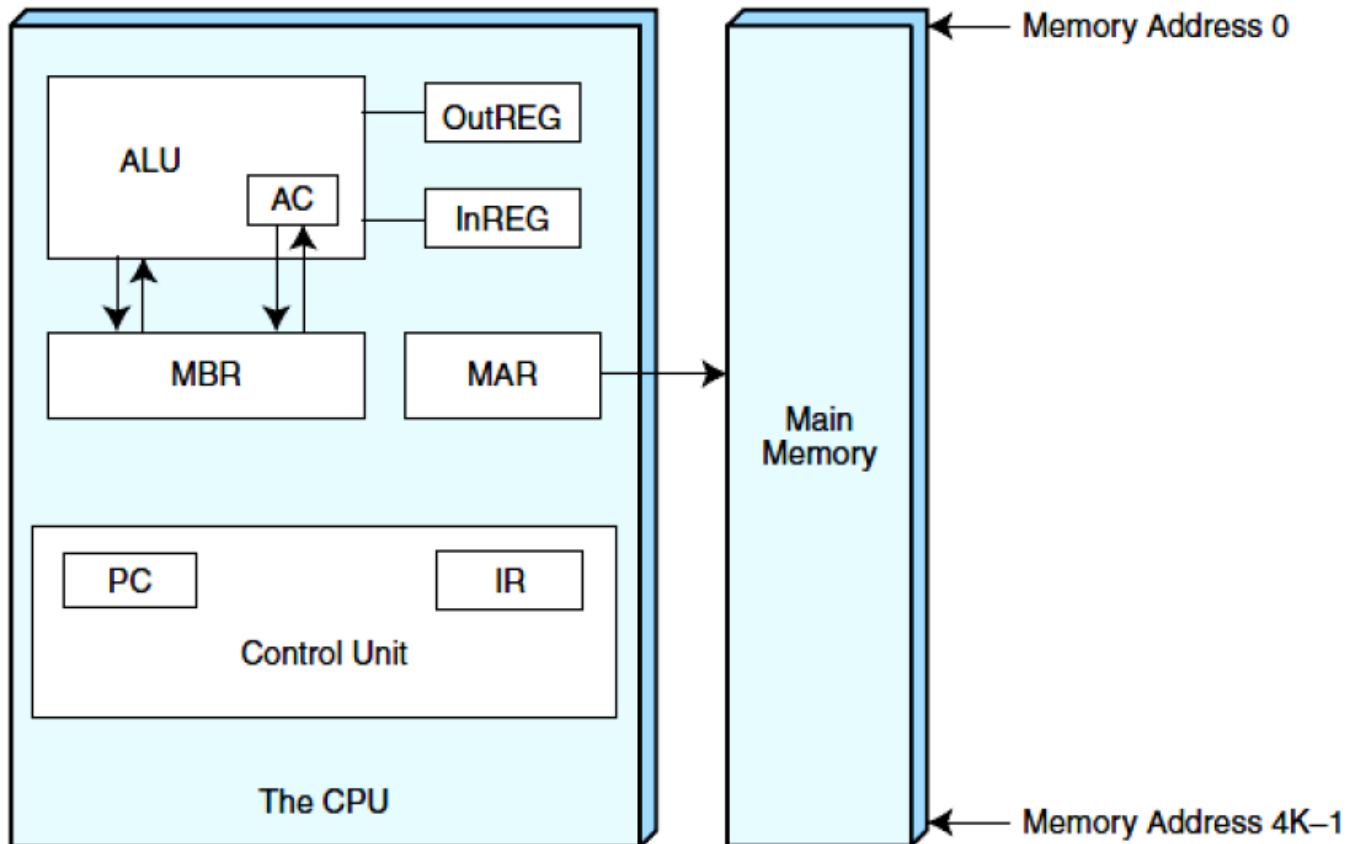


Figure 3.3 shows the architecture for MARIE.

4.6.2 Registers and Buses

- In MARIE, there are seven registers, as follows:
- **AC: The accumulator**, which holds data values. This is a general purpose register and holds data that the CPU needs to process.

- **MAR: The memory address register**, which holds the memory address of the data being referenced.
- **MBR: The memory buffer register**, which holds either the data just read from memory or the data ready to be written to memory.
- **PC: The program counter**, which holds the address of the next instruction to be executed in the program.
- **IR: The instruction register**, which holds the next instruction to be executed.
- **InREG: The input register**, which holds data from the input device.
- **OutREG: The output register**, which holds data for the output device.

4.6.3 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:

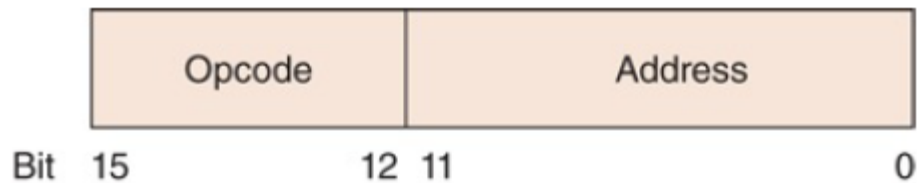
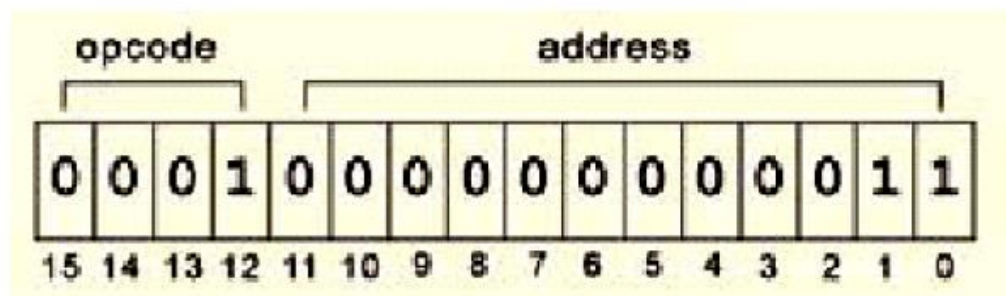


FIGURE 4.10 MARIE's Instruction Format

- 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:

