4.1 Introduction

• In this chapter, we first look at a very simple computer called MARIE: A Machine Architecture that is Really Intuitive and Easy.
• We then provide brief overviews of Intel and MIPS machines, two popular architectures reflecting the CISC (Complex Instruction Set Computer) and RISC (Reduced Instruction Set Computer) design philosophies.
• The objective of this chapter is to give you an understanding of how a computer functions.

4.2 CPU Basics and Organization

• The Central processing unit (CPU) is responsible for fetching program instructions, decoding each instruction that is fetched, and executing the indicated sequence of operations on the correct data.
• The two principal parts of the CPU are the datapath and the control unit.
• The datapath consists of an arithmetic-logic unit (ALU) and storage units (registers) that are interconnected by a data bus that is also connected to main memory.
• Various CPU components perform sequenced operations according to signals provided by its control unit.

• Registers hold data that can be readily accessed by the CPU.
• They can be implemented using D flip-flops. A 32-bit register requires 32 D flip-flops.
• The arithmetic-logic unit (ALU) carries out logical and arithmetic operations as directed by the control unit.
• The control unit determines which actions to carry out according to the values in a program counter register and a status register.

4.3 The Bus

• The CPU shares data with other system components by way of a data bus.
• A bus is a set of wires that simultaneously convey a single bit along each line.
• Two types of buses are commonly found in computer systems: point-to-point, and multipoint buses.

![Diagram showing point-to-point and multipoint buses.](image)

**FIGURE 4.1** (a) Point-to-Point Buses; (b) A Multipoint Bus
• At any one time, **only one** device (be it a register, the ALU, memory, or some other component) may use the bus.
• However, the sharing often results in a communications bottleneck.

• Master device is one that **initiates** actions and a slave **responds** to requests by a master.

• Buses consist of **data** lines, **control** lines, and **address** lines.
• While the data lines convey bits from one device to another, **control lines** determine the **direction** of data flow, and **when** each device can access the bus.
• Address lines determine the location of the source or destination of the data.

![Diagram of a bus system](image)

**FIGURE 4.2** The Components of a Typical Bus

• In a master-slave configuration, where more than one device can be the bus master, concurrent bus master requests must be arbitrated.

• Four categories of bus arbitration are:
  o Daisy chain: Permissions are passed from the highest **priority** device to the lowest.
  o Centralized parallel: Each device is directly connected to an arbitration circuit, and a **centralized arbiter** selects who gets the bus.
  o Distributed using self-detection: Devices decide which gets the bus among **themselves**.
  o Distributed using collision-detection: Any device can try to use the bus. If its data collides with the data of another device, the device tries again (Ethernet uses this type arbitration).
4.4 Clocks

- Every computer contains at least one clock that synchronizes the activities of its components.
- A fixed number of clock cycles are required to carry out each data movement or computational operation.
- The clock frequency, measured in megahertz or gigahertz, determines the speed with which all operations are carried out.
- Clock cycle time is the reciprocal of clock frequency.
  - An 800 MHz clock has a cycle time of 1.25 ns.
- The minimum clock cycle time must be at least as great as the maximum propagation delay of the circuit.
- The CPU time required to run a program is given by the general performance equation:

\[
\text{CPU Time} = \frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{avg. cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}
\]

- We see that we can improve CPU throughput when we reduce the number of instructions in a program, reduce the number of cycles per instruction, or reduce the number of nanoseconds per clock cycle.
- In general, multiplication requires more time than addition, floating point operations require more cycles than integer ones, and accessing memory takes longer than accessing registers.
- Bus clocks are usually slower than CPU clocks, causing bottleneck problems.

4.5 The Input/Output Subsystem

- I/O devices allow us to communicate with the computer system. A computer communicates with the outside world through its input/output (I/O) subsystem.
- I/O is the transfer of data between primary memory and various I/O peripherals.
- I/O devices are not connected directly to the CPU. I/O devices connect to the CPU through various interfaces.
- The CPU communicates to these external devices via input/output registers.
- This exchange of data is performed in two ways:
  - In memory-mapped I/O, the registers in the interface appear in the computer’s memory and there is no real difference accessing memory and accessing an I/O device. It uses up memory space in the system.
  - With instruction-based I/O, the CPU has specialized instructions that input and output. Although this does not use memory space, it requires specific I/O instructions.
- Interrupts play a very important part in I/O, because they are an efficient way to notify CPU that input or output is available for use.
4.6 Memory Organization and Addressing

- You can envision memory as a matrix of bits.
- Each row, implemented by a register, has a length typically equivalent to the word size of 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](image)

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

- Normally, memory is byte-addressable, which means that each individual byte has a unique address.
- For example, a computer might handle 32-bit word, 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.
- It is also possible that a computer might be word-addressable, which means each word has its own address, but most current machines are byte-addressable.
- If architecture is byte-addressable, and the instruction set architecture word is larger than 1 byte, the issue of alignment must be addressed.
- Memory is built from random access memory (RAM) chips. Memory is often referred to using the notation L X W (length X Length). For example,
  - 4M X 16 means the memory is 4M long (4M = 2^20 X 2^4 = 2^24 words) and it is 16 bits wide (each word is 16 bits).
  - To address this memory (assuming word addressing), we need to be able to uniquely identify 2^24 different items.
  - The memory locations for this memory are numbered 0 through 2^24 -1.
  - The memory bus of this system requires at least 22 address lines.
- In general, if a computer 2^n addressable units of memory, it will require N bits to uniquely address each byte.
• Physical memory usually consists of more than one RAM chip.

![Memory as a Collection of RAM Chips (32K X 16)](image)

• Access is more efficient when memory is organized into banks of chips with the addresses interleaved across the chips:
  - Accordingly, in **high-order** interleaving, the **high** order address bits specify the memory bank.

![High-Order Memory Interleaving (4bytes X 8chips)](image)

- With **low-order** interleaving, the **low** order bits of the address specify which memory bank contains the address of interest.

![Low-Order Memory Interleaving (4bytes X 8chips)](image)
4.7 Interrupts

- Interrupts are events that alter (or interrupt) the normal flow of execution in the system. An interrupt can be triggered for a variety of reasons, including:
  - I/O requests
  - Arithmetic errors (e.g., division by zero)
  - Arithmetic underflow or overflow
  - Hardware malfunction (e.g., memory parity error)
  - User-defined break points (such as when debugging a program)
  - Page faults (this is covered in detail in Chapter 6)
  - Invalid instructions (usually resulting from pointer issues)
  - Miscellaneous
- Each interrupt is associated with a procedure that directs the actions of the CPU when an interrupt occurs.

4.8 MARIE

- MARIE: a Machine Architecture that is Really Intuitive and Easy, is a simple architecture consisting of memory (to store program 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.

The Architecture

- MARIE has the following characteristics:
  - Binary, two's complement data representation.
  - Stored program, fixed word length data and instructions.
  - 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 register (IR)
  - A 16-bit memory buffer register (MBR)
  - A 12-bit program counter (PC)
  - A 12-bit memory address register (MAR)
  - A 8-bit input register
  - A 8-bit output register
FIGURE 4.8 MARIE’s Architecture

 Registers and Buses

- In MARIE, there are seven registers, as follows:
  o AC: The accumulator, which holds **data** values. This is a general purpose register and holds data that the CPU needs to process.
  o MAR: The memory address register, which holds the memory **address** of the data being referenced.
  o MBR: The memory buffer register, which holds either the **data** just read from memory or the data ready to be written to memory.
  o PC: The program counter, which holds the **address** of the **next** instruction to be executed in the program.
  o IR: The instruction register, which holds the **next** instruction to be executed.
  o InREG: The input register, which holds data from the input device.
  o OutREG: The output register, which holds data from the output device.

- In MARIE, we assume a common bus scheme.
The Instruction Set Architecture

- 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 thirteen instructions.
## Register Transfer Notation

- Each of our instructions actually consists of a sequence of smaller instructions called **microoperations**.
- The symbolic notation used to describe the behavior of microoperations is called register transfer notation (RTN) or register transfer language (RTL).
- In the MARIE RTL, we use the notation $M[X]$ to indicate the actual data value stored in memory location $X$, and $\leftarrow$ to indicate the transfer of bytes to a register or memory location.
- For example:
  - The RTL for the LOAD instruction is:
    
    $\begin{align*}
    &\text{MAR} \leftarrow X \\
    &\text{MBR} \leftarrow M[\text{MAR}], \text{AC} \leftarrow \text{MBR}
    \end{align*}$
  - The RTL for the ADD instruction is:
    
    $\begin{align*}
    &\text{MAR} \leftarrow X \\
    &\text{MBR} \leftarrow M[\text{MAR}] \\
    &\text{AC} \leftarrow \text{AC} + \text{MBR}
    \end{align*}$
  - SKIPCOND skips the next instruction according to the value of the AC.
    
    If $\text{IR}[11 - 10] = 00$ then
    If $\text{AC} < 0$ then $\text{PC} \leftarrow \text{PC} + 1$
    else If $\text{IR}[11 - 10] = 01$ then
      If $\text{AC} = 0$ then $\text{PC} \leftarrow \text{PC} + 1$
else If IR[11 - 10] = 11 then
    If AC > 0 then PC ← PC + 1
Instruction Processing

• All computers follow a basic machine cycle: the fetch, decode, and execute cycle.

The Fetch-Decode-Execute Cycle

• The fetch-decode-execute cycle represents the steps that a computer follows to run a program.

FIGURE 4.11 The Fetch-Decode-Execute Cycle
Interrupts and I/O

- When the CPU executes an input or output instruction, the appropriate I/O device is notified.
- The CPU then continues with other useful work until the device is ready.
- At that time, the device sends an interrupt signal to the CPU.
- The CPU then processes the interrupt, after which it continues with normal fetch-decode-execute cycle.

![Diagram of Interrupt Cycle]

FIGURE 4.12 Modified Instruction Cycle to Check for Interrupt

4.9 A Simple Program

- Consider the simple MARIE program given in TABLE 4.3. We show a set of mnemonic instructions stored at addresses 100 - 106 (hex):

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Binary Contents of Memory Address</th>
<th>Hex Contents of Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Load 104</td>
<td>00010001000000100</td>
<td>1104</td>
</tr>
<tr>
<td>101</td>
<td>Add 105</td>
<td>0011000100000101</td>
<td>3105</td>
</tr>
<tr>
<td>102</td>
<td>Store 106</td>
<td>0100000100000110</td>
<td>4106</td>
</tr>
<tr>
<td>103</td>
<td>Halt</td>
<td>0111000000000000</td>
<td>7000</td>
</tr>
<tr>
<td>104</td>
<td>0023</td>
<td>00000000000100011</td>
<td>0023</td>
</tr>
<tr>
<td>105</td>
<td>FFE9</td>
<td>11111111111101001</td>
<td>FFE9</td>
</tr>
<tr>
<td>106</td>
<td>0000</td>
<td>000000000000000000</td>
<td>0000</td>
</tr>
</tbody>
</table>

TABLE 4.3 A Program to Add Two Numbers
A Discussion on Assemblers

- Mnemonic instructions, such as LOAD 104, are easy for humans to write and understand.

What Do Assemblers Do?

- An assembler’s job is to convert assembly language (using mnemonics) into machine language (which consists entirely of binary values, or string of zeros and ones).
- In assembly language, there is a one-to-one correspondence between a mnemonic instruction and its machine code.
- The assembler reads a source file (assembly program) and produces an object file (the machine code).
- 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.

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Load X</td>
</tr>
<tr>
<td>101</td>
<td>Add Y</td>
</tr>
<tr>
<td>102</td>
<td>Store Z</td>
</tr>
<tr>
<td>103</td>
<td>Halt</td>
</tr>
<tr>
<td>X, 104</td>
<td>DEC 35</td>
</tr>
<tr>
<td>Y, 105</td>
<td>DEC -23</td>
</tr>
<tr>
<td>Z, 106</td>
<td>HEX 0000</td>
</tr>
</tbody>
</table>

TABLE 4.5 An Example Using Directives for Constants

During the first pass, we have a symbol table and the partial instructions

- During the second pass, the instructions are completed using the values from the symbol table.

Why Use Assembly Language?

- Most programmers agree that 10% of the code in a program uses approximately 90% of the CPU time.
- In time-critical applications, we often need to optimize this 10% of code. Programmers can make the program more efficient in terms of time (and space).
- If the overall size of the program or response time is critical, assembly language often becomes the language of choice.
• **Embedded Systems** must be reactive and often are found in time-constrained environments. These systems are designed to perform either a single instruction or a very specific set of instructions.

**Chapter Summary**

• The major components of a computer system are its control unit, registers, memory, ALU, and data path.
• MARIE has 4K 16-bit words of main memory, uses 16-bit instructions, and has seven registers.
• There is only one general purpose register, the AC.
• Instructions for MARIE use 4 bits for the opcode and 12 bits for an address.
• A built-in clock keeps everything synchronized.
• Computers run programs through iterative **fetch-decode-execute cycles**.
• Computers can run programs that are in machine language.
• An assembler converts mnemonic code to machine language.