CHAPTER 4

MARIE: An Introduction to a Simple Computer

4.1 Introduction 219
4.2 CPU Basics and Organization 219
   4.2.1 The Registers 220
   4.2.2 The ALU 221
   4.2.3 The Control Unit 221
4.3 The Bus 221
4.4 Clocks 225
4.5 The Input/Output Subsystem 227
4.6 Memory Organization and Addressing 227
4.7 Interrupts 235
4.8 MARIE 236
   4.8.1 The Architecture 236
   4.8.2 Registers and Buses 236
   4.8.3 Instruction Set Architecture 238
   4.8.4 Register Transfer Notation 242
4.9 Instruction Processing 244
   4.9.1 The Fetch–Decode–Execute Cycle 244
   4.9.2 Interrupts and the Instruction Cycle 246
   4.9.3 MARIE’s I/O 249
4.10 A Simple Program 249
4.11 A Discussion on Assemblers 252
   4.11.1 What Do Assemblers Do? 252
   4.11.2 Why Use Assembly Language? 254
4.12 Extending Our Instruction Set 255
4.13 A Discussion on Decoding: Hardwired Versus Microprogrammed Control 262
   4.13.1 Machine Control 262
   4.13.2 Hardwired Control 265
   4.13.3 Microprogrammed Control 270
4.14 Real-World Examples of Computer Architectures 274
   4.14.1 Intel Architectures 275
   4.14.2 MIPS Architectures 282
Chapter Summary 284
4.1 Introduction 219

- 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 219

- 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.
- All computers have a CPU that can be divide into two pieces: 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. Check Figure 1.5 The von Neumann Architecture.
- The control unit is a module responsible for sequencing operations and making sure the correct data are where they need to be at the correct time.
- Various CPU components perform sequenced operations according to signals provided by its control unit.

![Diagram of the von Neumann Architecture](image)

FIGURE 1.5 The von Neumann Architecture
4.2.1 The Registers 220

- 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.

4.2.2 The ALU 221

- The arithmetic-logic unit (ALU) carries out logical operations (such as comparisons) and arithmetic operations (such as add or multiply) directed by the control unit.

4.2.3 The Control Unit 221

- 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 221

- 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 of a computer system with buses and components]

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 Typical Bus](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 arbitration: Permissions are passed from the highest **priority** device to the lowest.
  o Centralized parallel arbitration: Each device is directly connected to an arbitration circuit, and a **centralized arbiter** selects who gets the bus.
  o Distributed arbitration using self-detection: Devices decide which gets the bus among **themselves**.
  o Distributed arbitration 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 225

- 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 227

- 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 227

- 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.

![Memory Matrix Diagram]

- 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) 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 2^22 different items.
  - The memory locations for this memory are numbered 0 through 2^22 - 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.

![Figure 4.5](image)

- Memory is $32K = 2^5 \times 2^{10} = 2^{15}$
- 15 bits are needed for each address.
- We need 4 bits to select the chip, and 11 bits for the offset into the chip that selects the byte.

• Access is more efficient when memory is organized into banks of chips with the addresses interleaved across the chips.

• Suppose we have a byte-addressable memory consisting of 8 modules of 4 bytes each, for a total of 32 bytes memory:
  - Accordingly, in **high-order** interleaving, the high order address bits specify the memory bank.

![Figure 4.6](image)
With **low-order** interleaving, the **low** order bits of the address specify which memory bank contains the address of interest.

In Figure 4.5 (32K X 8 memory), consider the address **001 0000 0010 0111**

- In high-order interleaving the high-order, we need **4 leftmost** bits to determine the chip, and **11** remaining bits as the offset. The address **001 0000 0010 0111** is stored on chip **2 (0010)** at offset **39 (000 0010 0111)**

- In low-order interleaving, we need **4 rightmost** bits to determine the chip. The address **001 0000 0010 0111** is stored on chip **7 (0111)** at offset **258 (001 0000 0010)**
4.7 Interrupts 235

- 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 236

- 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.

4.8.1 The Architecture 236

- 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

![MARIE's Architecture](image)
4.8.2 Registers and Buses 236

- 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 referred.
  - 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 from the output device.
- In MARIE, we assume a common bus scheme.

![The Data Path in MARIE](image)

FIGURE 4.9 The Data Path in MARIE
4.8.3 Instruction Set Architecture 238

- 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.

FIGURE 4.10 MARIE’s Instruction Format

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

TABLE 4.2 MARIE’s Instruction Set
4.8.4 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:
  - **Load X**
    The RTL for the LOAD instruction is:
    
    \[
    \begin{align*}
    \text{MAR} & \leftarrow X \\
    \text{MBR} & \leftarrow M[\text{MAR}] \\
    \text{AC} & \leftarrow \text{MBR}
    \end{align*}
    \]
    
    We see that the opcode is 1 and the address from which to load the data is 3.
  - **Add X**
    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 bits 11 and 10 in the IR are both 0)
    
    If \( \text{AC} < 0 \) then \( \text{PC} \leftarrow \text{PC} + 1 \)
    
    else If \( \text{IR}[11 - 10] = 01 \) then (if bit 11 = 0 and bit 10 = 1)
    
    If \( \text{AC} = 0 \) then \( \text{PC} \leftarrow \text{PC} + 1 \)
    
    else If \( \text{IR}[11 - 10] = 10 \) then (if bit 11 = 1 and bit 10 = 0)
    
    If \( \text{AC} > 0 \) then \( \text{PC} \leftarrow \text{PC} + 1 \)
4.9 Instruction Processing 244

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

4.9.1 The Fetch–Decode–Execute Cycle 244

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

![Diagram of the Fetch-Decode-Execute Cycle](image-url)
4.9.2 Interrupts and the Instruction Cycle 246

- All computers provide a means for the normal fetch-decode-execute cycle to be interrupted.
- These interruptions may be necessary for many reasons, including:
  - A program error (such as division by 0, arithmetic overflow, stack overflow, or attempting to access a protected area of memory)
  - A hardware error (such as a memory parity error or power failure)
  - An I/O completion (which happens when a disk read is requested and the data transfer is complete)
  - A user interrupt (such as hitting Ctrl-C or Ctrl-Break to stop a program)
  - An interrupt from a timer set by the operating system (such as is necessary when allocating virtual memory or performing certain bookkeeping functions)

- 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](image)

4.9.3 MARIE’s I/O 249

- MARIE has two registers to handle input and output. All output is placed in an output register, OutREG, and the CPU polls the input register, InREG, until input is sensed, at which time the value is copied into the accumulator.
4.10 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>0001000100000100</td>
<td>1104</td>
</tr>
<tr>
<td>101</td>
<td>Add 105</td>
<td>0011000100000010</td>
<td>3105</td>
</tr>
<tr>
<td>102</td>
<td>Store 106</td>
<td>0100000100000011</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>00000000000000000</td>
<td>0000</td>
</tr>
</tbody>
</table>

### TABLE 4.3 A Program to Add Two Numbers

a) Load 104

<table>
<thead>
<tr>
<th>Step</th>
<th>RTN</th>
<th>PC</th>
<th>IR</th>
<th>MAR</th>
<th>MBR</th>
<th>AC</th>
</tr>
</thead>
<tbody>
<tr>
<td>(initial values)</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fetch</td>
<td>MAR → PC</td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IR ← M[MAR]</td>
<td>100</td>
<td>1104</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PC ← PC + 1</td>
<td>101</td>
<td>1104</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decode</td>
<td>MAR ← IR[11–0]</td>
<td>101</td>
<td>1104</td>
<td>104</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Decode IR[15–12])</td>
<td>101</td>
<td>1104</td>
<td>104</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get operand</td>
<td>MBR ← M[MAR]</td>
<td>101</td>
<td>1104</td>
<td>104</td>
<td>0023</td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td>AC ← AC + MBR</td>
<td>101</td>
<td>1104</td>
<td>104</td>
<td>0023</td>
<td>0023</td>
</tr>
</tbody>
</table>

b) Add 105

<table>
<thead>
<tr>
<th>Step</th>
<th>RTN</th>
<th>PC</th>
<th>IR</th>
<th>MAR</th>
<th>MBR</th>
<th>AC</th>
</tr>
</thead>
<tbody>
<tr>
<td>(initial values)</td>
<td>101</td>
<td>1104</td>
<td>104</td>
<td>0023</td>
<td>0022</td>
<td></td>
</tr>
<tr>
<td>Fetch</td>
<td>MAR → PC</td>
<td>101</td>
<td>1104</td>
<td>101</td>
<td>0023</td>
<td>0023</td>
</tr>
<tr>
<td></td>
<td>IR ← M[MAR]</td>
<td>101</td>
<td>3105</td>
<td>101</td>
<td>0023</td>
<td>0023</td>
</tr>
<tr>
<td></td>
<td>PC ← PC + 1</td>
<td>102</td>
<td>3105</td>
<td>101</td>
<td>0023</td>
<td>0023</td>
</tr>
<tr>
<td>Decode</td>
<td>MAR ← IR[11–0]</td>
<td>102</td>
<td>3105</td>
<td>105</td>
<td>0023</td>
<td>0023</td>
</tr>
<tr>
<td></td>
<td>(Decode IR[15–12])</td>
<td>102</td>
<td>3105</td>
<td>105</td>
<td>0023</td>
<td>0023</td>
</tr>
<tr>
<td>Get operand</td>
<td>MBR ← M[MAR]</td>
<td>102</td>
<td>3105</td>
<td>105</td>
<td>FFE9</td>
<td>0023</td>
</tr>
<tr>
<td>Execute</td>
<td>AC ← AC + MBR</td>
<td>102</td>
<td>3105</td>
<td>105</td>
<td>FFE9</td>
<td>0000</td>
</tr>
</tbody>
</table>

c) Store 106

<table>
<thead>
<tr>
<th>Step</th>
<th>RTN</th>
<th>PC</th>
<th>IR</th>
<th>MAR</th>
<th>MBR</th>
<th>AC</th>
</tr>
</thead>
<tbody>
<tr>
<td>(initial values)</td>
<td>102</td>
<td>3105</td>
<td>105</td>
<td>FFE9</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>Fetch</td>
<td>MAR → PC</td>
<td>102</td>
<td>3105</td>
<td>102</td>
<td>FFE9</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>IR ← M[MAR]</td>
<td>102</td>
<td>2106</td>
<td>102</td>
<td>FFE9</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>PC ← PC + 1</td>
<td>103</td>
<td>2106</td>
<td>102</td>
<td>FFE9</td>
<td>0000</td>
</tr>
<tr>
<td>Decode</td>
<td>MAR ← IR[11–0]</td>
<td>103</td>
<td>2106</td>
<td>105</td>
<td>FFE9</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>(Decode IR[15–12])</td>
<td>103</td>
<td>2106</td>
<td>105</td>
<td>FFE9</td>
<td>0000</td>
</tr>
<tr>
<td>Get operand</td>
<td>(not necessary)</td>
<td>103</td>
<td>2106</td>
<td>105</td>
<td>FFE9</td>
<td>0000</td>
</tr>
<tr>
<td>Execute</td>
<td>MBR ← AC</td>
<td>103</td>
<td>2106</td>
<td>106</td>
<td>000C</td>
<td>0000</td>
</tr>
</tbody>
</table>

FIGURE 4.14 A Trace of the Program to Add Two Numbers
4.11 A Discussion on Assemblers 252

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

4.11.1 What Do Assemblers Do? 252

- 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 is 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

- The first pass of the assembler creates a symbol table and the partially-assembled instructions as shown.

```
X  104
Y  105
Z  106
X  1
Y  3
Z  2
HEX 7000
```

CMPS375 Class Notes (Chap04)    Page 19 / 27    Dr. Kuo-pao Yang
On the second pass, the assembler uses the symbol tale to fill in the addresses and create the corresponding machine language instruction.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>E</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

4.11.2 Why Use Assembly Language? 254

- 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.
- Chance are you use some type of embedded system every day:
  - Consumer electronics (such as cameras, camcorders, cellular phones, PDAs, and interactive games)
  - Consumer products (such as washers, microwave ovens, and washing machines)
  - Automobiles (particularly engine control and antilock brakes)
  - Medical instruments (such as CAT scanners and heart monitors)
  - Industry (for process controllers and avionics)
4.12 Extending Our Instruction Set 255

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>RTN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Jmp X</td>
<td>HBR ← PC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAR ← X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>H[MAR] ← MBR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MBR ← X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AC ← 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AC ← AC + MBR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PC ← AC</td>
</tr>
<tr>
<td>0001</td>
<td>Load X</td>
<td>MAR ← X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MBR ← H[MAR], AC ← MBR</td>
</tr>
<tr>
<td>0010</td>
<td>Store X</td>
<td>MAR ← X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>H[MAR] ← AC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MBR ← MBR</td>
</tr>
<tr>
<td>0011</td>
<td>Add X</td>
<td>MAR ← X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MBR ← H[MAR]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AC ← AC + MBR</td>
</tr>
<tr>
<td>0100</td>
<td>Subt X</td>
<td>MAR ← X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MBR ← H[MAR]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AC ← AC - MBR</td>
</tr>
<tr>
<td>0101</td>
<td>Input</td>
<td>AC ← InREG</td>
</tr>
<tr>
<td>0110</td>
<td>Output</td>
<td>OutREG ← AC</td>
</tr>
<tr>
<td>0111</td>
<td>Halt</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>Skpcond</td>
<td>If IR[11-10] = 00 then</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If AC &lt; 0 then PC ← PC + 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Else If IR[11-10] = 01 then</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If AC &gt; 0 then PC ← PC + 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Else If IR[11-10] = 10 then</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If AC &gt; 0 then PC ← PC + 1</td>
</tr>
<tr>
<td>1001</td>
<td>Jump X</td>
<td>PC ← IR[11-0]</td>
</tr>
<tr>
<td>1010</td>
<td>Clear</td>
<td>AC ← 0</td>
</tr>
<tr>
<td>1011</td>
<td>AddI X</td>
<td>MAR ← X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MBR ← H[MAR]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAR ← MBR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MBR ← H[MAR]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AC ← AC + MBR</td>
</tr>
<tr>
<td>1100</td>
<td>JumpI X</td>
<td>MAR ← X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MBR ← H[MAR]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PC ← MBR</td>
</tr>
</tbody>
</table>

**TABLE 4.7 MARIE’s Full Instruction Set**

- **EXAMPE 4.2** Here is an example using a loop to add five numbers
- **EXAMPE 4.3** This example illustrates the use of an if/else construct to allow for selection. In particular, it implements the following:
  
  if X = Y then
  
  X := X * 2
  
  else
  
  Y := Y - X;

- **EXAMPE 4.5** This example illustrates the use of a simple subroutine to double the value stored at X.
4.13 A Discussion on Decoding: Hardwired Versus Microprogrammed Control 262

- A computer’s control unit keeps things synchronized, making sure that bits flow to the correct components as the components are needed.
- There are two general ways in which a control unit can be implemented: hardwired control and microprogrammed control.
  - With **microprogrammed control**, a small program is placed into read-only memory in the microcontroller.
  - **Hardwired controllers** implement this program using digital logic components.

4.13.1 Machine Control 262

- P0, P1, P2, controls reading from memory or a register, and the other set consisting of P3, P4, P5, controls writing to memory or a register.

![FIGURE 4.15 Connection of MARIE’s MBR to the Datapath](image-url)
4.13.2 Hardwired Control 265

- The advantage of hardwired control is that it is very **fast**.
- The disadvantage is that the instruction set and the control logic are directly tied together by special circuits that are complex and difficult to design or **modify**.
- For example, a 4-to-16 decoder could be used to decode the opcode. By using the contents of the IR register and the status of the ALU, this controls the registers, the ALU operations, all shifters, and bus access.

![Diagram of Hardwired Control Unit]

**FIGURE 4.17 Hardwired Control Unit**
- Careful inspection of MARIE’s RTL reveals that the ALU has only three operations: add, subtract, and clear.
  - We will also define a fourth “do nothing” state.
- The entire set of MARIE’s control signals consists of:
  - Register controls: \( P_0 \) through \( P_5 \) (\( P_0P_1P_2 \) for read and \( P_3P_4P_5 \) for write)
  - ALU controls: \( A_0 \) through \( A_1 \) (do nothing, add, subtract, and clear)
  - Timing: \( T_0 \) through \( T_7 \) and counter reset \( C_r \)
- In FIGURE 4.9, Main Memory (address 0 or 111), MAR (address 1 or 001), PC (address 2 or 010), MBR (address 3 or 011), AC (address 4 or 100), InREG (address 5 or 101), OutREG (address or 110), and IR (address 7 or 111).
- Here is the complete signal sequence for MARIE’s Add instruction:

\[
\begin{align*}
\text{P}_0\text{P}_1\text{P}_2\text{T}_0: & \quad \text{MAR} \leftarrow X \quad // \text{X is in the rightmost 12 bits of the IR; OpCode 0011 (Add)} \\
& \quad // \text{T}_0 \\
& \quad // \text{P}_0\text{P}_1\text{P}_2: \text{Read from IR (address 7 or 111)} \\
& \quad // \text{P}_3: \text{Write to MAR (address 1 or 001)} \\
\text{P}_3\text{T}_1: & \quad \text{MBR} \leftarrow \text{M}[\text{MAR}] \quad // \text{T}_1 \\
& \quad // \text{Read from Mem (address 0 or 000)} \\
& \quad // \text{P}_3\text{P}_4: \text{Write to MBR (address 3 or 011)} \\
\text{A}_0\text{P}_3\text{P}_5\text{T}_2: & \quad \text{AC} \leftarrow \text{AC} + \text{MBR} \quad // \text{T}_2 \\
& \quad // \text{P}_5\text{P}_1: \text{Read from MBR (address 3 or 011)} \\
& \quad // \text{P}_3: \text{Write to AC (address 4 or 100)} \\
& \quad // \text{A}_0: \text{ALU Add (address 1 or 01)} \\
\text{C}_r\text{T}_3: & \quad \text{[Reset counter]}
\end{align*}
\]

- These signals are ANDed with combinational logic to bring about the desired machine behavior.

![FIGURE 4.20 Combinational Logic for Signal Controls of MARIE’s Add Instruction](image-url)
4.13.3 Microprogrammed Control 270

- The other approach called microprogramming, uses software for control illustrated in Figure 4.21.
- All machine instructions are input into a special program, the microprogram, to convert the instruction into the appropriate control signals.
- The microprogram is essentially an interpreter, written in microcode, that is stored in firmware (ROM, PROM, EPROM), which is often referred to as the control store.
- Essentially there is one subroutine in this program for each machine instruction.
- The advantage of this approach is that if the instruction set requires modification the microprogram is simply updated to match – no change is required in the actual hardware.
- The disadvantage to this approach is that all instructions must go through an additional level of interpretation, slowing down the program execution.

FIGURE 4.21 Microprogrammed Control Unit
4.14 Real-World Examples of Computer Architectures 274

- Each member of the x86 family of Intel architectures is known as a CISC (Complex Instruction Set Computer) machine, whereas the Pentium family and the MIPS architectures are examples of RISC (Reduced Instruction Set Computer) machines.
- The main objective of RISC machines is to simplify instructions so they can execute more quickly. Each instruction performs only one operation; they are all the same size.

4.14.1 Intel Architectures 275

- The classic Intel architecture, the 8086, was born in 1979. It is a CISC architecture.
- It was adopted by IBM for its famed PC, which was released in 1981.
- The 8086 operated on 16-bit data words and supported 20-bit memory addresses.
- Later, to lower costs, the 8-bit 8088 was introduced. Like the 8086, it used 20-bit memory addresses.
- In 1985, Intel introduced the 32-bit 80386.
- It also had no built-in floating-point unit.
- The 80486, introduced in 1989, was an 80386 that had built-in floating-point processing and cache memory.
- The 80386 and 80486 offered downward compatibility with the 8086 and 8088.
- Software written for the smaller word systems was directed to use the lower 16 bits of the 32-bit registers.
- Intel’s most advanced 32-bit microprocessor is the Pentium 4.
- It can run as fast as 3.06 GHz. This clock rate is over 350 times faster than that of the 8086.
- The introduction of the Itanium processor in 2001 marked Intel’s first 64-bit chip (IA-64). This processor has four integer units, two floating-point units, a significant amount of cache memory at four different levels, 128 floating-point registers, 128 integer registers, and multiple miscellaneous registers.
- Speed enhancing features include multilevel cache and instruction pipelining.
- Intel, along with many others, is marrying many of the ideas of RISC architectures with microprocessors that are largely CISC.

4.14.2 MIPS Architectures 282

- The MIPS family of CPUs has been one of the most successful in its class.
- In 1986 the first MIPS CPU was announced.
- It had a 32-bit word size and could address 4GB of memory.
- Over the years, MIPS processors have been used in general purpose computers as well as in games.
- The MIPS architecture now offers 32- and 64-bit versions.
- MIPS was one of the first RISC microprocessors.
• The original MIPS architecture had only 55 different instructions, as compared with the 8086 which had over 100.
• MIPS Assembler and Runtime Simulator (MARS) is a Java-based MIPS R2000 and R3000 simulator designed especially for undergraduate education.
• SPIM is a popular simulator widely used by students and professional.
• MIPS was designed with performance in mind: It is a load/store architecture, meaning that only the load and store instructions can access memory.
• The large number of registers in the MIPS architecture keeps bus traffic to a minimum.

**Chapter Summary 284**

• 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.
• Control units can be microprogrammed or hardwired.
• Hardwired control units give better performance, while microprogrammed units are more adaptable to changes.
• The Intel architecture is an example of a **CISC** architecture; MIPS is an example of a **RISC** architecture.