CHAPTER 5

Computer Organization

Objectives

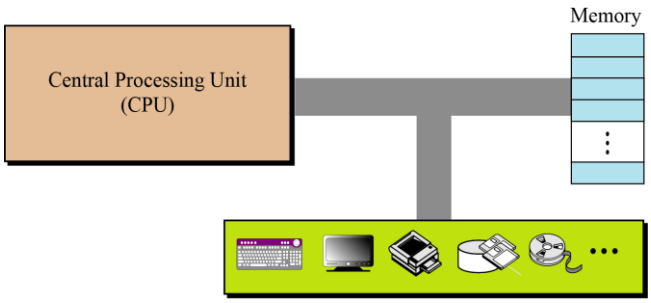
After studying this chapter, the student should be able to:

- ❑ **List the three subsystems of a computer.**
- ❑ **Describe the role of the central processing unit (CPU).**
- ❑ **Describe the fetch-decode-execute phases of a cycle.**
- ❑ **Describe the main memory and its addressing space.**
- ❑ **Define the input/output subsystem.**
- ❑ **Understand the interconnection of subsystems.**
- ❑ **Describe different methods of input/output addressing.**
- ❑ **Distinguish the two major trends in the design of computers.**
- ❑ **Understand how computer throughput can be improved using pipelining and parallel processing.**

5-1 INTRODUCTION

We can divide the parts that make up a computer into three broad categories or subsystem: **the central processing unit (CPU), the main memory**, and **the input/output subsystem.**

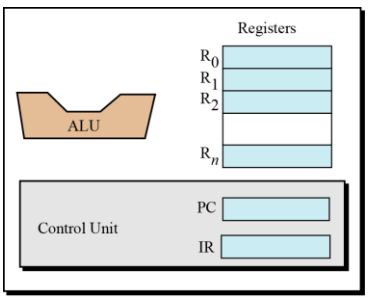
Figure 5.1 **Computer hardware (subsystems)**



5-2 CENTRAL PROCESSING UNIT

The **central processing unit (CPU)** performs operations on data. In most architectures it has three parts: an **arithmetic logic unit (ALU),** a **control unit**, and a set of **registers**, fast storage locations.

Figure 5.2 **Central processing unit (CPU)**



5.2.1 The arithmetic logic unit (ALU)

The central processing unit (CPU) performs operations on data. In most architectures it has three parts: an arithmetic logic unit (ALU), a control unit, and a set of registers, fast storage locations (Figure 5.2).

5.2.2 Registers

Registers are fast stand-alone storage locations that hold data temporarily. Multiple registers are needed to facilitate the operation of the CPU. Some of these registers are shown in Figure 5.2.

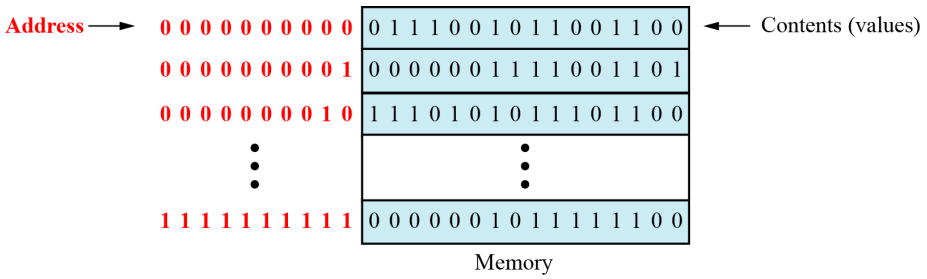
5.2.3 The control unit

The third part of any CPU is the control unit. The control unit controls the operation of each subsystem. Controlling is achieved through signals sent from the control unit to other subsystems.

5-3 MAIN MEMORY

Main memory is the second major subsystem in a computer (Figure 5.3). It consists of a collection of storage locations, each with a unique identifier, called an **address**. Data is transferred to and from memory in groups of bits called **words**. A word can be a group of 8 bits, 16 bits, 32 bits, or 64 bits (and growing). If the word is 8 bits, it is referred to as a **byte**. The term "byte" is so common in computer science that sometimes a 16-bit word is referred to as a 2-byte word, or a 32-bit word is referred to as a 4-byte word.

Figure 5.3 Main memory



5.3.1 Address space

To access a word in memory requires an identifier. Although programmers use a name to identify a word (or a collection of words), at the hardware level each word is identified by an address. The total number of uniquely identifiable locations in memory is called the **address space**. For example, a memory with 64 kilobytes and a word size of 1 byte has an address space that ranges from 0 to 65,535.

Table 5.1 Memory units

Unit	*Exact Number of Bytes*	*Approximation*
kilobyte	2^{10} (1024) bytes	10^3 bytes
megabyte	2^{20} (1,048,576) bytes	10^6 bytes
gigabyte	2^{30} (1,073,741,824) bytes	10^9 bytes
terabyte	2^{40} bytes	10^{12} bytes

Memory addresses are defined using unsigned binary integers.

Example 5.1

A computer has 32 MB (megabytes) of memory. How many bits are needed to address any single byte in memory?

Solution

The memory address space is 32 MB, or 2^{25} ($2^5 \times 2^{20}$). This means that we need $\log_2 2^{25}$, or **25 bits**, to address each byte.

Example 5.2

A computer has 128 MB of memory. Each word in this computer is eight bytes. How many bits are needed to address any single word in memory?

Solution

The memory address space is 128 MB, which means 2^{27}. However, each word is eight (2^3) bytes, which means that we have 2^{24} words. This means that we need $\log_2 2^{24}$, or **24 bits**, to address each word.

5.3.2 Memory types

Two main types of memory exist: **RAM** and **ROM**.

Random access memory (RAM)

- ❑ Static RAM (SRAM)
- ❑ Dynamic RAM (DRAM)

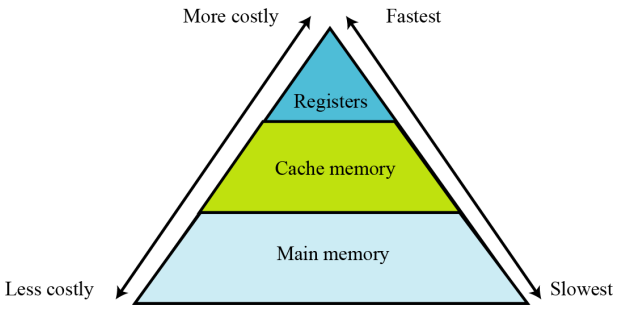
Read-only memory (ROM)

- ❑ Programmable read-only memory (PROM).
- ❑ Erasable programmable read-only memory (EPROM).
- ❑ Electrically erasable programmable read-only memory (EEPROM).

5.3.3 Memory hierarchy

Computer users need a lot of memory, especially memory that is very fast and inexpensive. This demand is not always possible to satisfy—very fast memory is usually not cheap. A compromise needs to be made. The solution is **hierarchical** levels of memory.

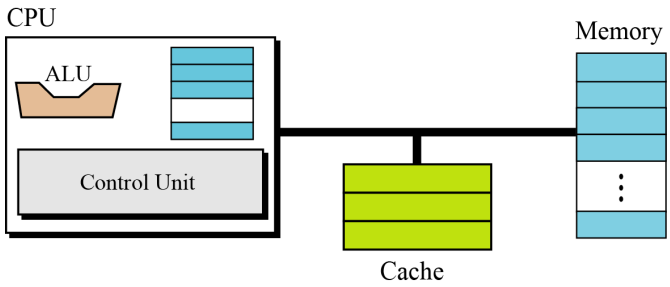
Figure 5.4 **Memory hierarchy**



5.3.4 Cache memory

Cache memory is faster than main memory but slower than the CPU and its registers. Cache memory, which is normally small in size, is placed between the CPU and main memory (Figure 5.5).

Figure 5.5 Cache memory



5-4 INPUT/OUTPUT SUBSYSTEM

The third major subsystem in a computer is the collection of devices referred to as the input/output (I/O) subsystem. This subsystem allows a computer to communicate with the outside world, and to store programs and data even when the power is off. Input/output devices can be divided into two broad categories: **non-storage** and **storage** devices.

5.4.1 Non-storage devices

Non-storage devices allow the CPU/memory to communicate with the outside world, but they cannot store information.

- **Keyboard and monitor**
- **Printer**

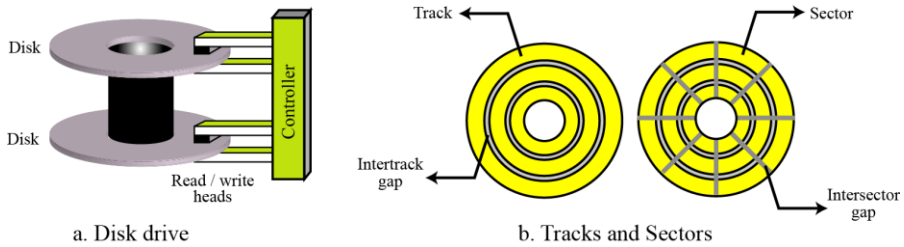
5.17

5.4.2 Storage devices

Storage devices, although classified as I/O devices, can store large amounts of information to be retrieved at a later time. They are cheaper than main memory, and their contents are nonvolatile—that is, not erased when the power is turned off. They are sometimes referred to as auxiliary storage devices. We can categorize them as either **magnetic** or **optical**.

Magnetic storage devices

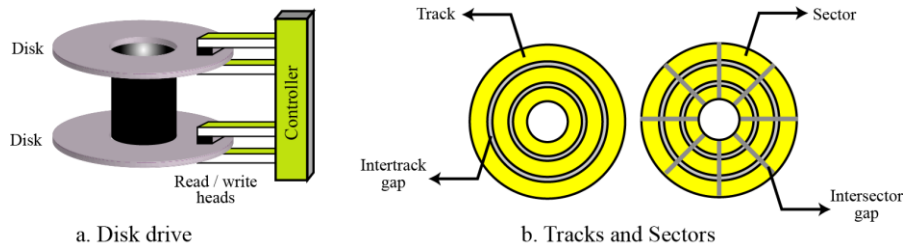
Figure 5.6 A magnetic disk



a. Disk drive

b. Tracks and Sectors

Figure 5.6 A magnetic disk



a. Disk drive

b. Tracks and Sectors

Optical storage devices

Figure 5.8 Creation and use of CD-ROMs

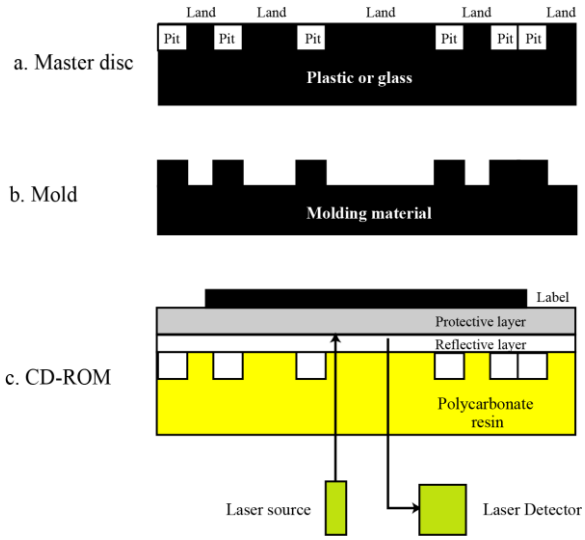
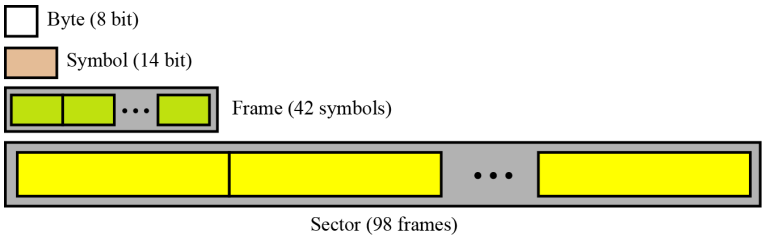


Table 5.2 CD-ROM speeds

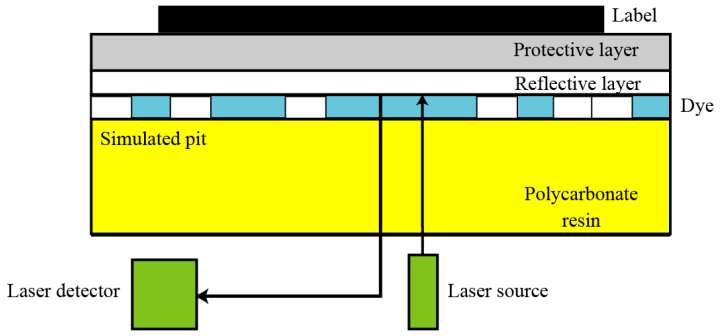
Speed	*Data rate*	*Approximation*
1*x*	153,600 bytes per second	150 KB/s
2*x*	307,200 bytes per second	300 KB/s
4*x*	614,400 bytes per second	600 KB/s
6*x*	921,600 bytes per second	900 KB/s
8*x*	1,228,800 bytes per second	1.2 MB/s
12*x*	1,843,200 bytes per second	1.8 MB/s
16*x*	2,457,600 bytes per second	2.4 MB/s
24*x*	3,688,400 bytes per second	3.6 MB/s
32*x*	4,915,200 bytes per second	4.8 MB/s
40*x*	6,144,000 bytes per second	6 MB/s

Figure 5.9 CD-ROM format



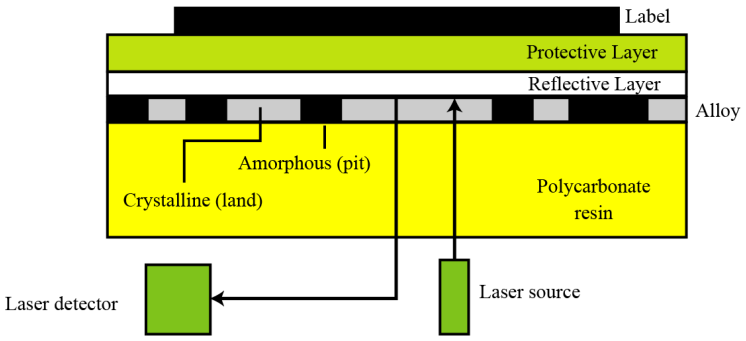
5.23

Figure 5.10 Making a CD-R



5.24

Figure 5.11 Making a CD-RW



5.25

Table 5.3 DVD capacities

Feature	*Capacity*
Single-sided, single-layer	4.7 GB
Single-sided, dual-layer	8.5 GB
Double-sided, single-layer	9.4 GB
Double-sided, dual-layer	17 GB

5.26

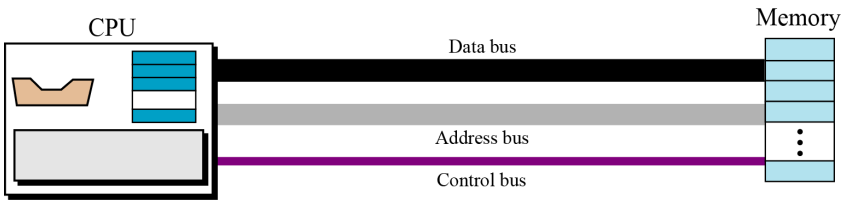
5-5 SUBSYSTEM INTERCONNECTION

The previous sections outlined the characteristics of the three subsystems (CPU, main memory, and I/O) in a stand-alone computer. In this section, we explore how these three subsystems are interconnected. The interconnection plays an important role because information needs to be exchanged between the three subsystems.

5.5.1 Connecting CPU and memory

The CPU and memory are normally connected by three groups of connections, each called a **bus**: *data bus*, *address bus*, and *control bus* (Figure 5.12).

Figure 5.12 **Connecting CPU and memory using three buses**

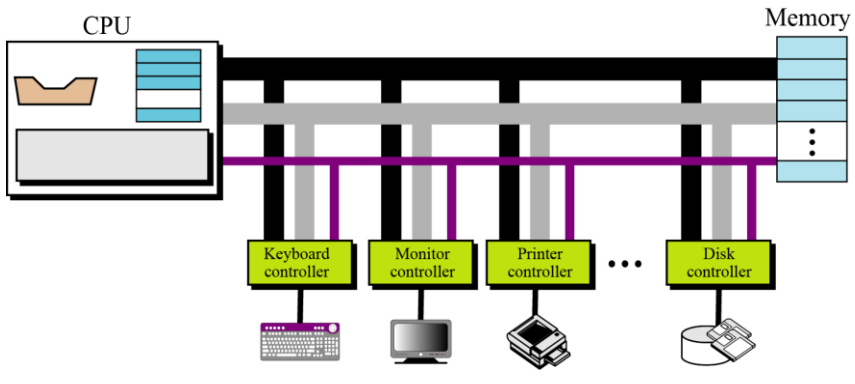


5.5.2 Connecting I/O devices

I/O devices cannot be connected directly to the buses that connect the CPU and memory because the nature of I/O devices is different from the nature of CPU and memory. I/O devices are electromechanical, magnetic, or optical devices, whereas the CPU and memory are electronic devices. I/O devices also operate at a much slower speed than the CPU/memory. There is a need for some sort of intermediary to handle this difference. Input/output devices are therefore attached to the buses through input/output controllers or interfaces. There is one specific controller for each input/output device (Figure 5.13).

5.29

Figure 5.13 Connecting I/O devices to the buses



5.30

Figure 5.14 SCSI controller

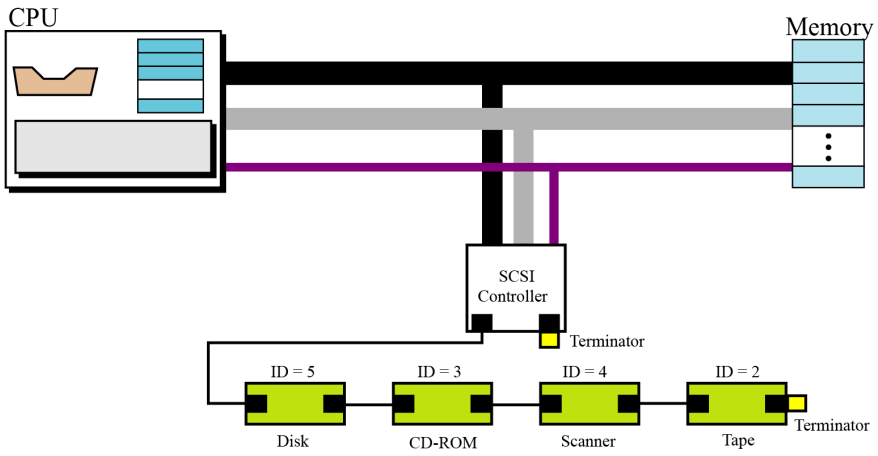


Figure 5.15 FireWire controller

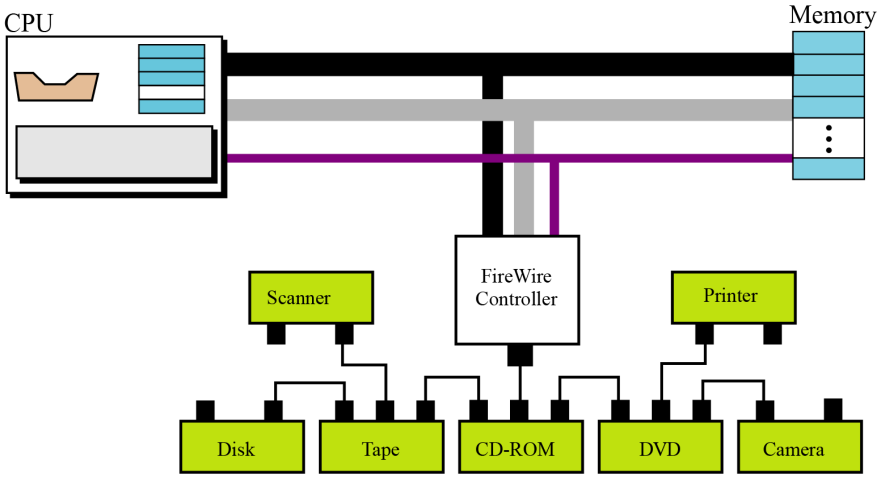
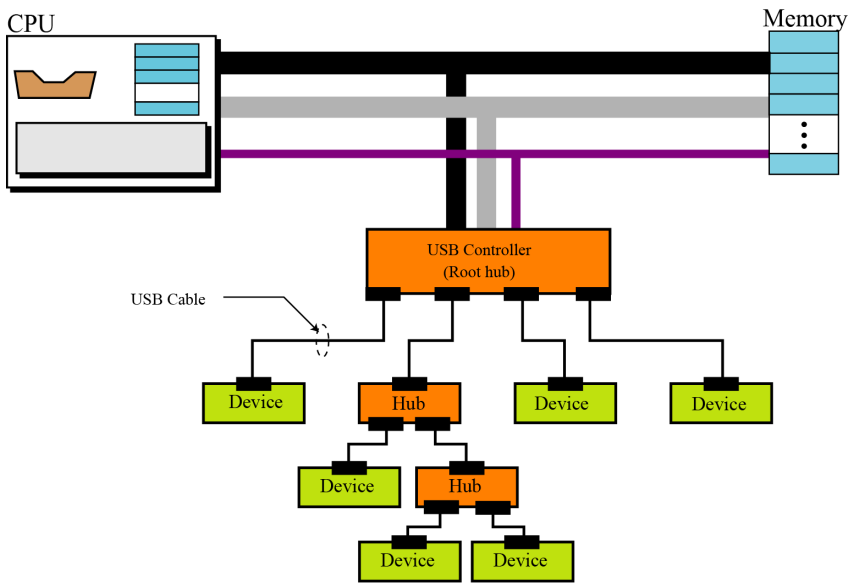


Figure 5.16 USB controller



5.5.3 Addressing input/output devices

The CPU usually uses the same bus to read data from or write data to main memory and I/O device. The only difference is the instruction. If the instruction refers to a word in main memory, data transfer is between main memory and the CPU. If the instruction identifies an I/O device, data transfer is between the I/O device and the CPU. There are two methods for handling the addressing of I/O devices: isolated I/O and memory-mapped I/O.

Figure 5.17 Isolated I/O addressing

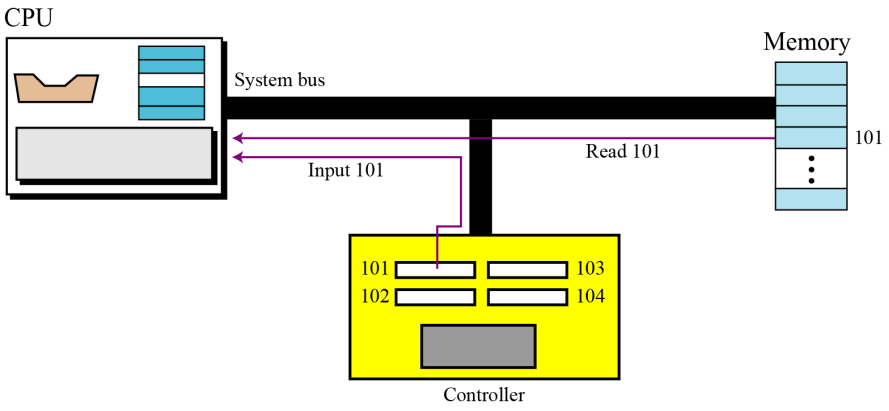
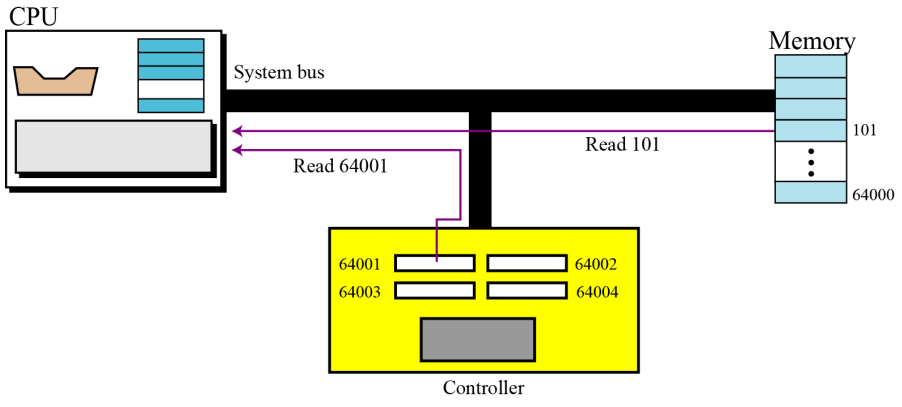


Figure 5.18 Memory-mapped I/O addressing



5-6 PROGRAM EXECUTION

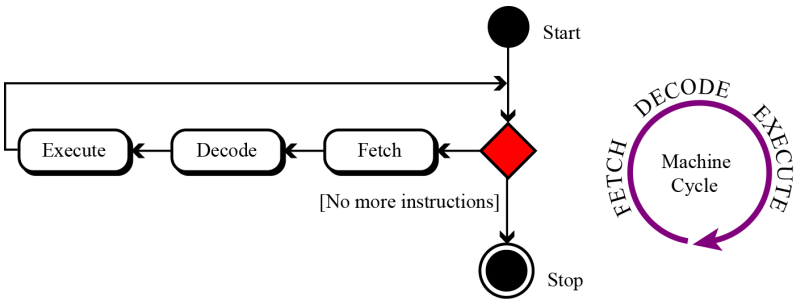
Today, **general-purpose computers** use a set of instructions called a **program** to process data. A computer executes the program to create output data from input data. Both the program and the data are stored in memory.

At the end of this chapter we give some examples of how a hypothetical simple computer executes a program.

5.6.1 Machine cycle

The CPU uses repeating **machine cycles** to execute instructions in the program, one by one, from beginning to end. A simplified cycle can consist of three phases: ***fetch***, ***decode***, and ***execute*** (Figure 5.19).

Figure 5.19 **The steps of a cycle**



5.6.2 Input/output operation

Commands are required to transfer data from I/O devices to the CPU and memory. Because I/O devices operate at much slower speeds than the CPU, the operation of the CPU must be somehow synchronized with the I/O devices. Three methods have been devised for this synchronization: programmed I/O, interrupt driven I/O, and direct memory access (DMA).

- ❑ **Programmed I/O**
- ❑ **Interrupt driven I/O**
- ❑ **Direct memory access (DMA**

5.39

Figure 5.20 Programmed I/O

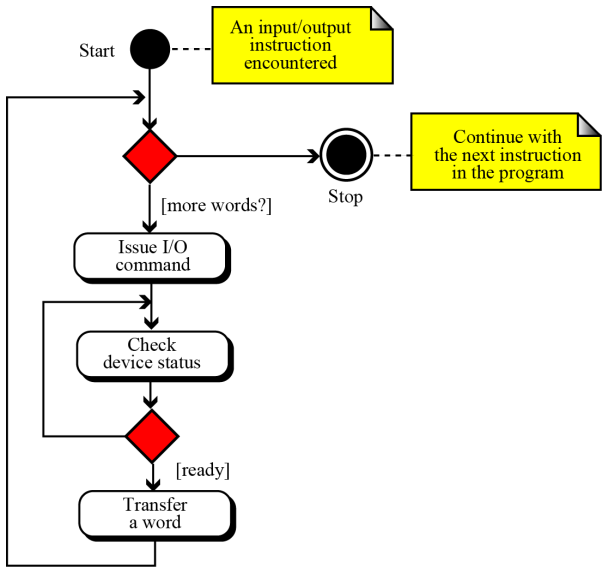


Figure 5.21 Interrupt-driven I/O

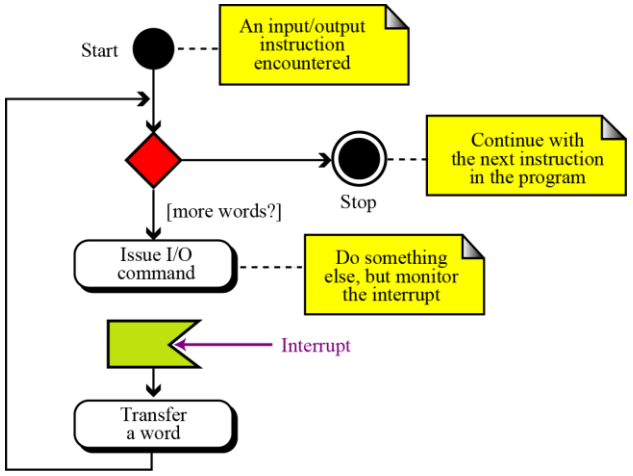


Figure 5.22 DMA connection to the general bus

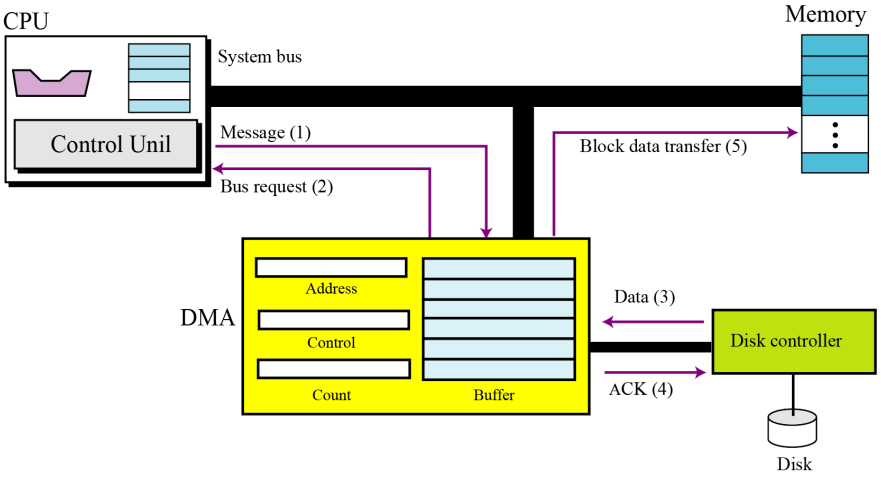
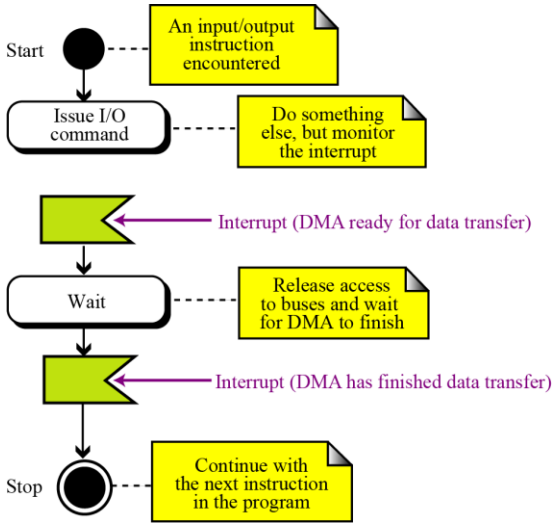


Figure 5.23 DMA input/output



5-7 DIFFERENT ARCHITECTURES

The architecture and organization of computers has gone through many changes in recent decades. In this section we discuss some common architectures and organization that differ from the simple computer architecture we discussed earlier.

5.7.1 CISC

CISC (pronounced sisk) stands for complex instruction set computer (CISC). The strategy behind CISC architectures is to have a large set of instructions, including complex ones. Programming CISC-based computers is easier than in other designs because there is a single instruction for both simple and complex tasks. Programmers therefore do not have to write a set of instructions to do a complex task.

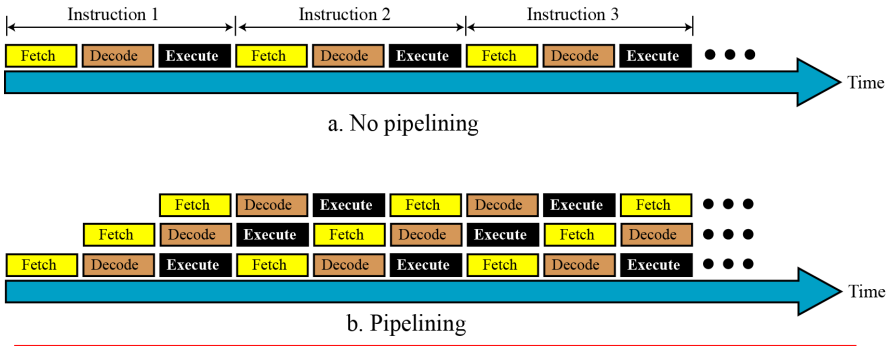
5.7.2 RISC

RISC (pronounced risk) stands for reduced instruction set computer. The strategy behind RISC architecture is to have a small set of instructions that do a minimum number of simple operations. Complex instructions are simulated using a subset of simple instructions. Programming in RISC is more difficult and time-consuming than in the other design because most of the complex instructions are simulated using simple instructions.

5.7.3 Pipelining

We have learned that a computer uses three phases of fetch, decode, and execute for each instruction. In early computers, these three phases needed to be done in series for each instruction. In other words, instruction *n* needs to finish all of these phases before the instruction *n* + 1 can start its own phases. Modern computers use a technique called **pipelining** to improve the throughput (the total number of instructions performed in each period of time). The idea is that if the control unit can do two or three of these phases simultaneously, the next instruction can start before the previous one is finished.

Figure 5.24 **Pipelining**



5.7.4 Parallel processing

Traditionally a computer had a single control unit, a single arithmetic logic unit, and a single memory unit. With the evolution in technology and the drop in the cost of computer hardware, today we can have a single computer with multiple control units, multiple arithmetic logic units and multiple memory units. This idea is referred to as **parallel processing**. Like pipelining, parallel processing can improve throughput.

Figure 5.24 **A taxonomy of computer organization**

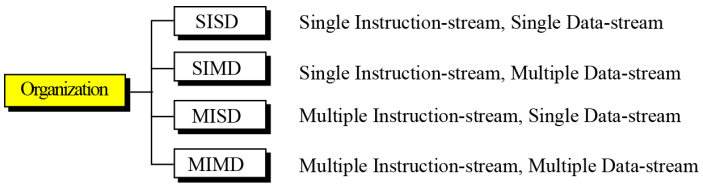


Figure 5.26 **SISD organization**

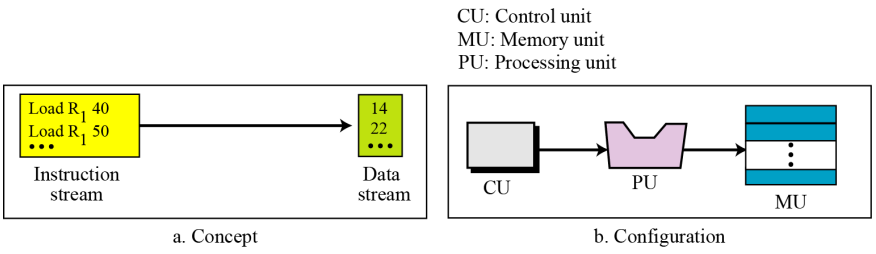


Figure 5.27 SIMD organization

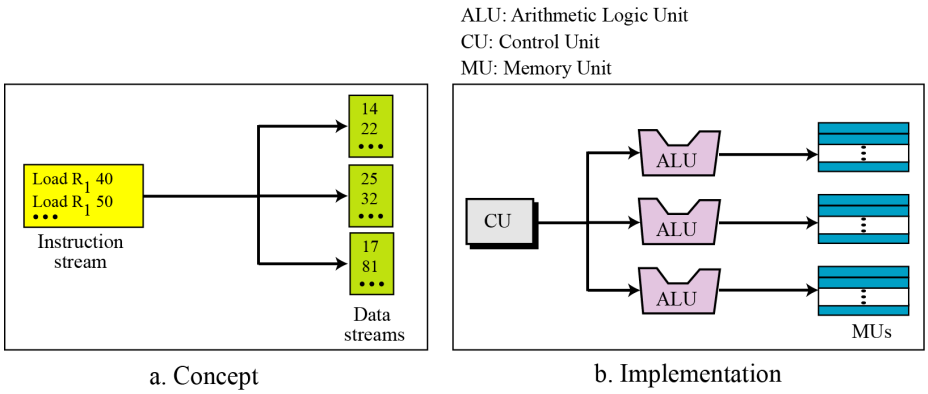


Figure 5.28 MISD organization

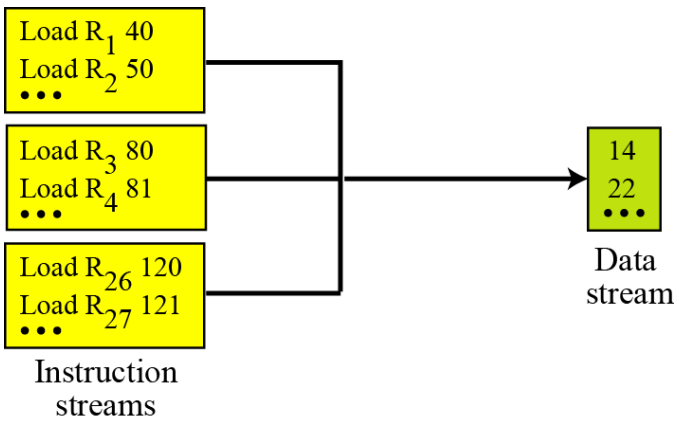
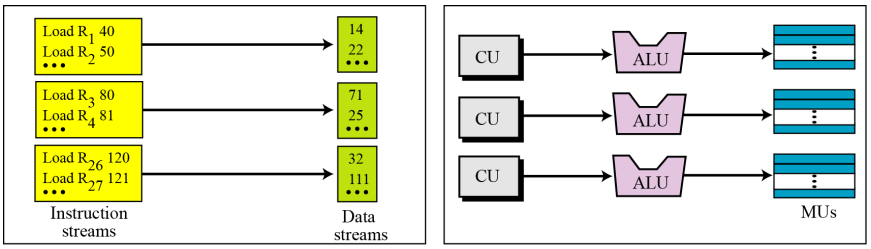


Figure 5.29 MIMD organization

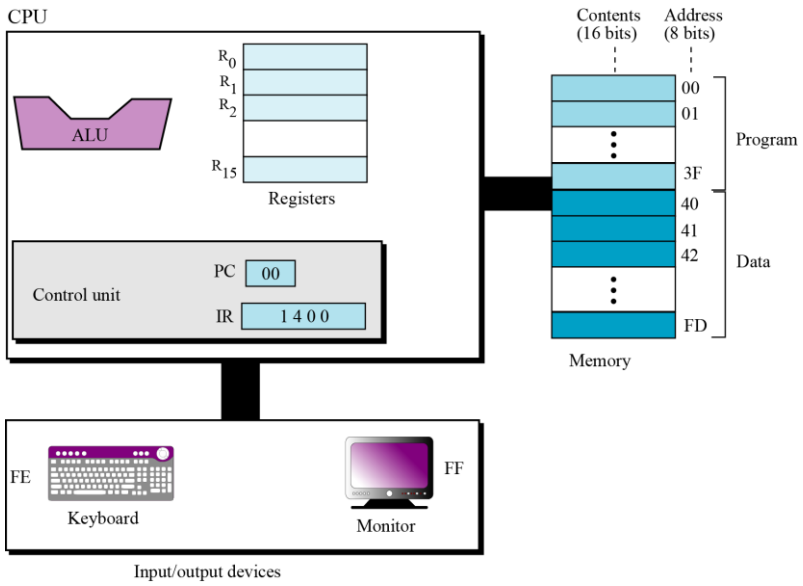


a. Concept b. Implementation

5-8 A SIMPLE COMPUTER

To explain the architecture of computers as well as their instruction processing, we introduce a simple (unrealistic) computer, as shown in Figure 5.30. Our simple computer has three components: CPU, memory, and an input/output subsystem.

Figure 5.30 The components of a simple computer



Instruction set

Our simple computer is capable of having a set of sixteen instructions, although we are using only fourteen of these instructions. Each computer instruction consists of two parts: the **operation code (opcode)** and the **operand (s).** The opcode specifies the type of operation to be performed on the operand (s). Each instruction consists of sixteen bits divided into four 4-bit fields. The leftmost field contains the opcode and the other three fields contains the operand or address of operand (s), as shown in Figure 5.31.

Instruction set

Our simple computer is capable of having a set of sixteen instructions, although we are using only fourteen of these instructions. Each computer instruction consists of two parts: the **operation code (opcode)** and the **operand (s).** The opcode specifies the type of operation to be performed on the operand (s). Each instruction consists of sixteen bits divided into four 4-bit fields. The leftmost field contains the opcode and the other three fields contains the operand or address of operand (s), as shown in Figure 5.31.

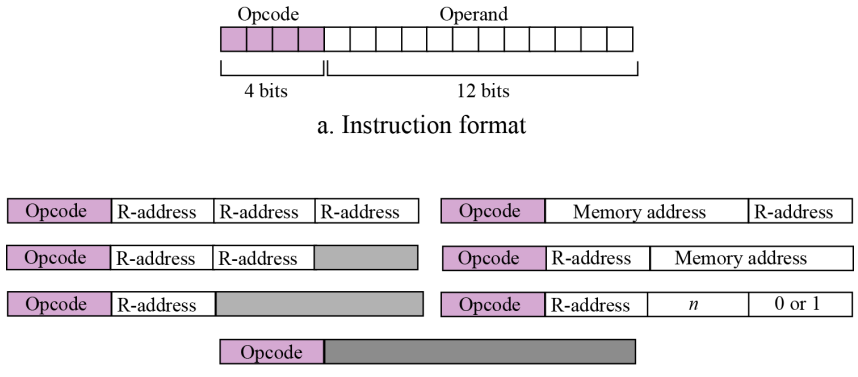
Instruction set

Our simple computer is capable of having a set of sixteen instructions, although we are using only fourteen of these instructions. Each computer instruction consists of two parts: the **operation code (opcode)** and the **operand (s).** The opcode specifies the type of operation to be performed on the operand (s). Each instruction consists of sixteen bits divided into four 4-bit fields. The leftmost field contains the opcode and the other three fields contains the operand or address of operand (s), as shown in Figure 5.31.

Instruction set

Our simple computer is capable of having a set of sixteen instructions, although we are using only fourteen of these instructions. Each computer instruction consists of two parts: the **operation code (opcode)** and the **operand (s).** The opcode specifies the type of operation to be performed on the operand (s). Each instruction consists of sixteen bits divided into four 4-bit fields. The leftmost field contains the opcode and the other three fields contains the operand or address of operand (s), as shown in Figure 5.31.

Figure 5.31 Format and different instruction types



5.8.5 Processing the instructions

Our simple computer, like most computers, uses machine cycles. A cycle is made of three phases: fetch, decode, and execute. During the fetch phase, the instruction whose address is determined by the PC is obtained from the memory and loaded into the IR. The PC is then incremented to point to the next instruction. During the decode phase, the instruction in IR is decoded and the required operands are fetched from the register or from memory. During the execute phase, the instruction is executed and the results are placed in the appropriate memory location or the register. Once the third phase is completed, the control unit starts the cycle again, but now the PC is pointing to the next instruction. The process continues until the CPU reaches a HALT instruction.

5.61

Table 5.4 List of instructions for the simple computer

Instruction	*Code*	*Operands*				*Action*
	d_1	d_2	d_3	d_4		
HALT	0					Stops the execution of the program
LOAD	1	R_D	M_S			$R_D \leftarrow M_S$
STORE	2	M_D		R_S		$M_D \leftarrow R_S$
ADDI	3	R_D	R_{S1}	R_{S2}		$R_D \leftarrow R_{S1} + R_{S2}$
ADDF	4	R_D	R_{S1}	R_{S2}		$R_D \leftarrow R_{S1} + R_{S2}$
MOVE	5	R_D	R_S			$R_D \leftarrow R_S$
NOT	6	R_D	R_S			$R_D \leftarrow \overline{R_S}$
AND	7	R_D	R_{S1}	R_{S2}		$R_D \leftarrow R_{S1}$ AND R_{S2}
OR	8	R_D	R_{S1}	R_{S2}		$R_D \leftarrow R_{S1}$ OR R_{S2}
XOR	9	R_D	R_{S1}	R_{S2}		$R_D \leftarrow R_{S1}$ XOR R_{S2}
INC	A	R				R $\leftarrow$ R + 1
DEC	B	R				R $\leftarrow$ R – 1
ROTATE	C	R	n	0 or 1		Rot_n R
JUMP	D	R	n			IF $R_0 \neq$ R then PC = *n*, otherwise continue

Key: R_S, R_{S1}, R_{S2}: Hexadecimal address of source registers
R_D: Hexadecimal address of destination register
M_S: Hexadecimal address of source memory location
M_D: Hexadecimal address of destination memory location
n: hexadecimal number
d_1, d_2, d_3, d_4: First, second, third, and fourth hexadecimal digits

5.62

An example

Let us show how our simple computer can add two integers A and B and create the result as C. We assume that integers are in two's complement format. Mathematically, we show this operation as:

$$C = A + B$$

We assume that the first two integers are stored in memory locations $(40)_{16}$ and $(41)_{16}$ and the result should be stored in memory location $(42)_{16}$. To do the simple addition needs five instructions, as shown next:

5.63

1. Load the contents of M_{40} into register R_0 ($R_0 \leftarrow M_{40}$).
2. Load the contents of M_{41} into register R_1 ($R_1 \leftarrow M_{41}$).
3. Add the contents of R_0 and R_1 and place the result in R_2 ($R_2 \leftarrow R_0 + R_1$).
4. Store the contents R_2 in M_{42} ($M_{42} \leftarrow$ R2).
5. Halt.

In the language of our simple computer, these five instructions are encoded as:

Code	*Interpretation*			
$(1040)_{16}$	**1**: LOAD	0: R_0	40: M_{40}	
$(1141)_{16}$	**1**: LOAD	1: R_1	41: M_{41}	
$(3201)_{16}$	**3**: ADDI	2: R_2	0: R_0	1: R_1
$(2422)_{16}$	**2**: STORE	42: M_{42}		2: R_2
$(0000)_{16}$	**0**: HALT			

5.8.6 Storing program and

We can store the five-line program in memory starting from location $(00)_{16}$ to $(04)_{16}$. We already know that the data needs to be stored in memory locations (40)16, (41)16, and (42)16.

5.8.7 Cycles

Our computer uses one cycle per instruction. If we have a small program with five instructions, we need five cycles. We also know that each cycle is normally made up of three steps: fetch, decode, execute. Assume for the moment that we need to add 161 + 254 = 415. The numbers are shown in memory in hexadecimal is, $(00A1)_{16}$, $(00FE)_{16}$, and $(019F)_{16}$.

5.65

Figure 5.32 Status of cycle 1

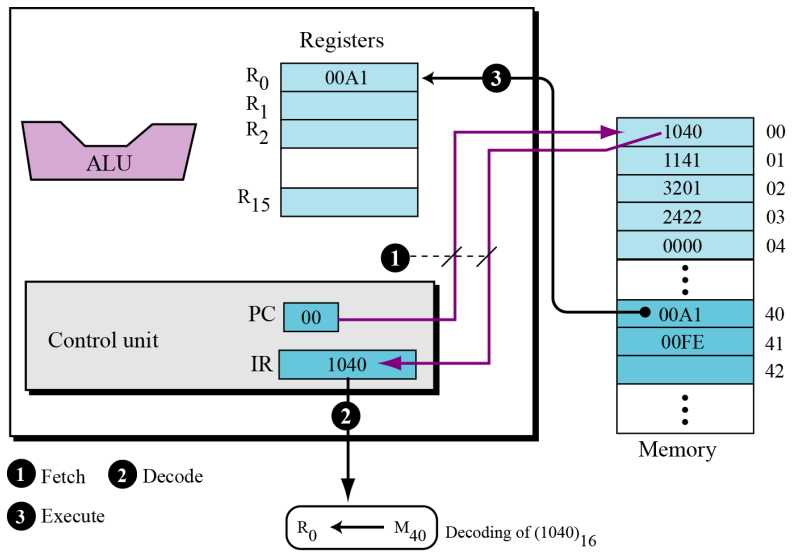


Figure 5.33 Status of cycle 2

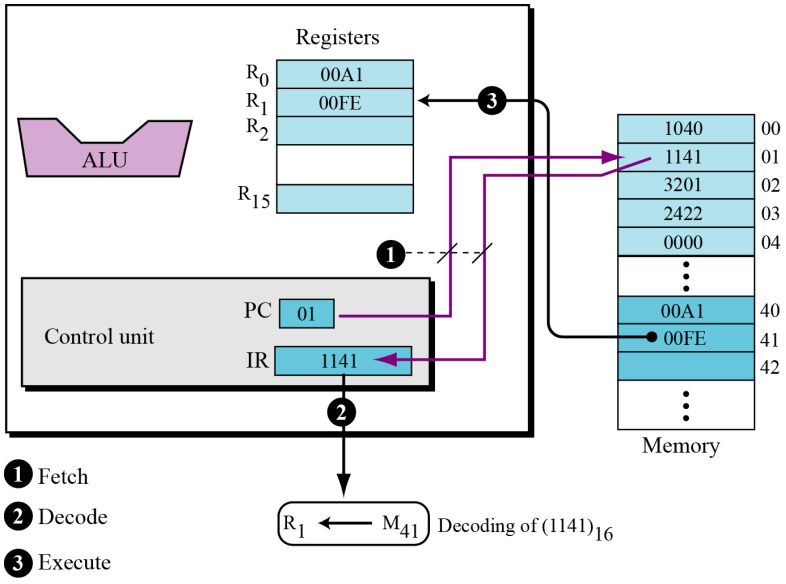


Figure 5.34 Status of cycle 3

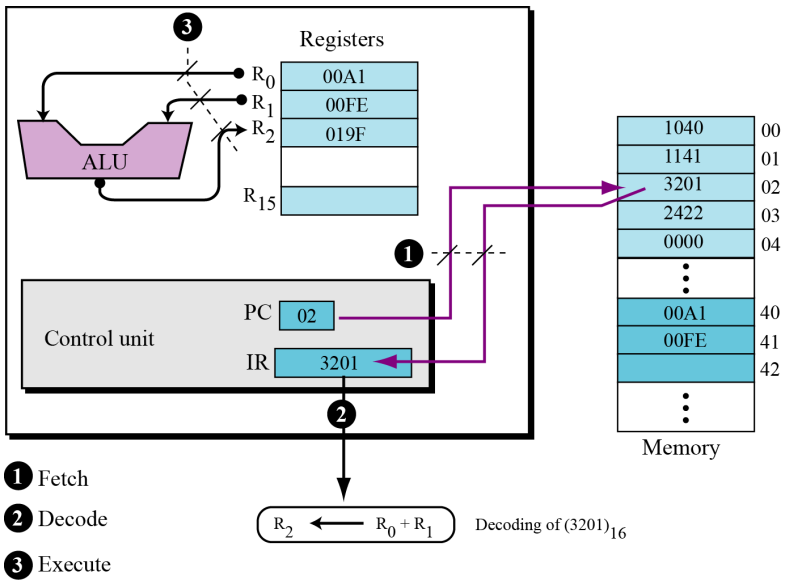


Figure 5.35 Status of cycle 4

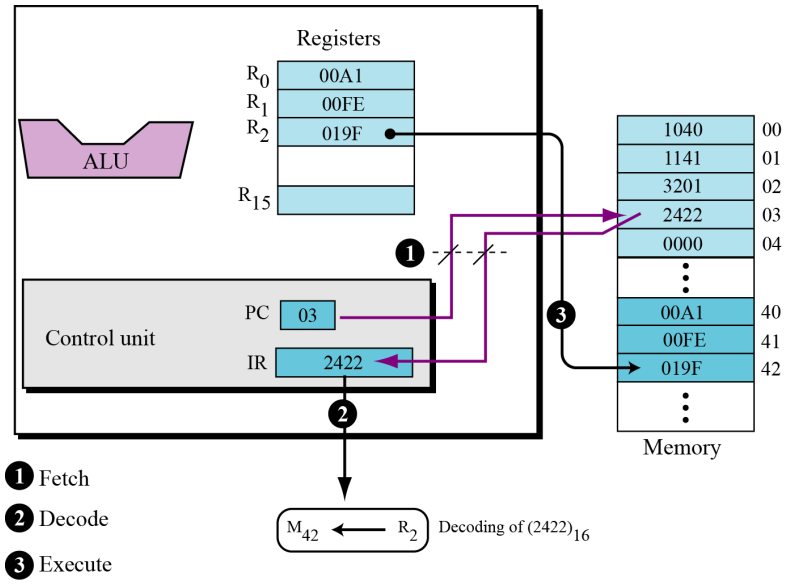
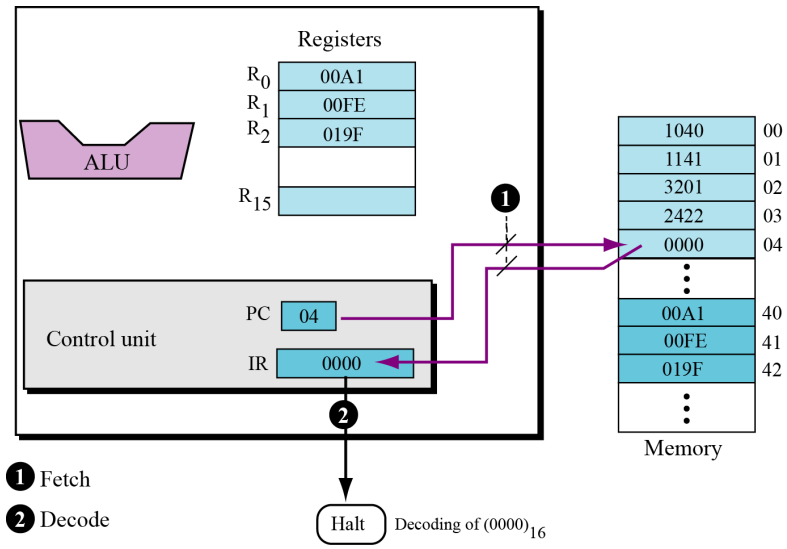


Figure 5.36 Status of cycle 5



5.8.8 Another example

In the previous example we assumed that the two integers to be added were already in memory. We also assume that the result of addition will be held in memory. You may ask how we can store the two integers we want to add in memory, or how we use the result when it is stored in the memory. In a real situation, we enter the first two integers into memory using an input device such as keyboard, and we display the third integer through an output device such as a monitor. Getting data via an input device is normally called a read operation, while sending data to an output device is normally called a write operation. To make our previous program more practical, we need modify it as follows:

5.71

1. Read an integer into M_{40}.
2. $R_0 \leftarrow M_{40}$.
3. Read an integer into M_{41}.
4. $R_1 \leftarrow M_{41}$.
5. $R_2 \leftarrow R_0 + R_1$.
6. $M_{42} \leftarrow R2$.
7. Write the integer from M_{42}.
8. Halt.

In our computer we can simulate read and write operations using the LOAD and STORE instruction. Furthermore, LOAD and STORE read data input to the CPU and write data from the CPU. We need two instructions to read data into memory or write data out of memory. The read operation is:

5.72

$R \leftarrow M_{EF}$	Because the keyboard is assumed to be memory location $(EF)_{16}$
$M \leftarrow R$	

The write operation is:

$R \leftarrow M$	
$M_{FF} \leftarrow R$	Because the monitor is assumed to be memory location $(EF)_{16}$

The input operation must always read data from an input device into memory: the output operation must always write data from memory to an output device.

5.73

The program is coded as:

1	$(1FFE)_{16}$	5	$(1040)_{16}$	9	$(1F42)_{16}$
2	$(240F)_{16}$	6	$(1141)_{16}$	10	$(2FFF)_{16}$
3	$(1FFE)_{16}$	7	$(3201)_{16}$	11	$(0000)_{16}$
4	$(241F)_{16}$	8	$(2422)_{16}$		

Operations 1 to 4 are for input and operations 9 and 10 are for output. When we run this program, it waits for the user to input two integers on the keyboard and press the enter key. The program then calculates the sum and displays the result on the monitor.

5.74

5.8.9 Reusability

Operation …….

5.75