Chapter 5

Computer Organization

OBJECTIVES

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

- **List the three categories of operations performed on data.**
- **Perform unary and binary logic operations on bit patterns.**
- **Distinguish between logic shift operations and arithmetic shift operations.**
- **Perform addition and subtraction on integers when they are stored in two's complement format.**
- **Perform addition and subtraction on integers when stored in sign-and-magnitude format.**
- **Perform addition and subtraction operations on reals stored in floating-point format.**

Three subsystems in Computer

- The central processing unit (CPU), the main memory and the input/output subsystem.

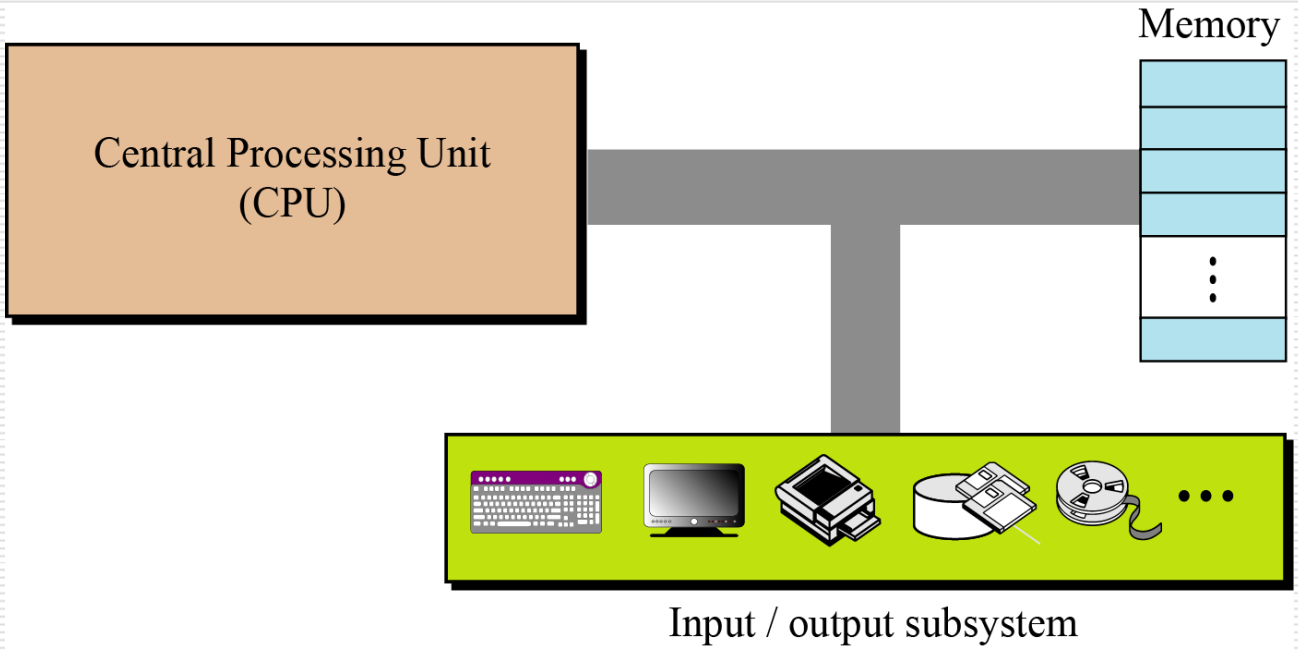

Figure 5.1 **Computer hardware (subsystems)**

5.1

CENTRAL PROCESSING UNIT

Outlines

- Three parts in CPU
  - An **arithmetic logic unit (ALU)**,
  - A **control unit** and a set of **registers**,
  - Fast storage locations.

ALU

- The central processing unit (CPU) performs operations on data.

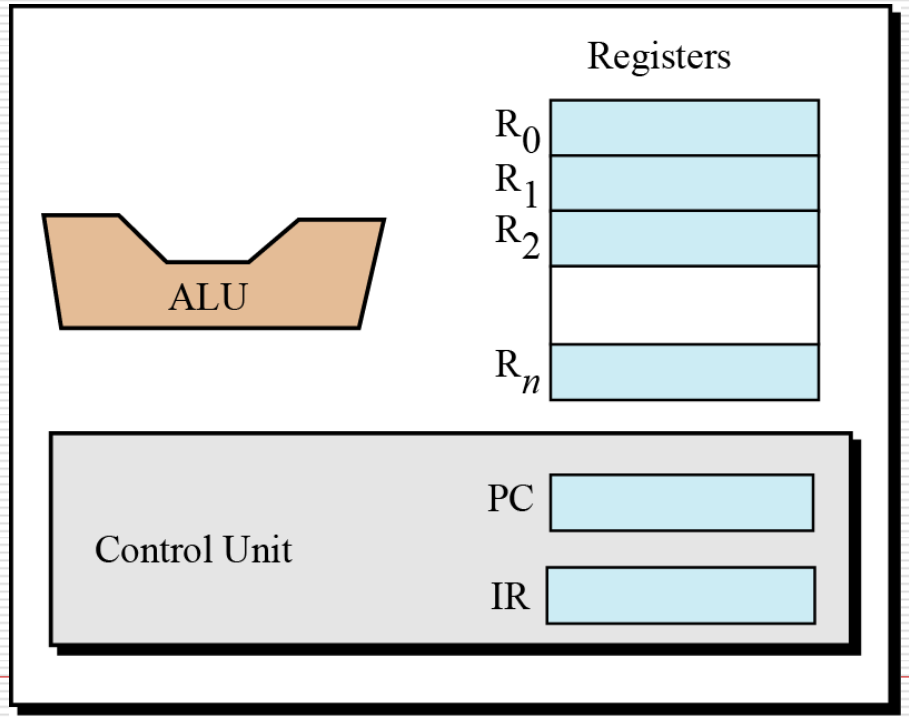

Figure 5.2 Central processing unit (CPU)

Register

- 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.
- Types
  - Data registers
  - Instruction register
  - Program counter

Control Unit

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

5.2

Main Memory

Main Memory

- **Main memory** is the second major subsystem in a computer (Figure 5.3).

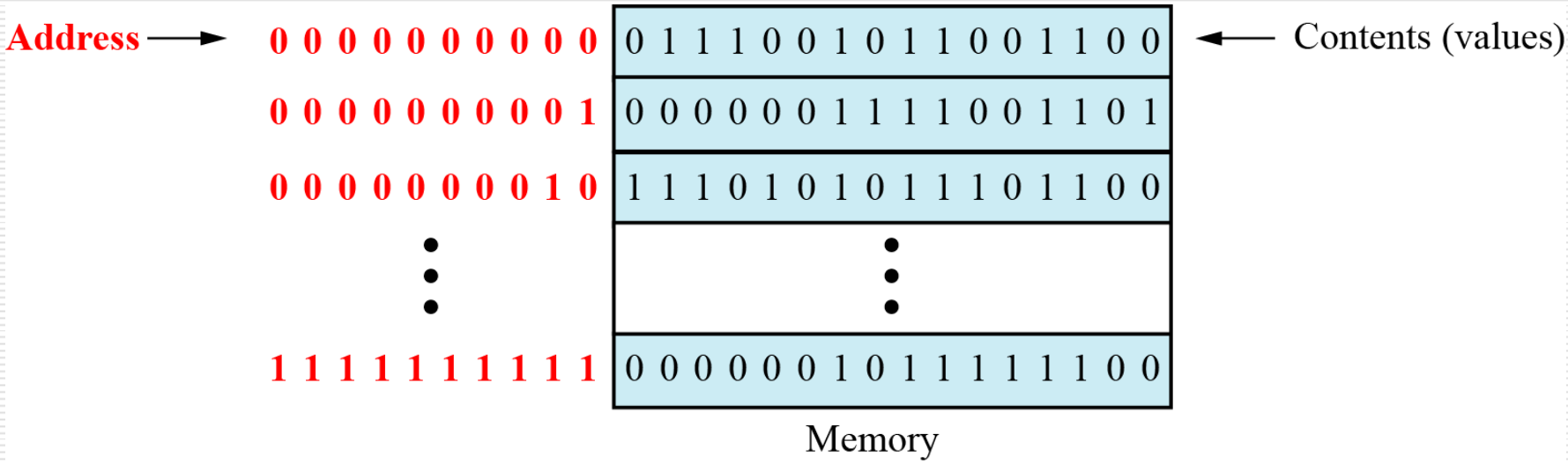

Figure 5.3 Main memory

Main Memory

- 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
  - 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.

Address space

- To access a word in memory requires an identifier.
  - 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**.

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} ($2^5 \times 2^{20}$). 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. We need $\log_2 2^{24}$, or **24 bits**, to address each word.

Memory Types

- Two main types: **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).

Memory hierarchy

- Computer users need a lot of memory,
  - especially memory that is very fast and inexpensive. (very fast memory is not cheap).
- The solution is **hierarchical** levels of memory.

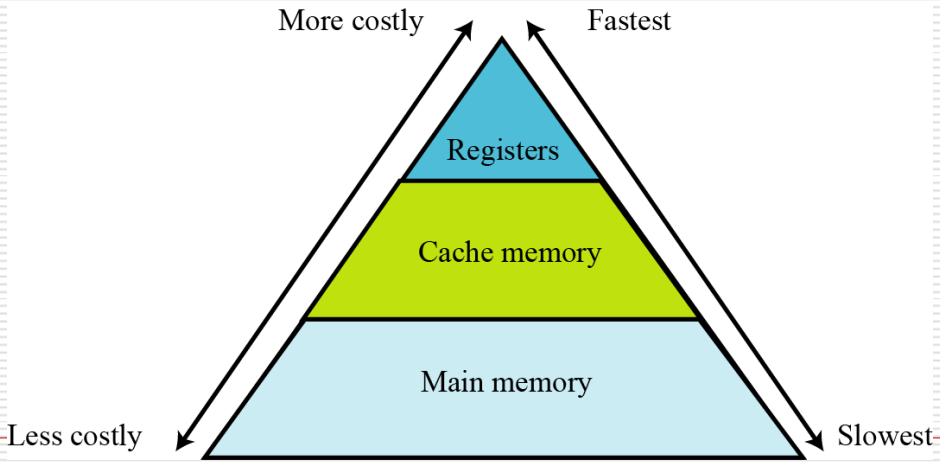

Figure 5.4 Memory hierarchy

Cache Memory

- Cache memory
  - Faster than main memory, but slower than the CPU and its registers.
  - Normally small in size and Being placed between the CPU and main memory (Figure 5.5).

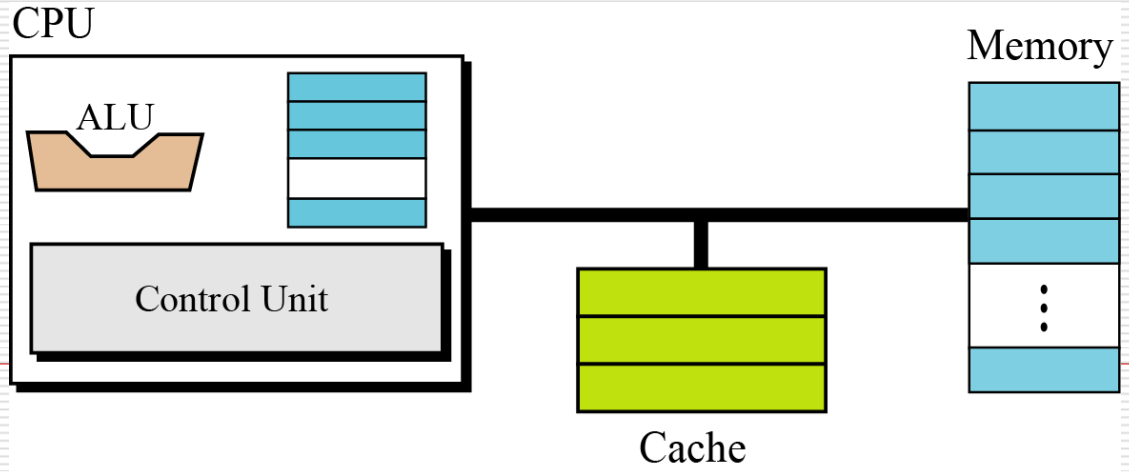

Figure 5.5
Cache memory

5.3

INPUT/OUTPUT SUBSYSTEM

I/O subsystem

- The collection of devices referred to as the input/output (I/O) subsystem in a computer.
- Communicate with the outside world and to store programs and data.
- Two broad categories: **non-storage** and **storage** devices.

I/O subsystem

- **Non-storage devices**
  - Allow the CPU/memory to communicate with the outside world, but they cannot store information.
  - E.g **Keyboard, monitor, and Printer**
- **Storage devices**
  - Store large amounts of information to be retrieved at a later time.
  - Cheaper than main memory, and their contents are nonvolatile (not erased when the power is turned off).
  - Auxiliary storage devices.
    - Two categorize: **magnetic** or **optical**.

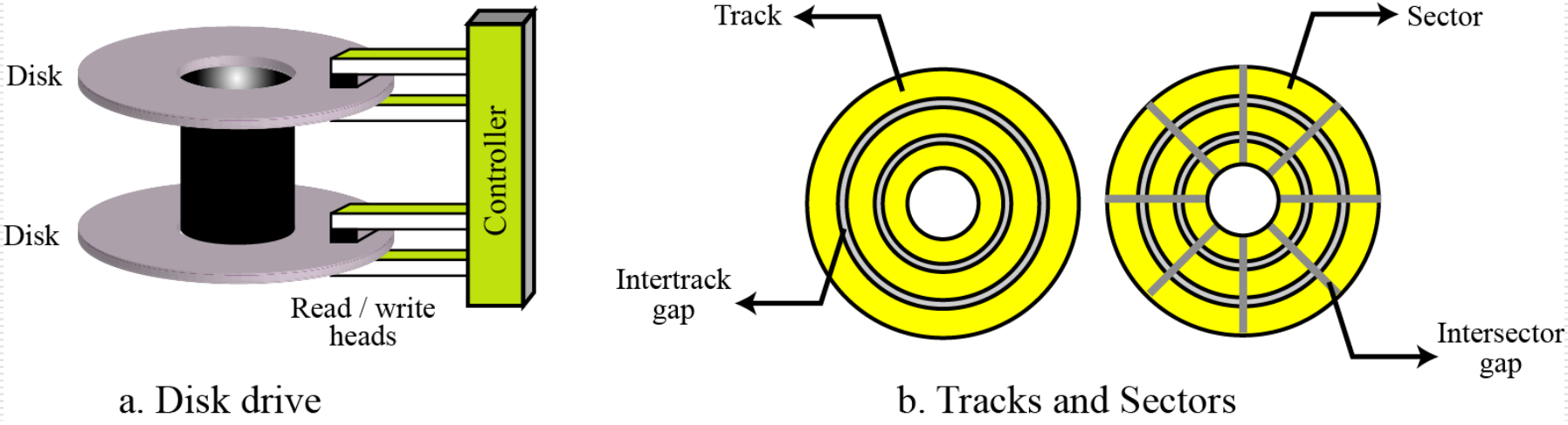

Figure 5.6 A magnetic disk

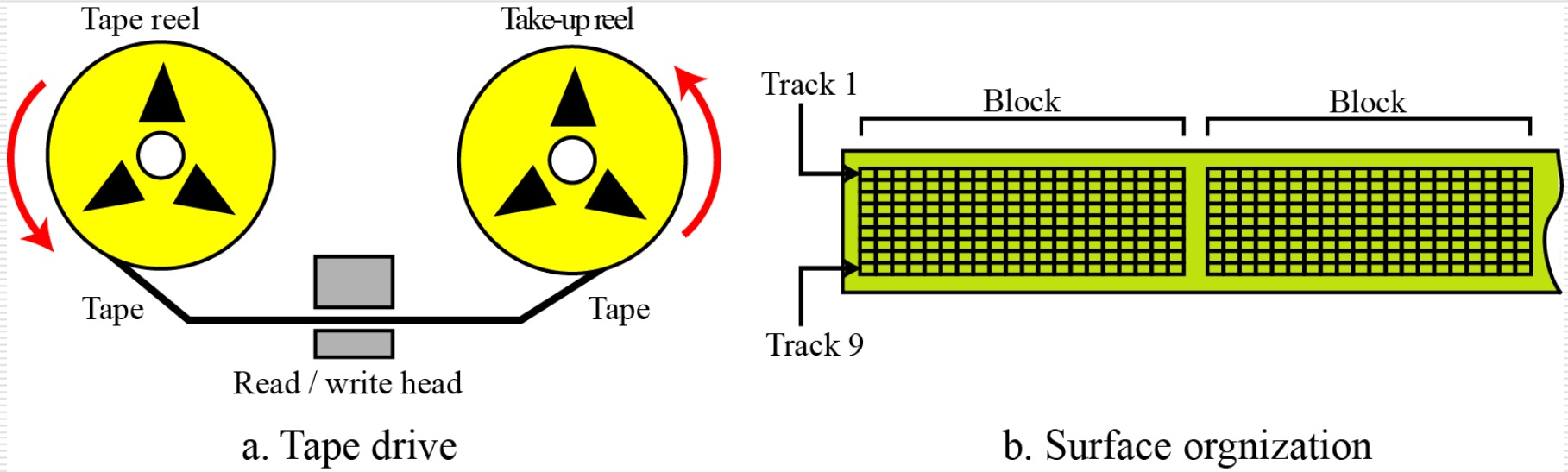

Figure 5.7 A magnetic tape

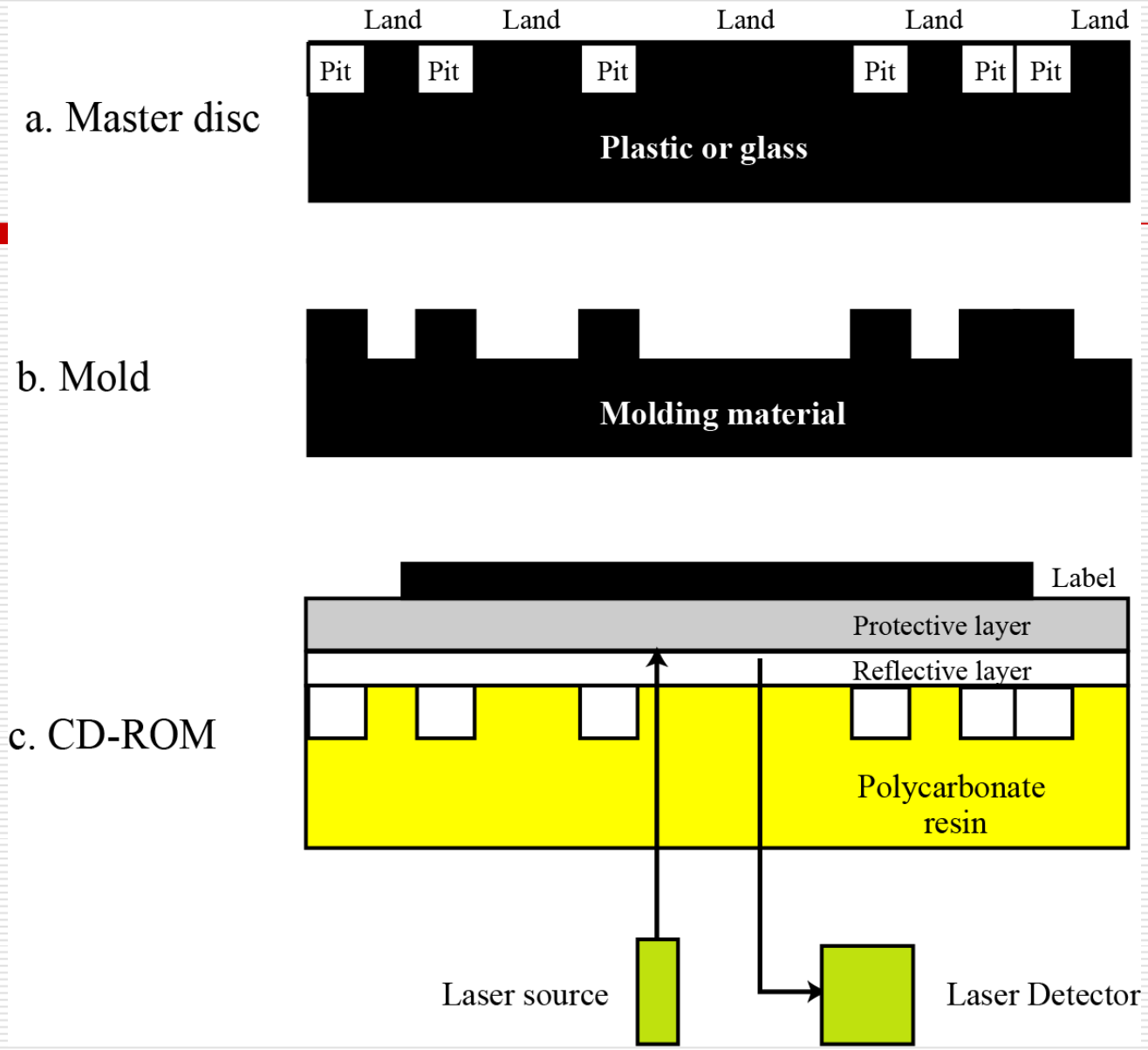

Figure 5.8 Creation and use of CD-ROMs

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

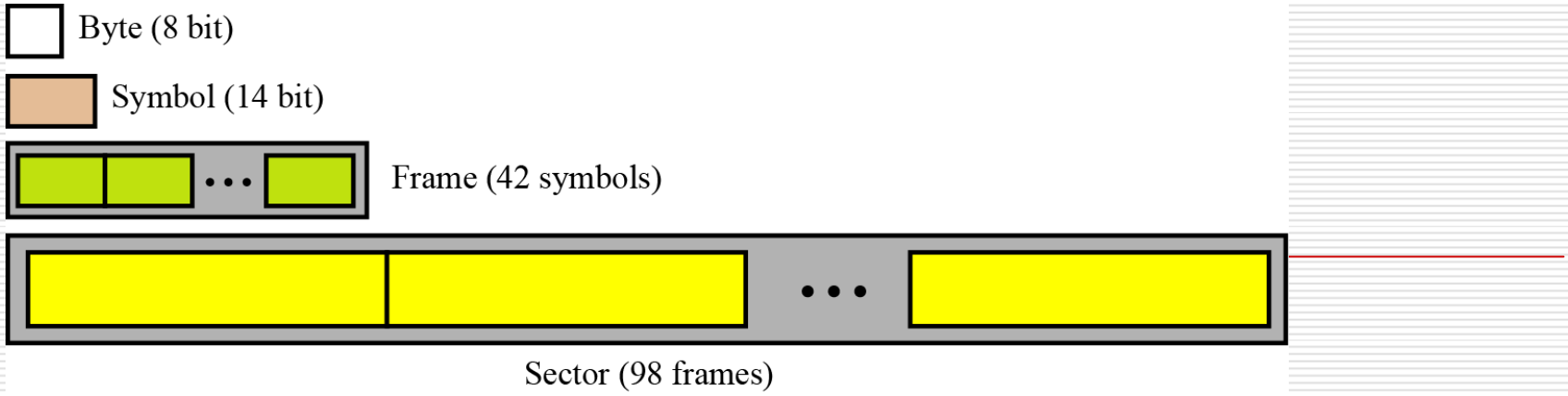

Figure 5.9 CD-ROM format

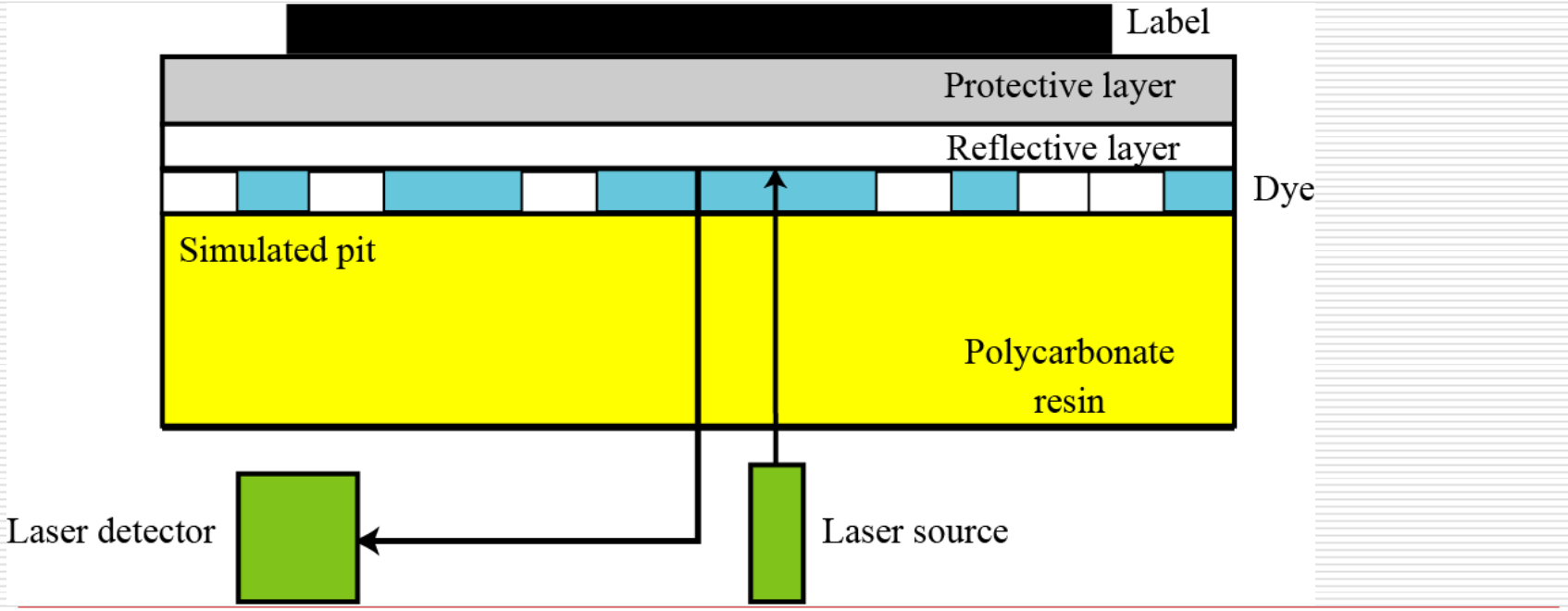

Figure 5.10 Making a CD-R

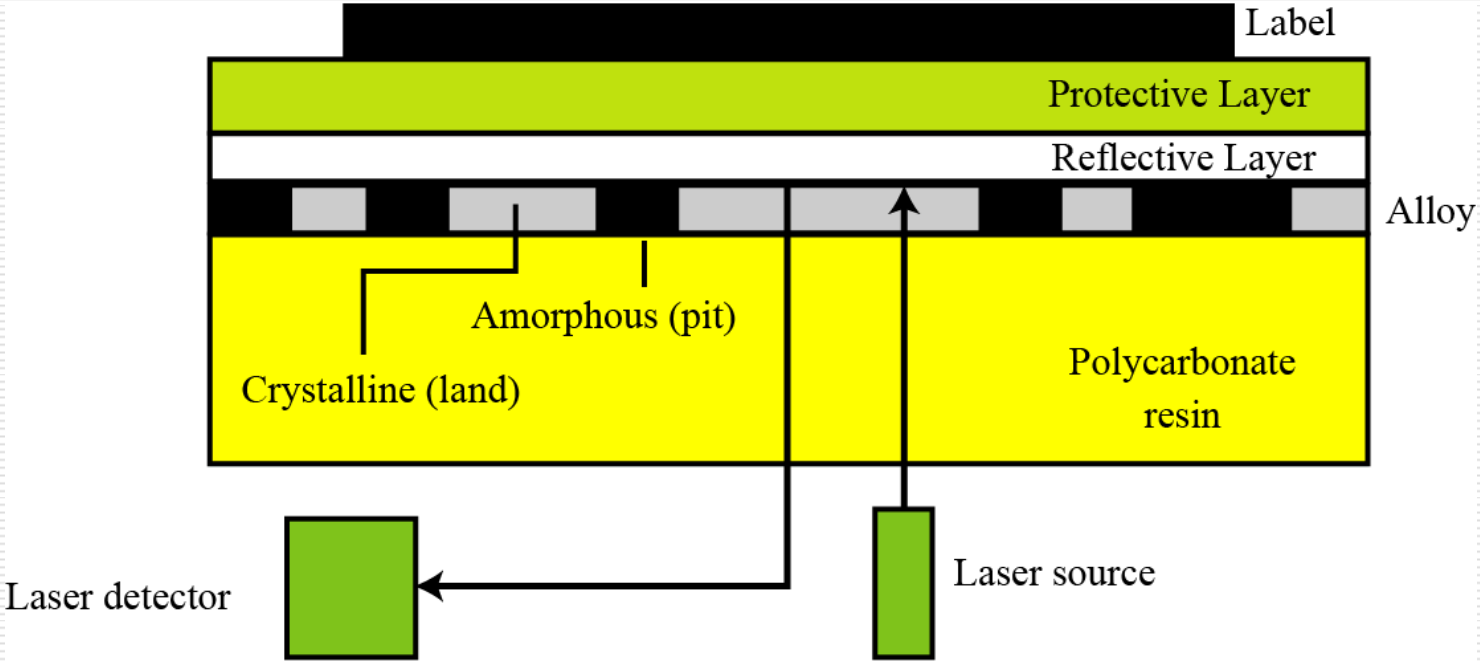

Figure 5.11 **Making a CD-RW**

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

5.4

SUBSYSTEM INTERCONNECTION

Description

- The interconnection plays an important role because information needs to be exchanged between the three subsystems.
  - **Connecting CPU and memory**
  - **Connecting I/O devices**
  - **Addressing input/output devices**

Connecting CPU and memory

- By three groups of connections, each called a **bus**: data bus, address bus and control bus.

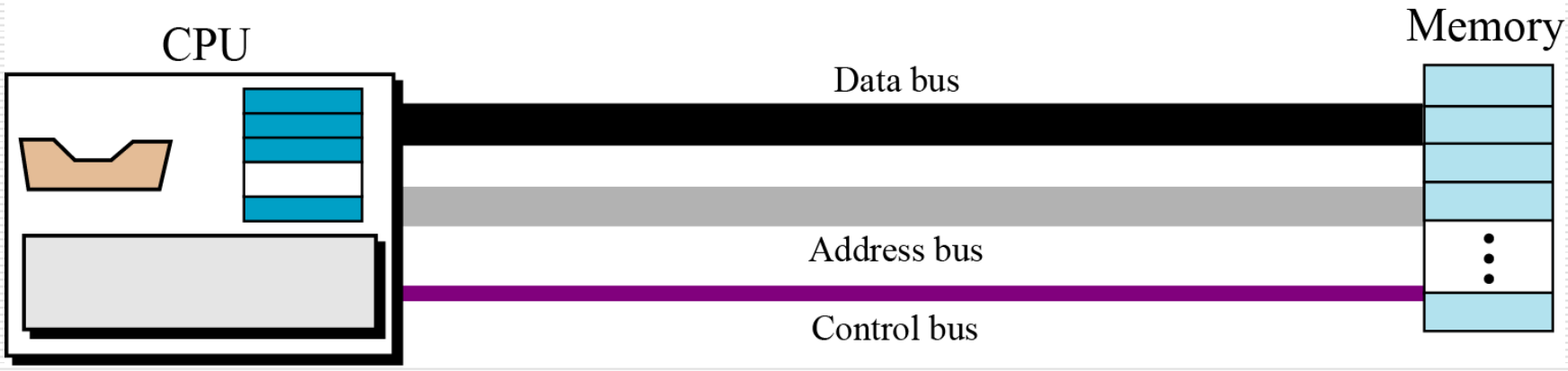

Figure 5.12 **Connecting CPU and memory using three buses**

Connecting I/O devices

- I/O devices cannot be connected directly to the buses that connect the CPU and memory.
- I/O devices are electromechanical, magnetic, or optical devices, whereas the CPU and memory are electronic devices.
  - Much slower speed than the CPU/memory.
  - 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.

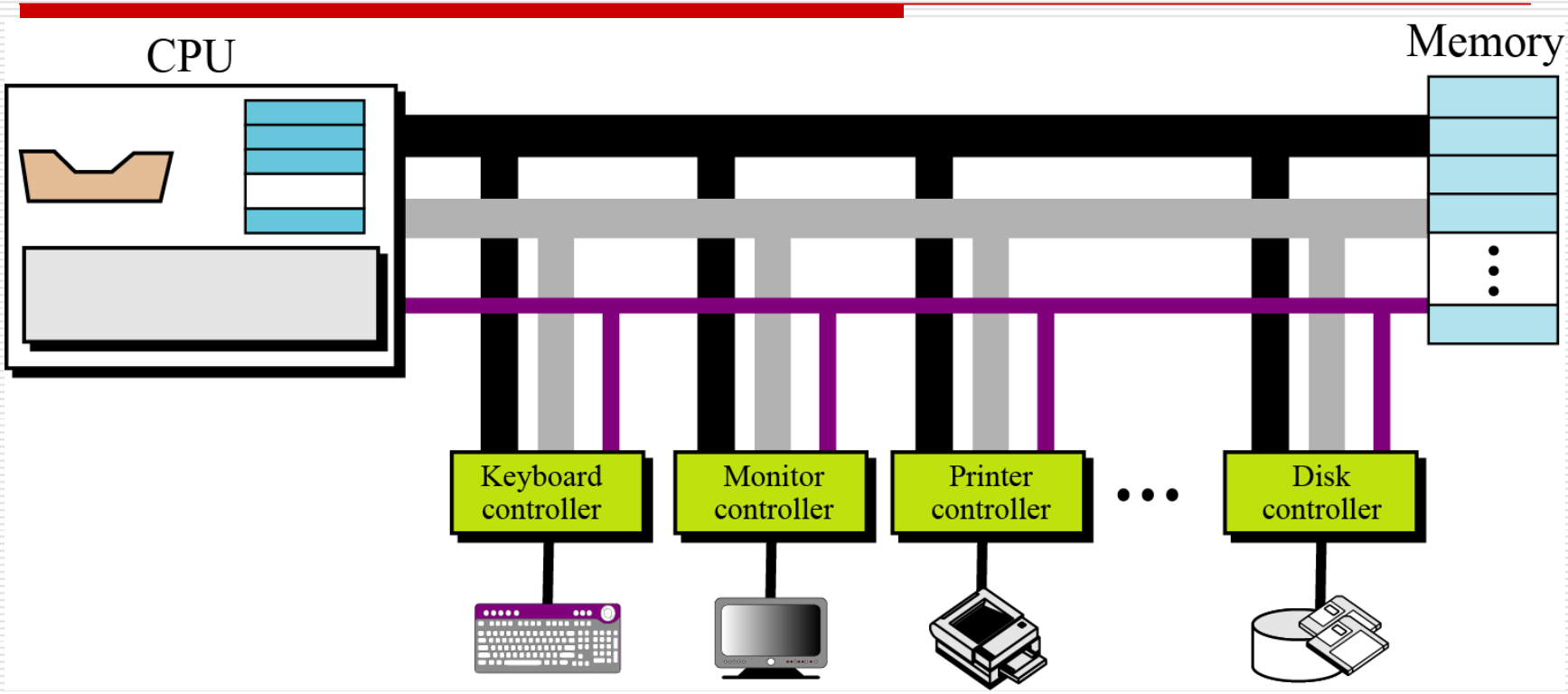

Figure 5.13 Connecting I/O devices to the buses

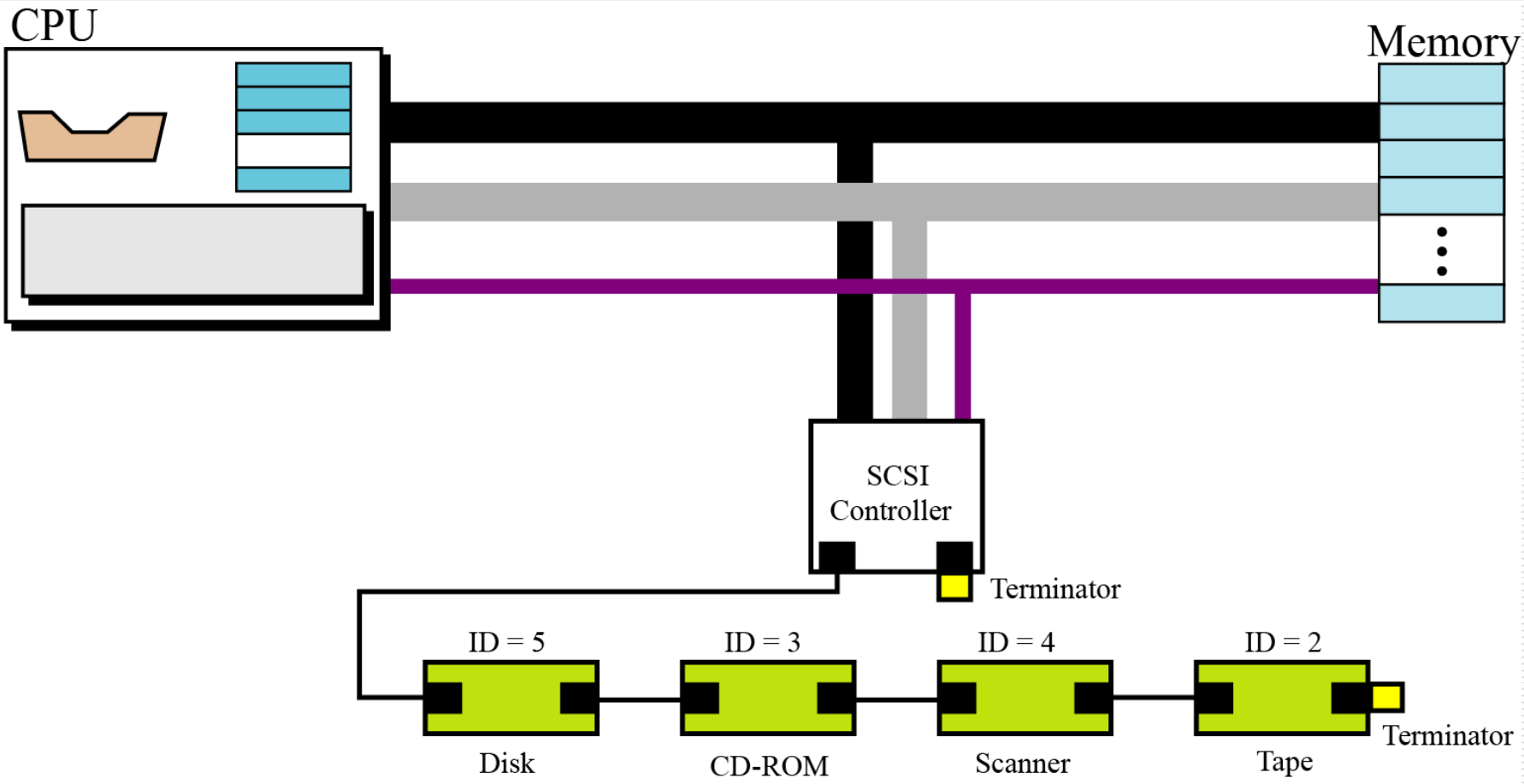

Figure 5.14 SCSI controller

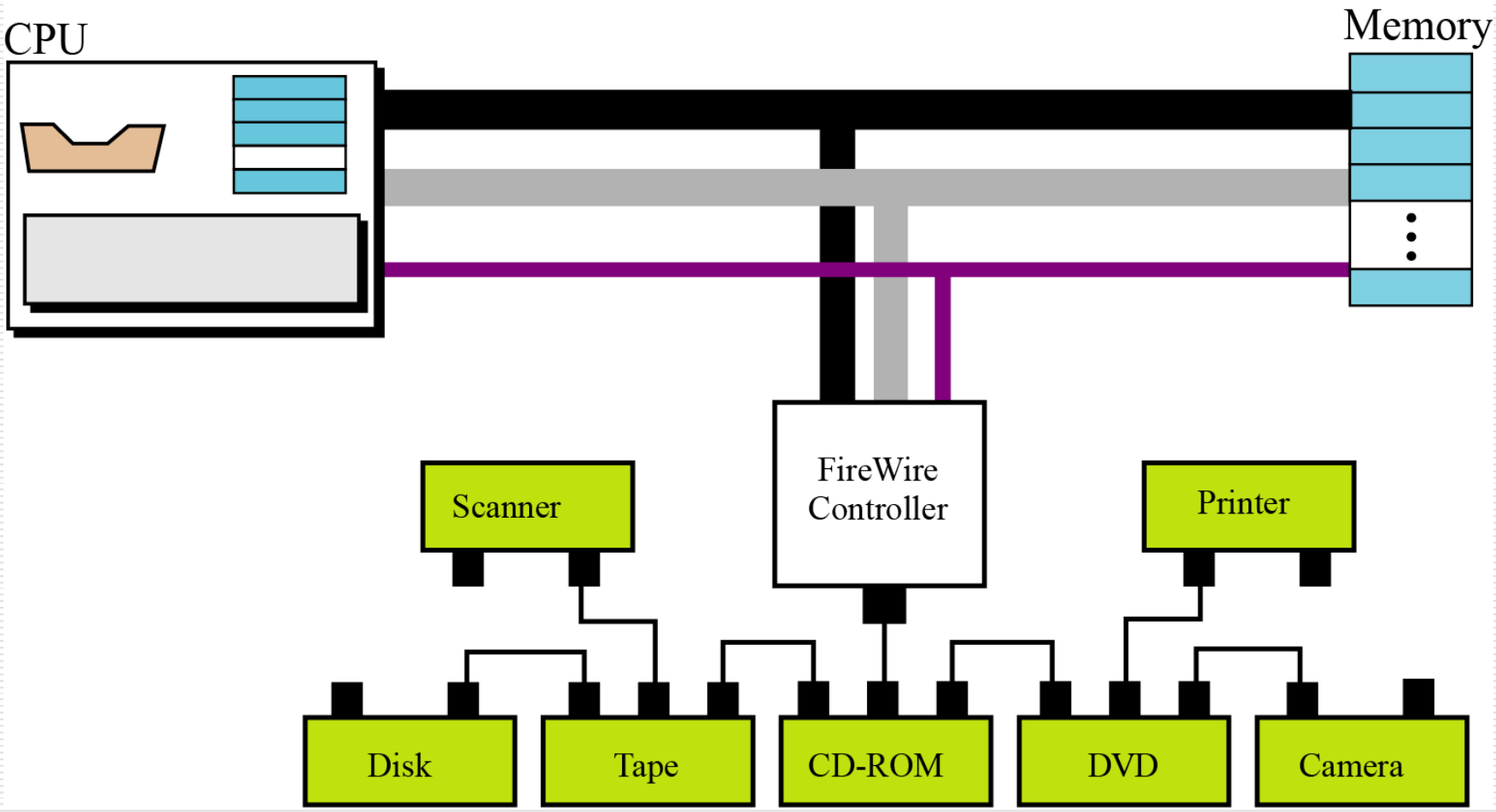

Figure 5.15 **FireWire controller**

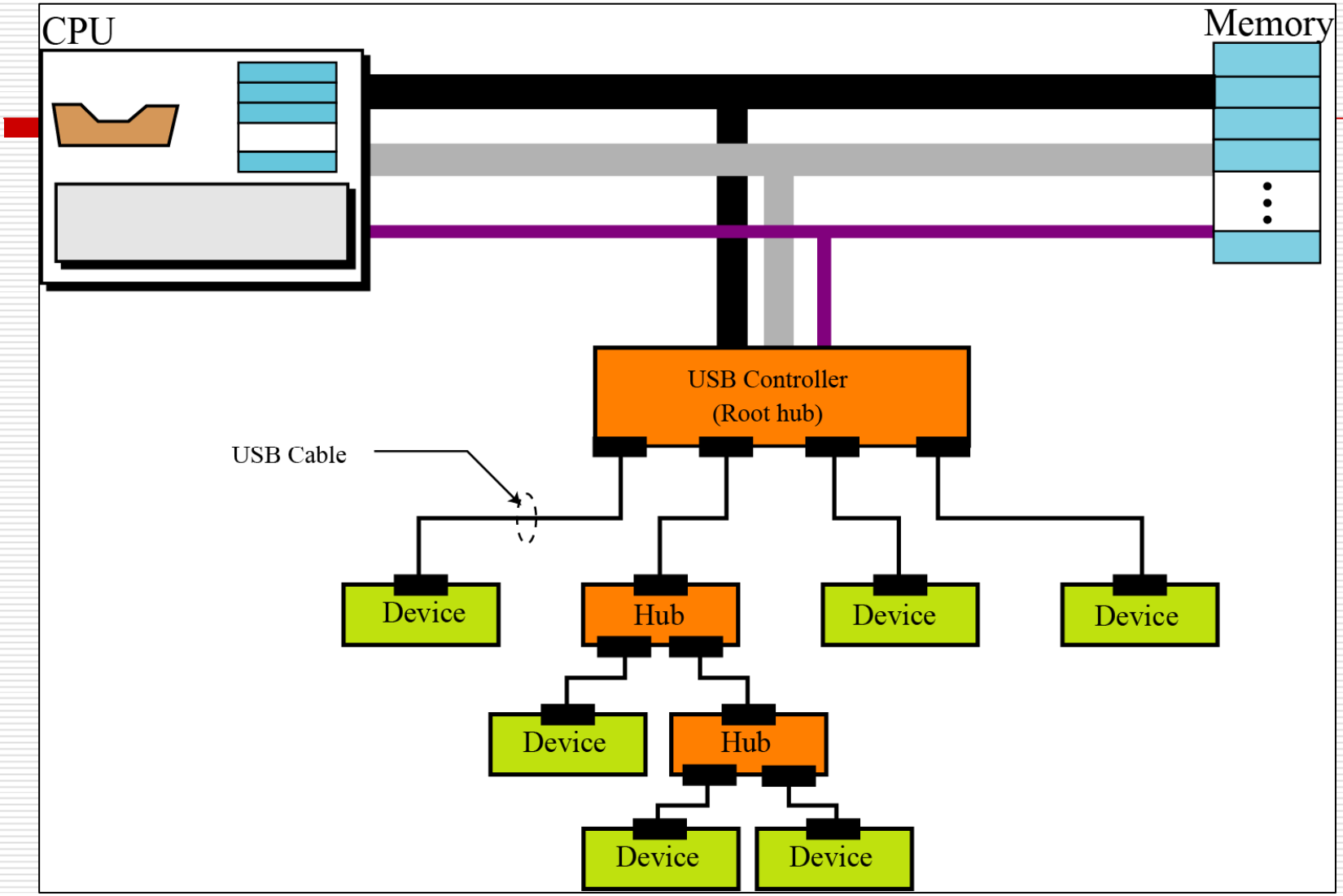

Figure 5.16 **USB controller**

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.

Addressing input/output devices

- Two methods for handling the addressing of I/O devices
  - Isolated I/O
  - memory-mapped I/O.

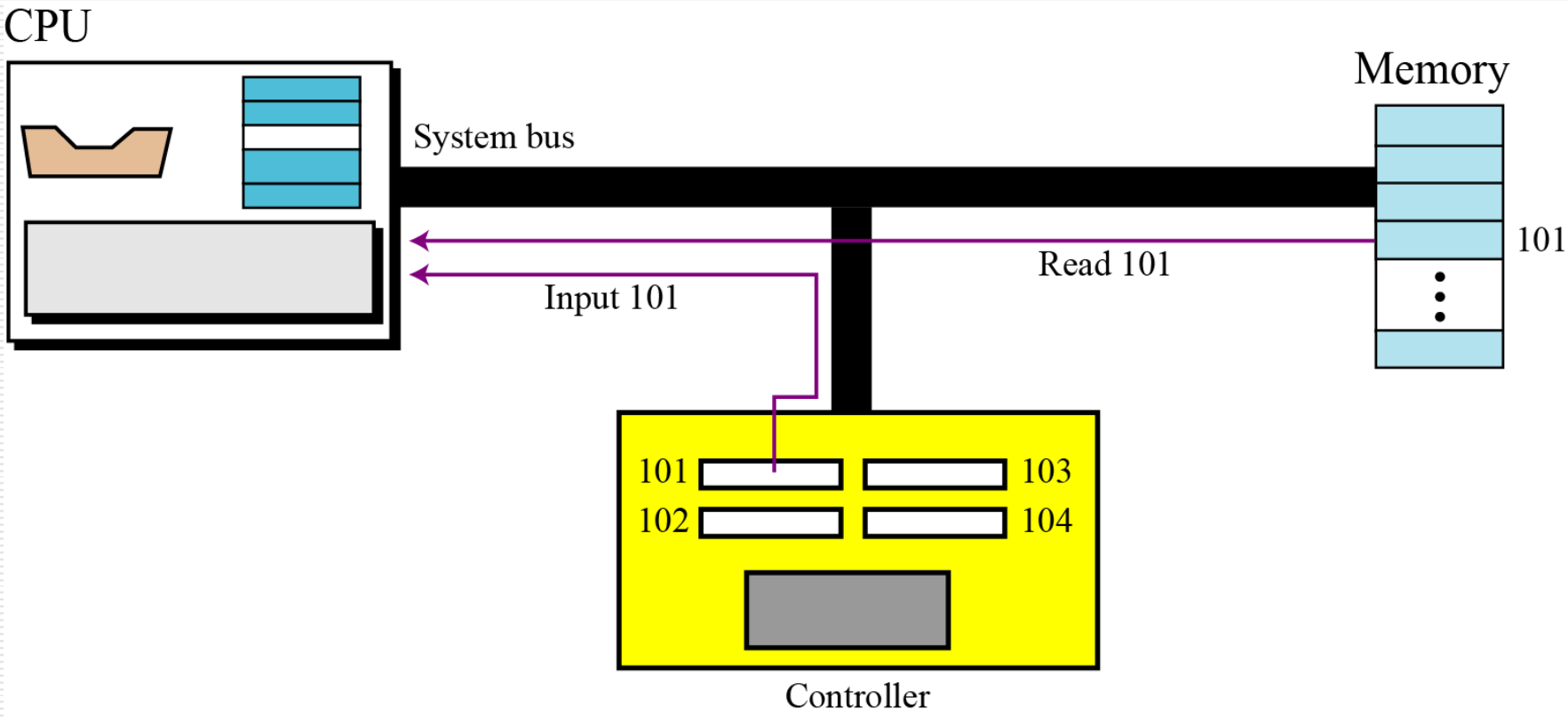

Figure 5.17 **Isolated I/O addressing**

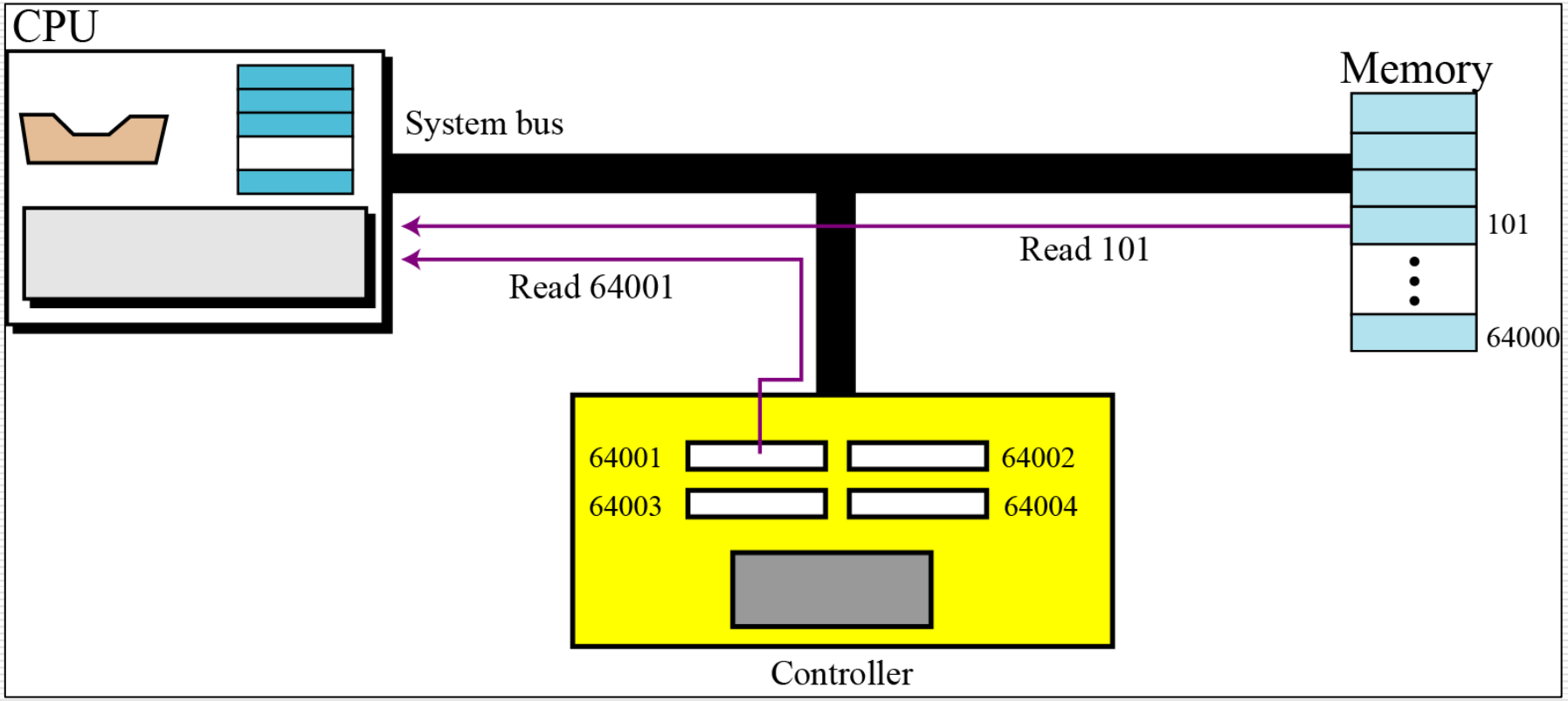

Figure 5.18 **Memory-mapped I/O addressing**

5.5

PROGRAM EXECUTION

Description

- 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.

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).

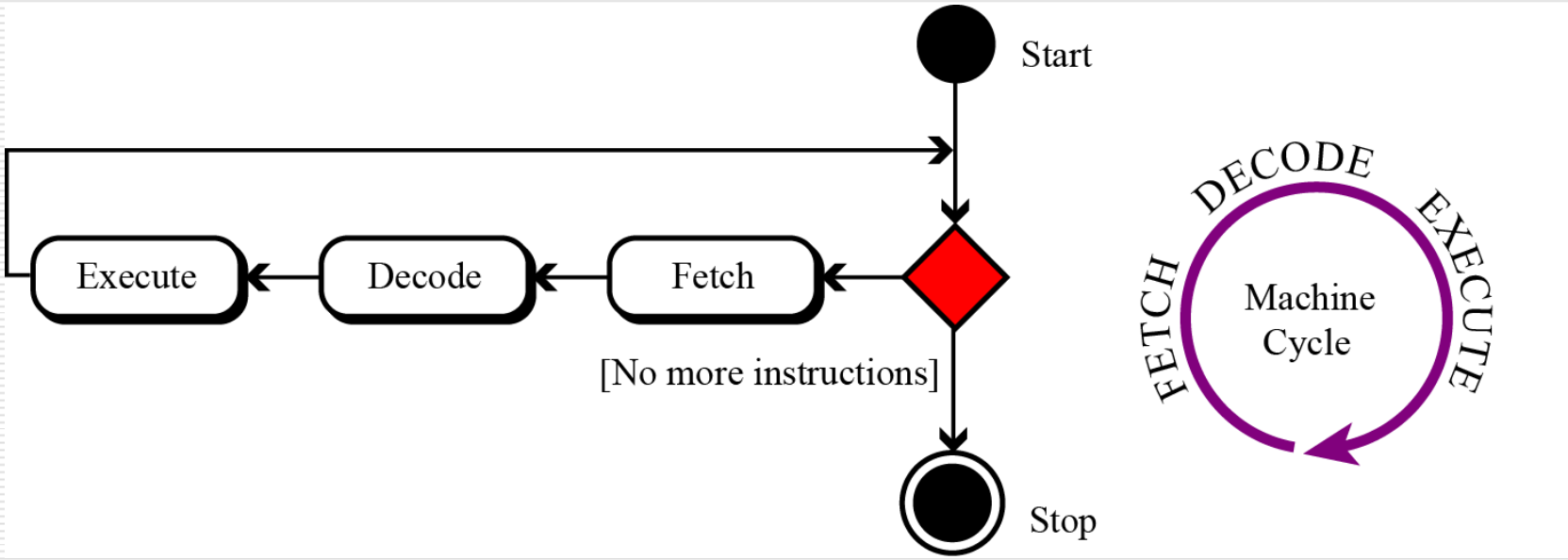

Figure 5.19 The steps of a cycle

Input/output operation

- Commands are required to transfer data from I/O devices to the CPU and memory.
- 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
  - Direct memory access (DMA).

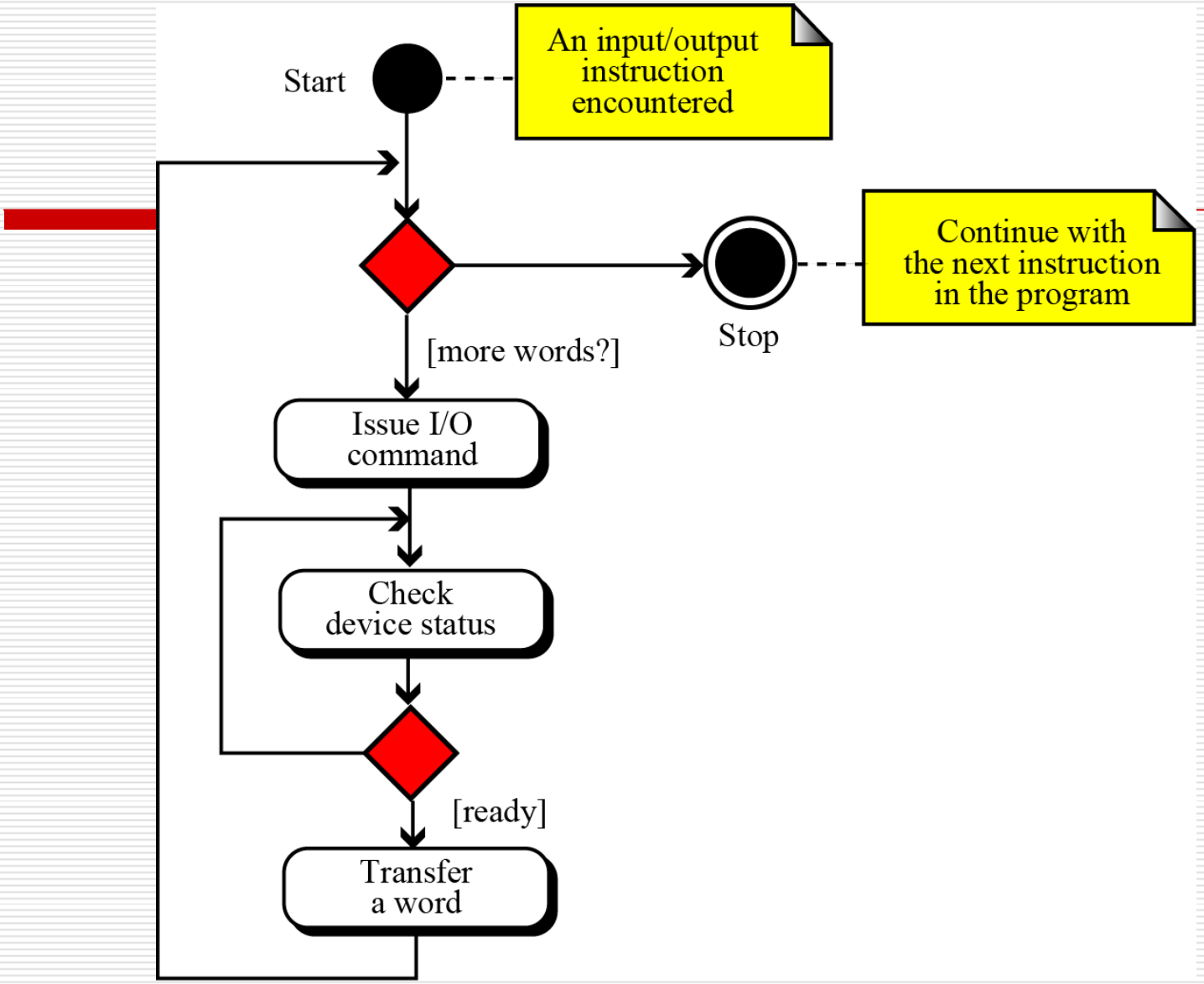

Figure 5.20 Programmed I/O

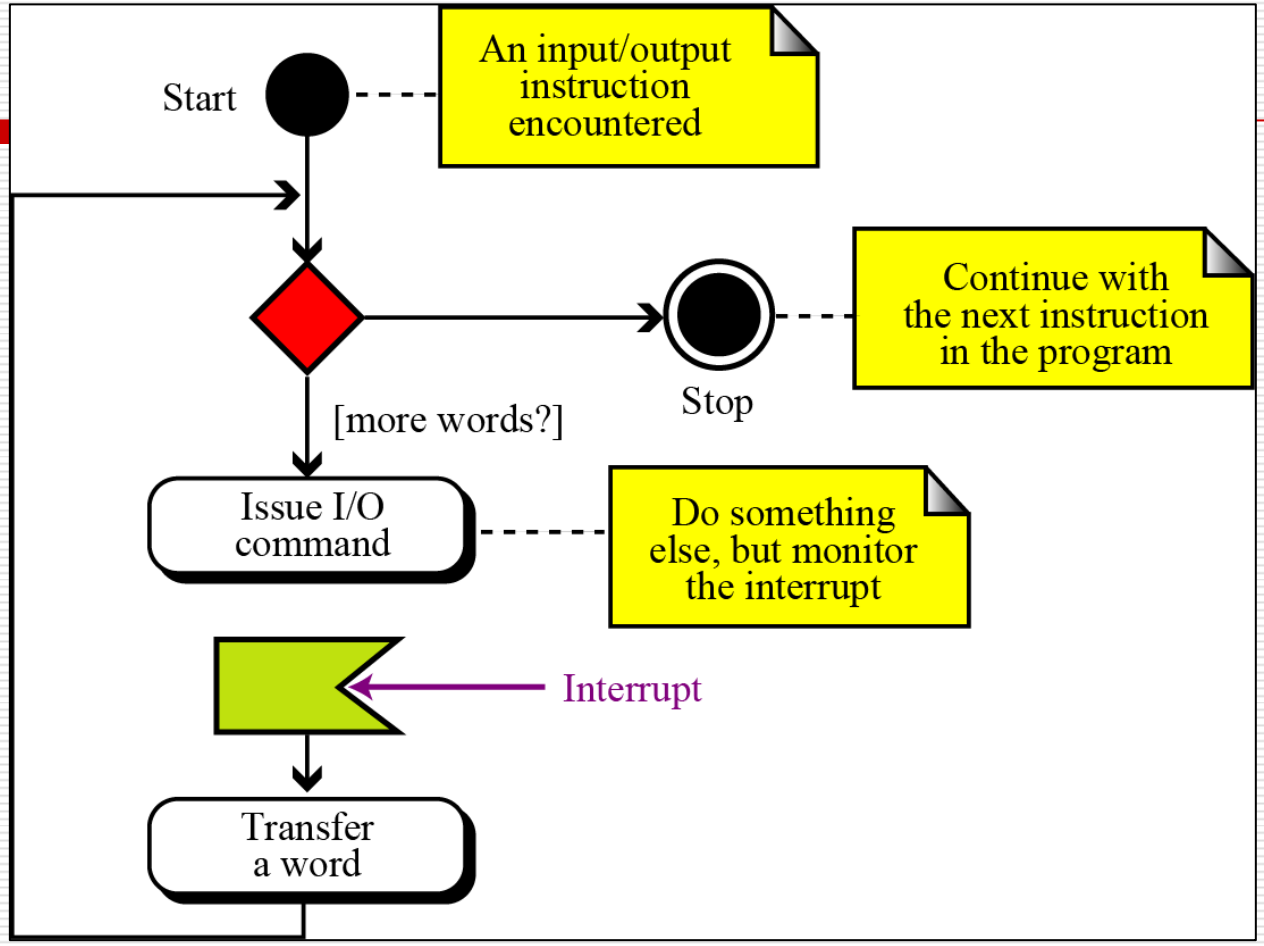

Figure 5.21 Interrupt-driven I/O

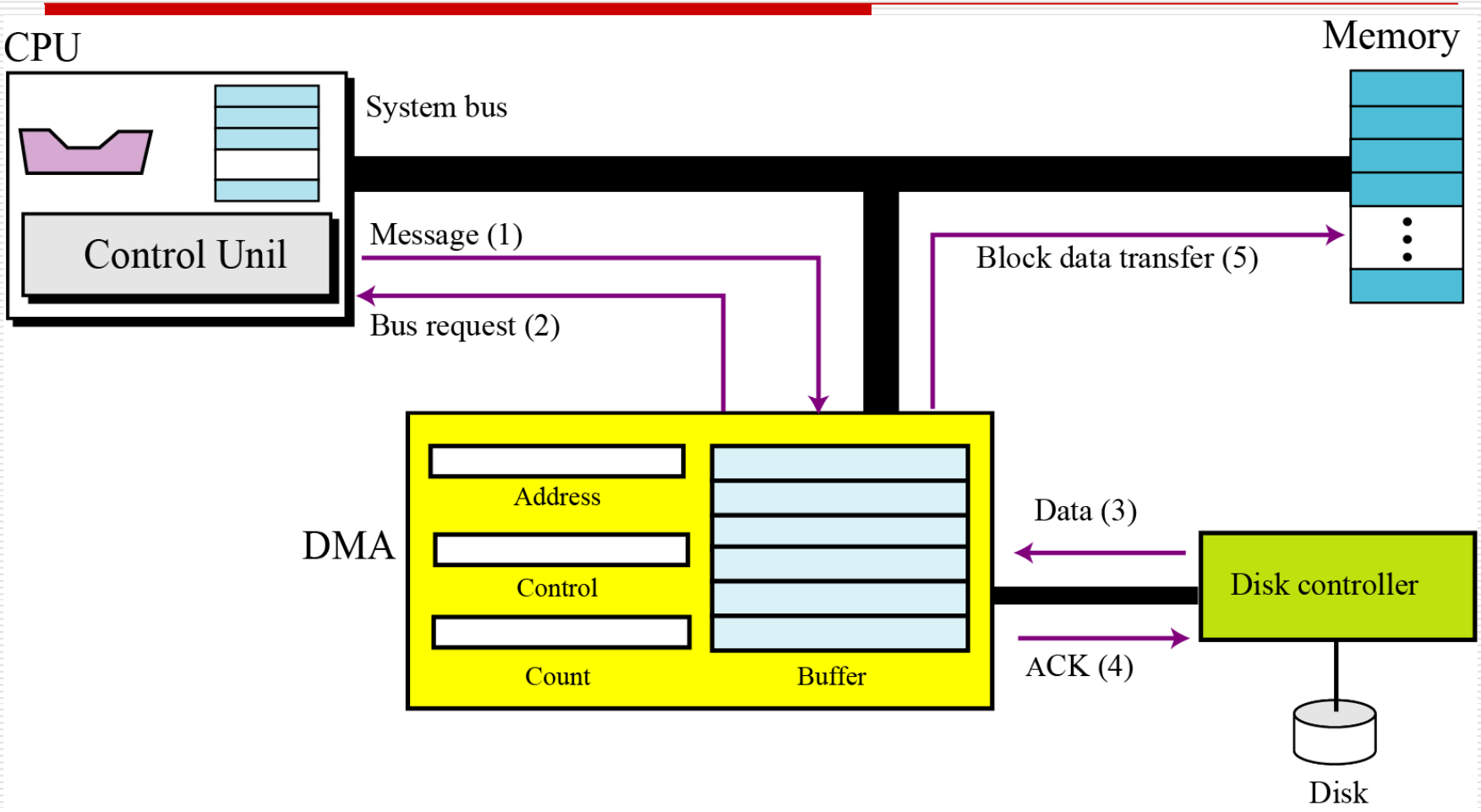

Figure 5.22 DMA connection to the general bus

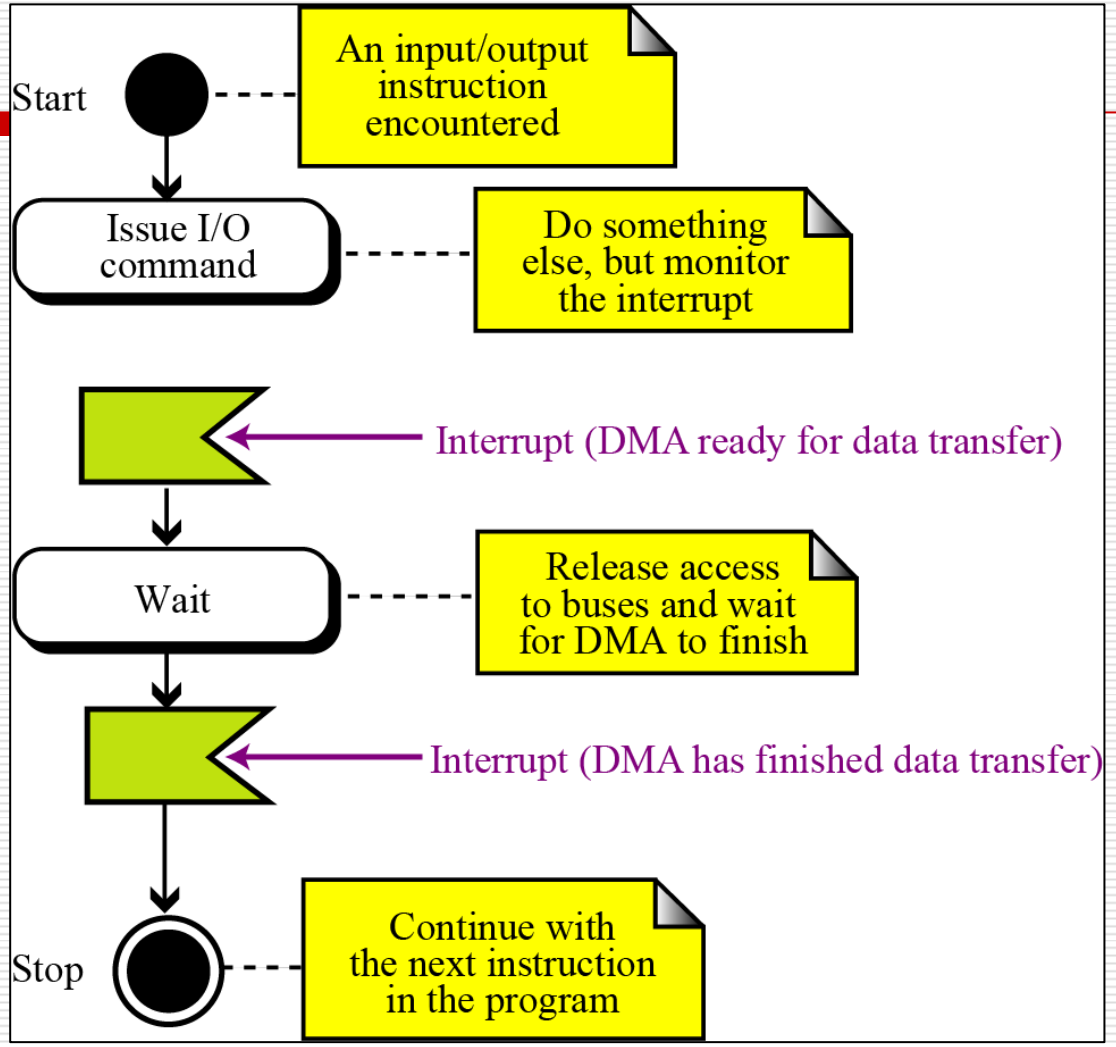

Figure 5.23 DMA input/output

5.6

DIFFERENT ARCHITECTURES

Description

- The architecture and organization of computers has gone through many changes in recent decades.
  - **CISC**
  - **RISC**
  - **Pipelining**
  - **Parallel processing**

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.

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.

Pipelining

- A computer uses three phases, fetch, decode and execute, for each instruction.
- In early computers, these three phases needed to be done in series for each instruction.
- A technique called **pipelining** to improve the throughput
  - The control unit can do two or three of these phases simultaneously, the next instruction can start before the previous one is finished.

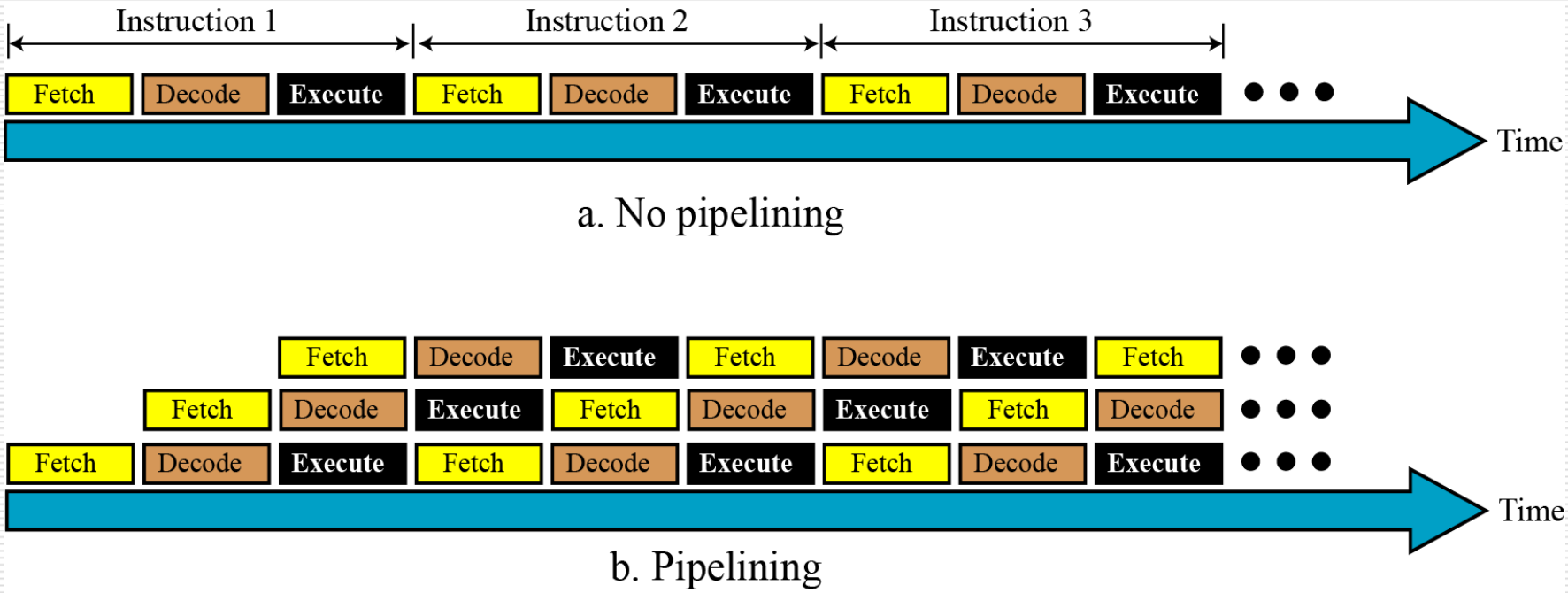

Figure 5.24 Pipelining

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, a single computer has multiple control units, multiple arithmetic logic units and multiple memory units, today.

Parallel processing

- This idea is referred to as **parallel processing**. Like pipelining, parallel processing can improve throughput.

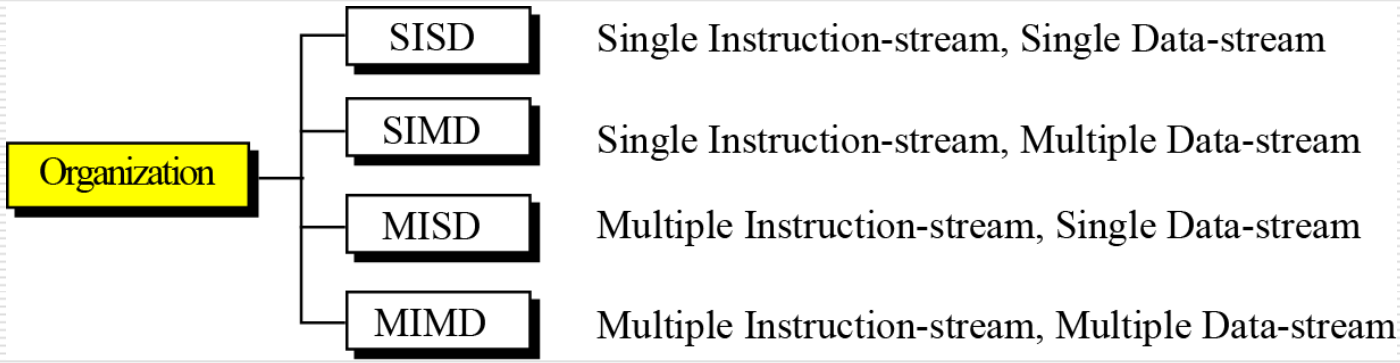

Figure 5.24 **A taxonomy of computer organization**

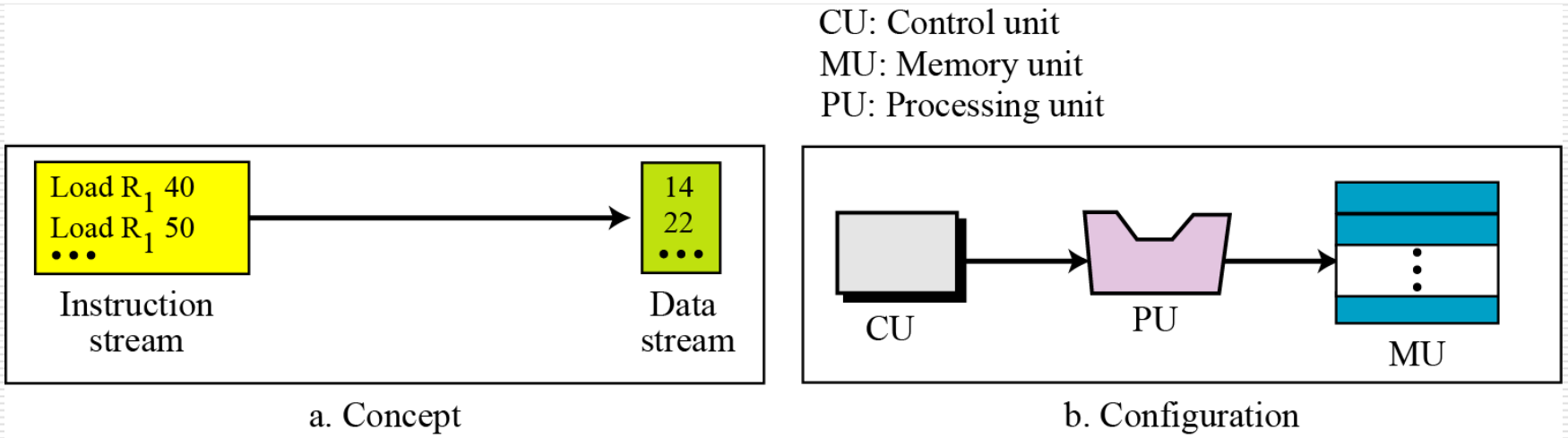

Figure 5.26 SISD organization

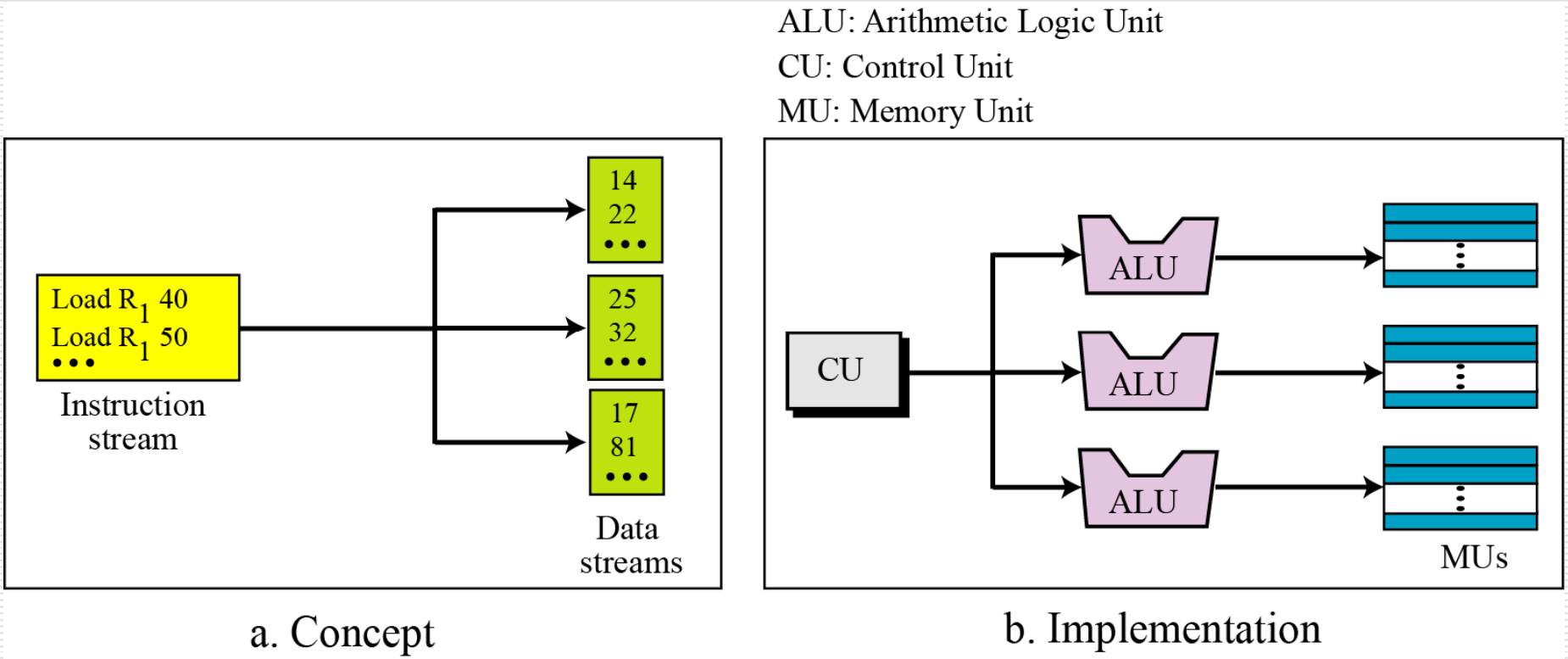

Figure 5.27 SIMD organization

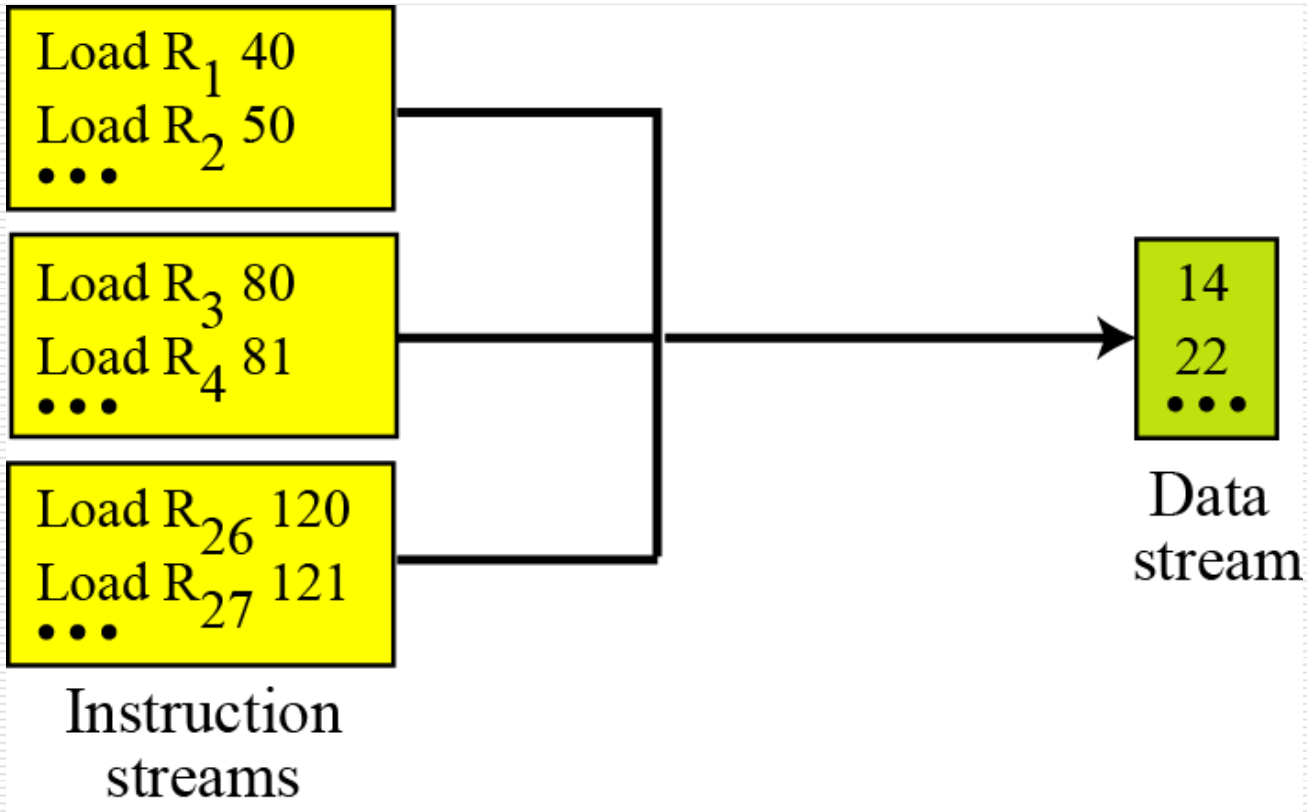

Figure 5.28 MISD organization

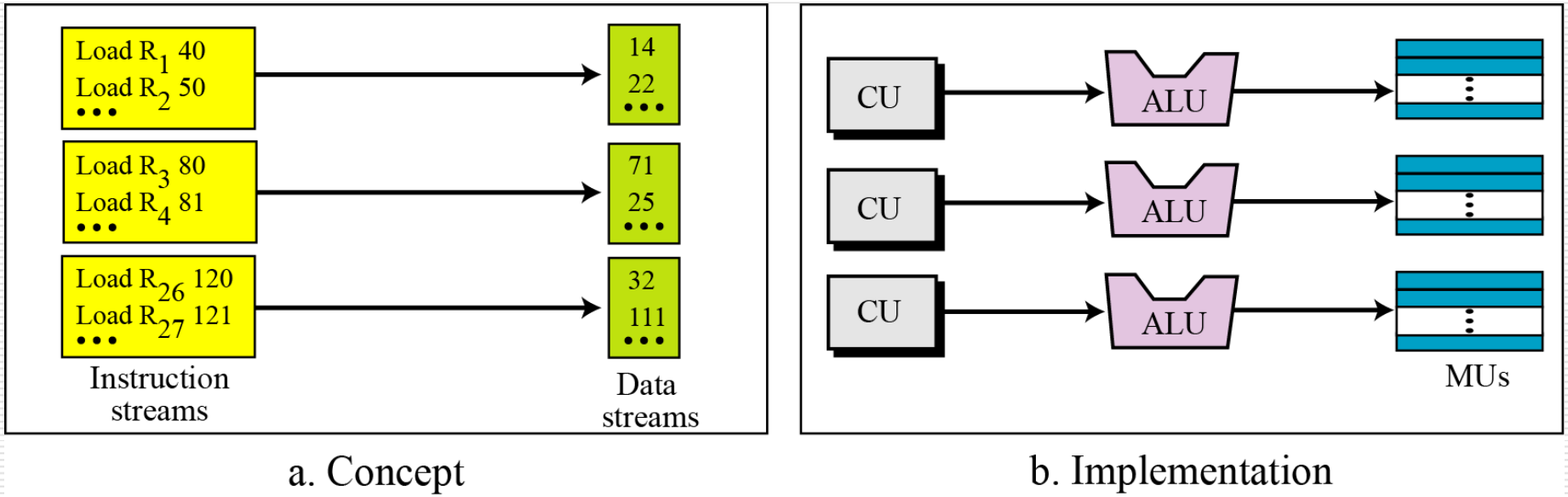

Figure 5.29 MIMD organization

5.7

A SIMPLE COMPUTER

A simple computer

- A simple (unrealistic) computer, as shown in Figure 5.30.
- The simple computer has three components: CPU, memory, and an input/output subsystem.

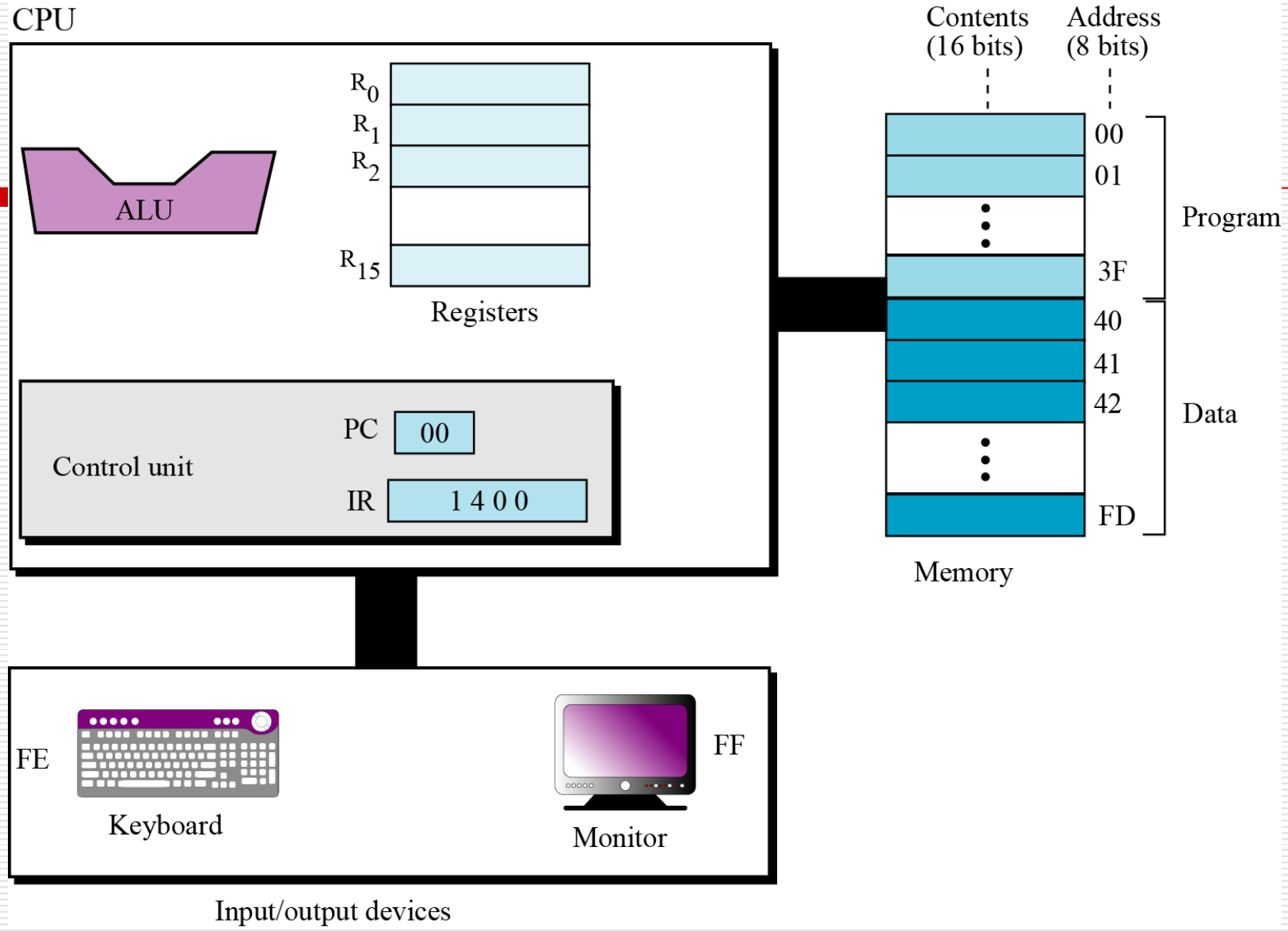

Figure 5.30 The components of a simple computer

Instruction set

- The simple computer is capable of having a set of 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).

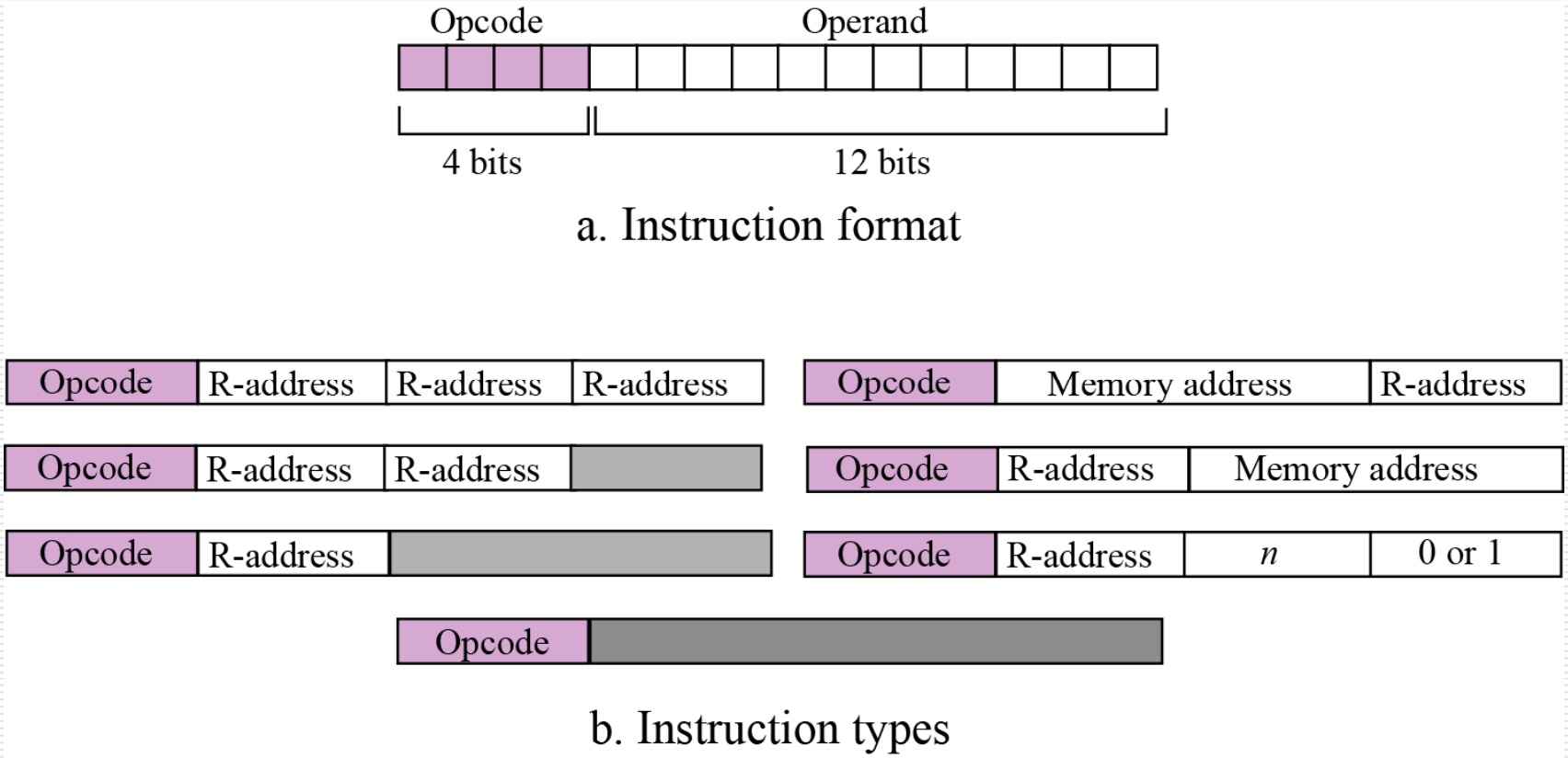

Figure 5.31 **Format and different instruction types**

Processing the instructions

- 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 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.

Processing the instructions

- 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.

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

An example

- The computer can add two integers A and B and create the result as C. We assume that integers are in two's complement format. Mathematical show: $C = A + B$

An example

- 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:

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.

An example

- In the language of The 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			

An example

- **Storing program and data**
  - Store the five-line program in memory starting from location $(00)_{16}$ to $(04)_{16}$.
  - Data needs to be stored in memory locations $(40)_{16}$, $(41)_{16}$, and $(42)_{16}$.
- **Cycles**
  - One cycle per instruction.
  - If we have a small program with five instructions, we need five cycles.
  - Add: 161 + 254 = 415. The numbers are shown in memory in hexadecimal is, $(00A1)_{16}$, $(00FE)_{16}$, and $(019F)_{16}$.

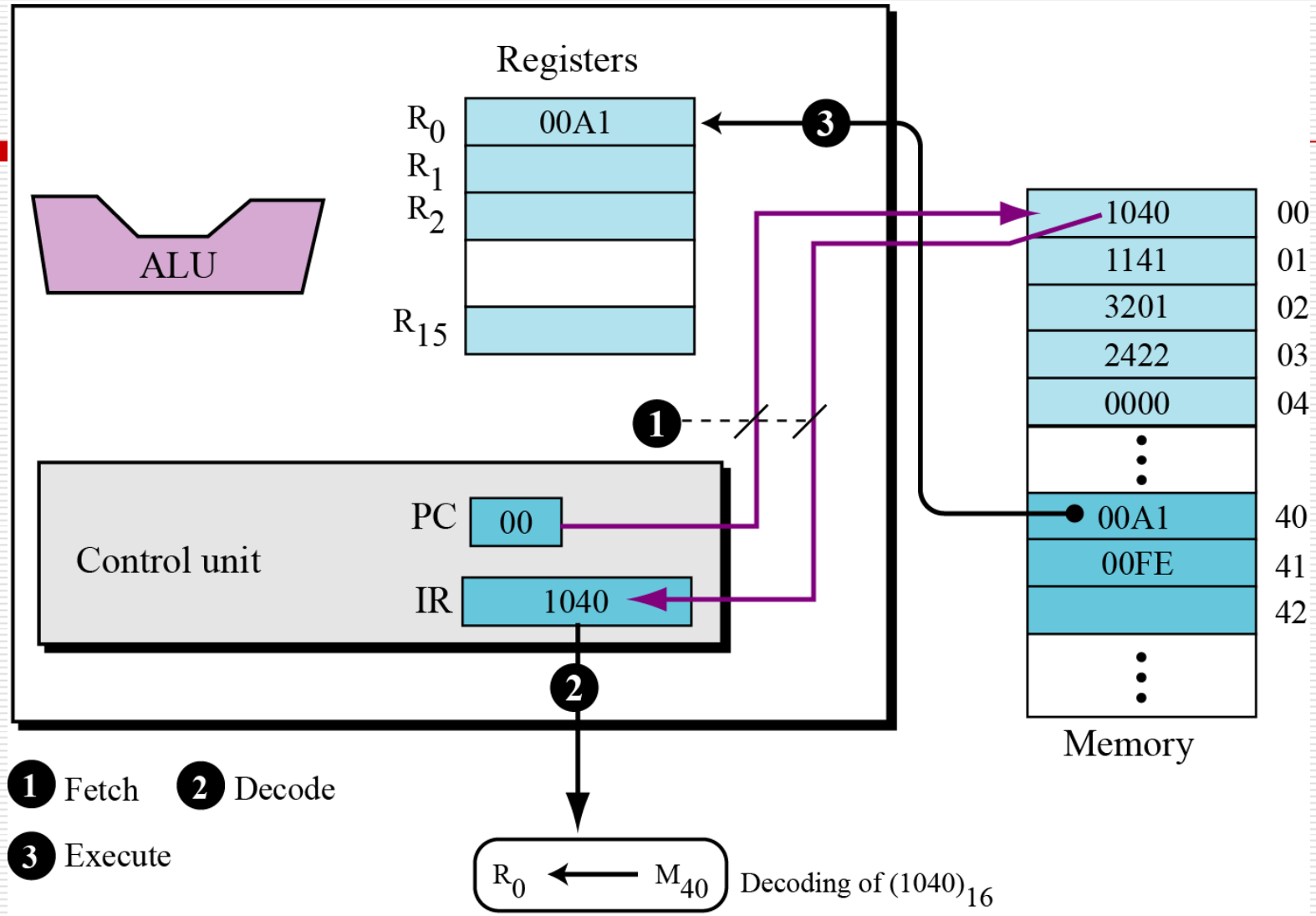

Figure 5.32 Status of cycle 1

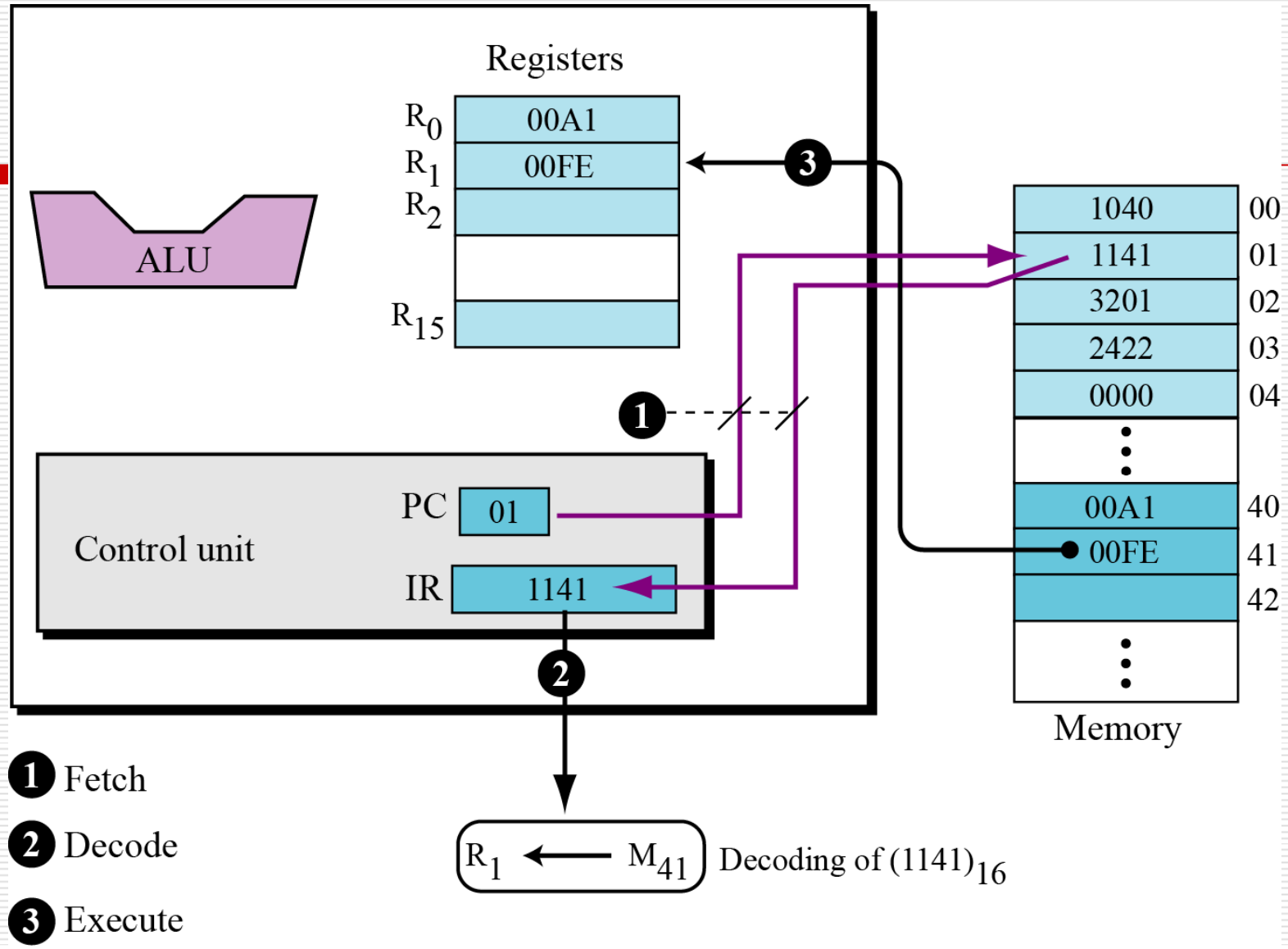

Figure 5.33 Status of cycle 2

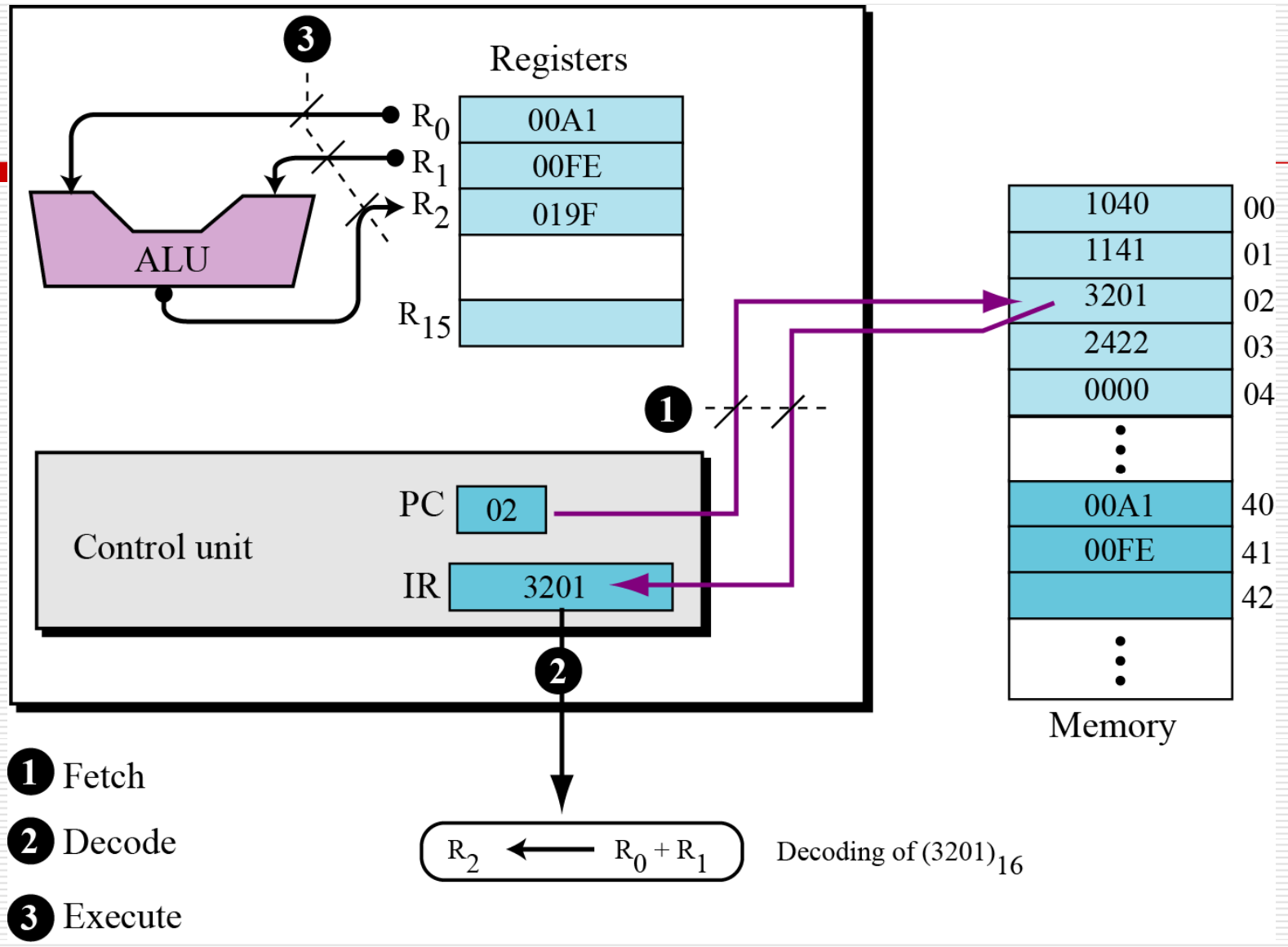

Figure 5.34 Status of cycle 3

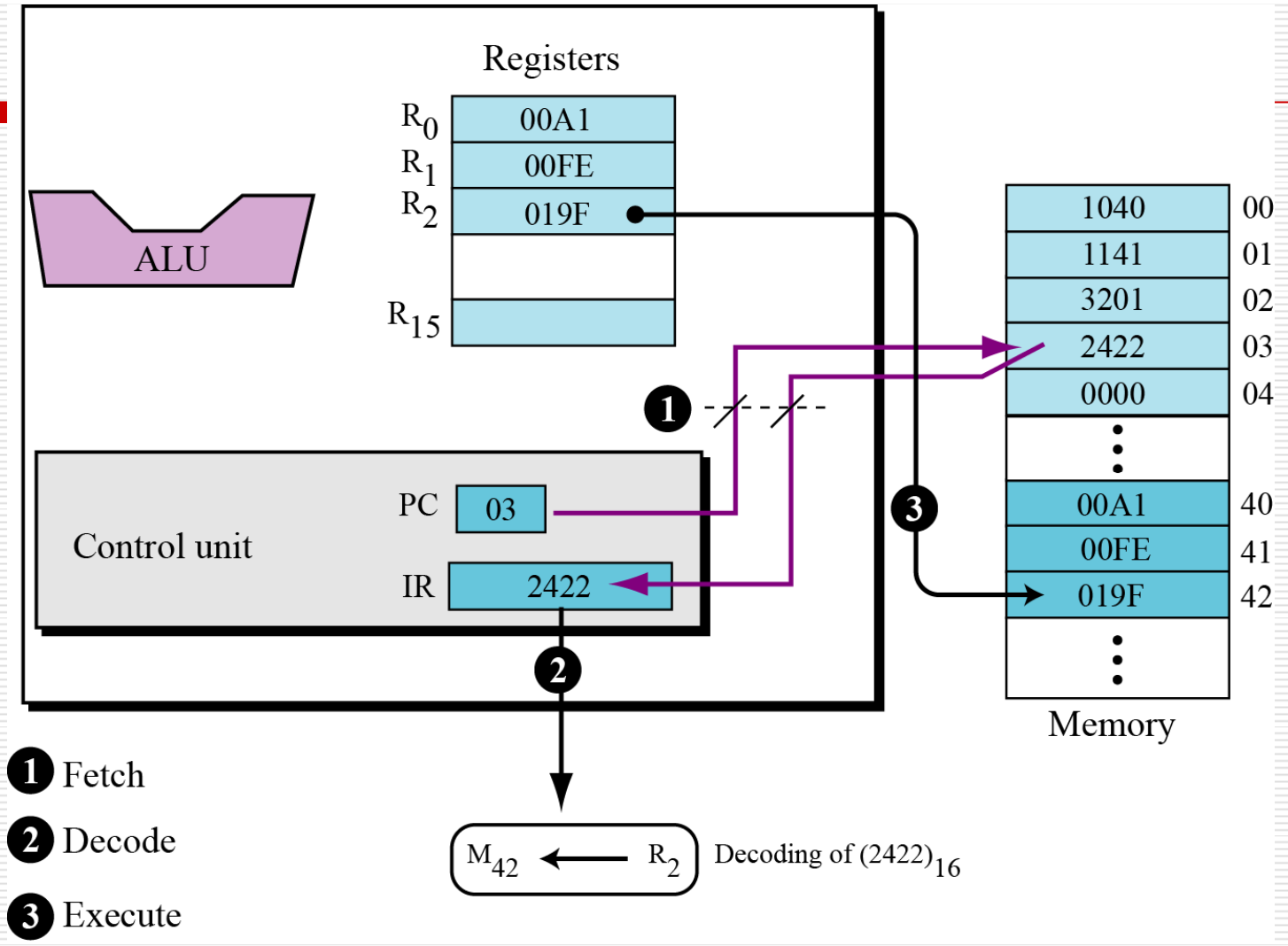

Figure 5.35 Status of cycle 4

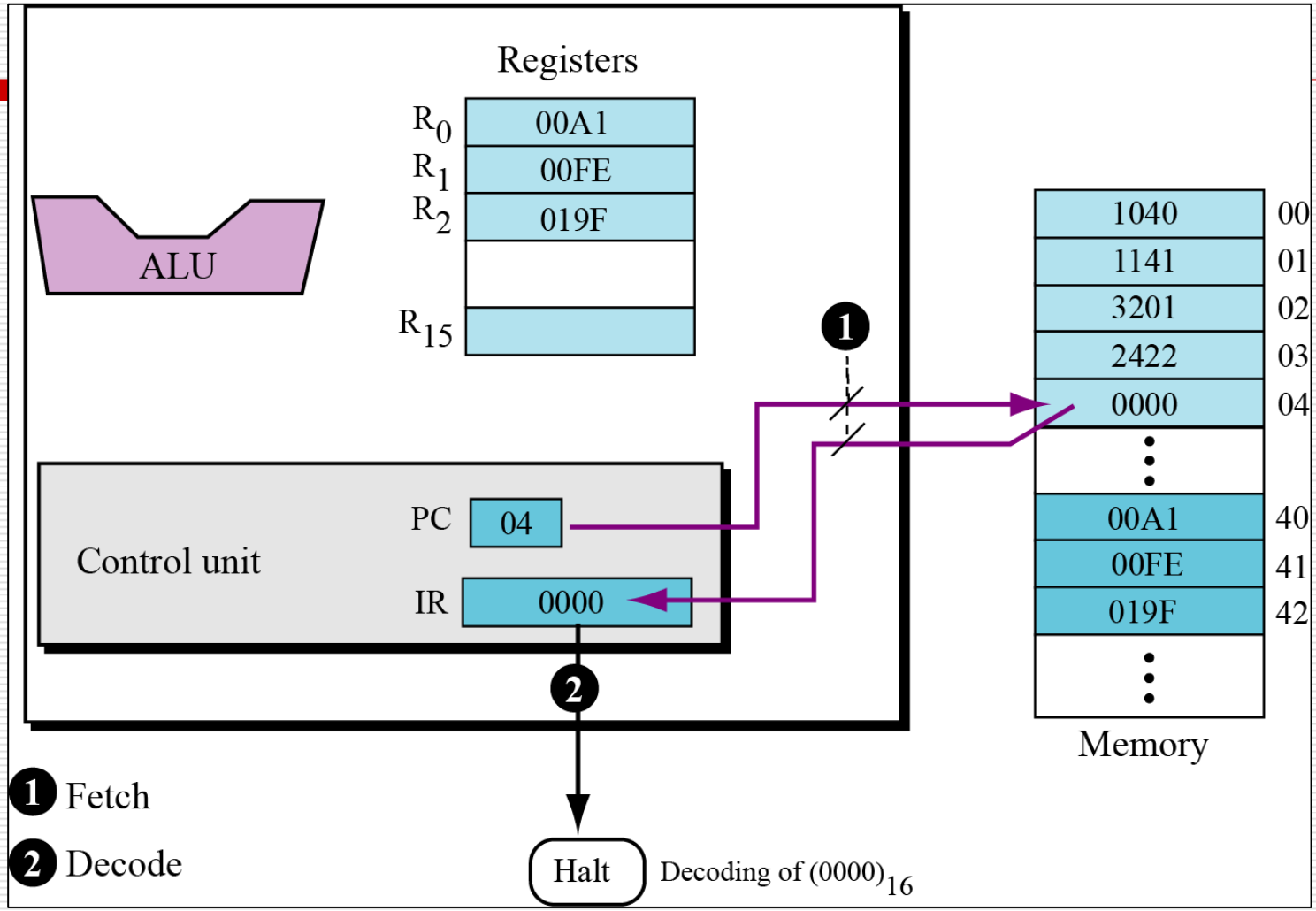

Figure 5.36 Status of cycle 5

Another example

- 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.

Another example

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.

- Read and Write operations using the LOAD and STORE instruction.
- Read data input to the CPU and write data from the CPU.

Another example

- Two instructions to read data into memory or write data out of memory.
- The read operation is:

$R \leftarrow M_{EF}$ 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.

Another example

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.