

Computer Organization Continue

Eng. Eman Hamdy

جامعة الجميع الذكية

تسعى لتكون الكيان الأول الذي يتبنى مبدأ المجتمع التعليمي
التعاوني ويخدم العلم مقابل العمل للتحفة وبناء المجتمعات

IT111 - Introduction to Information Technology



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

Objective

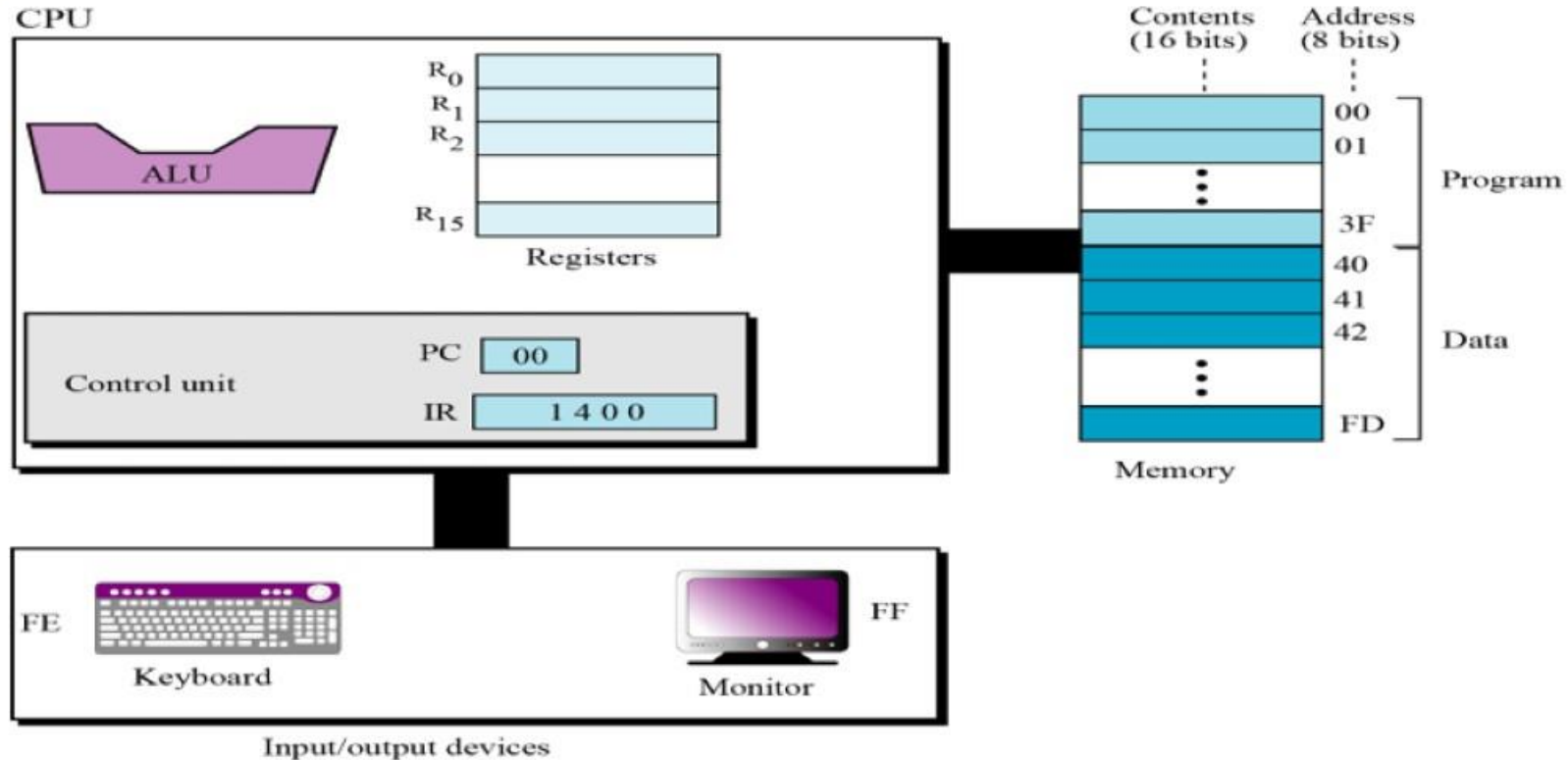
After studying this chapter, the student should be able to:

- ☐ Simple Computer
- ☐ Distinguish the two major trends in the design of computers.
- ☐ Understand how computer throughput can be improved using pipelining and parallel processing.

5-8 A SIMPLE COMPUTER

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

The Components of a Simple Computer



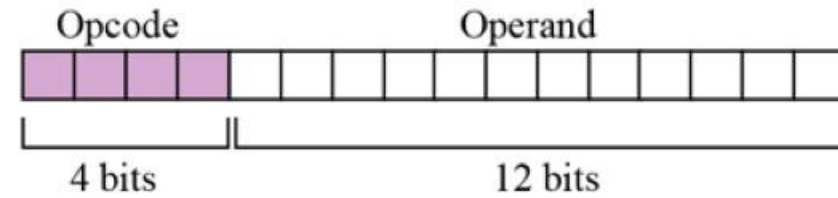
0 1 1 0 1 1 0 0 0 1 0 1 0 1 0 1

One Register (16 bits)

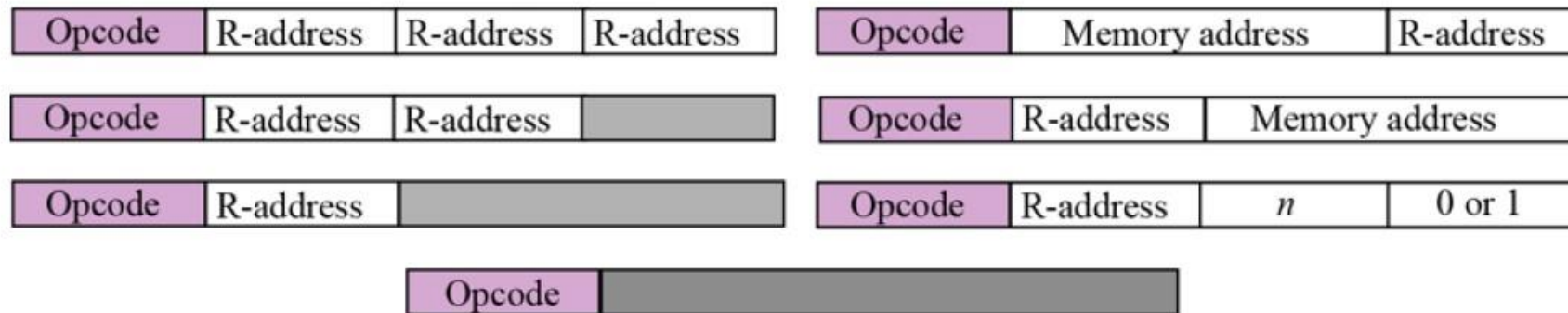
Instruction set

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



a. Instruction format



b. 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.

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 ← M _S	
STORE	2	M _D		R _S	M _D ← R _S	
ADDI	3	R _D	R _{S1}	R _{S2}	R _D ← R _{S1} + R _{S2}	
ADDF	4	R _D	R _{S1}	R _{S2}	R _D ← R _{S1} + R _{S2}	
MOVE	5	R _D	R _S		R _D ← R _S	
NOT	6	R _D	R _S		R _D ← $\overline{R_S}$	
AND	7	R _D	R _{S1}	R _{S2}	R _D ← R _{S1} AND R _{S2}	
OR	8	R _D	R _{S1}	R _{S2}	R _D ← R _{S1} OR R _{S2}	
XOR	9	R _D	R _{S1}	R _{S2}	R _D ← R _{S1} XOR R _{S2}	
INC	A	R			R ← R + 1	
DEC	B	R			R ← R – 1	
ROTATE	C	R	n	0 or 1	Rot _n R	
JUMP	D	R	n		IF R ₀ ≠ 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

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:

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} ($M_{42} \leftarrow R_2$).
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 data

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 $(42)_{16}$.

$(1040)_{16}$	00
$(1141)_{16}$	01
$(3201)_{16}$	02
$(2422)_{16}$	03
$(0000)_{16}$	04

0000	40
0000	41
0000	42

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 $(019F)_{16}$.

$(1040)_{16}$	00
$(1141)_{16}$	01
$(3201)_{16}$	02
$(2422)_{16}$	03
$(0000)_{16}$	04

00A1	40
00FE	41
019F	42



Figure 5.32 Status of cycle 1

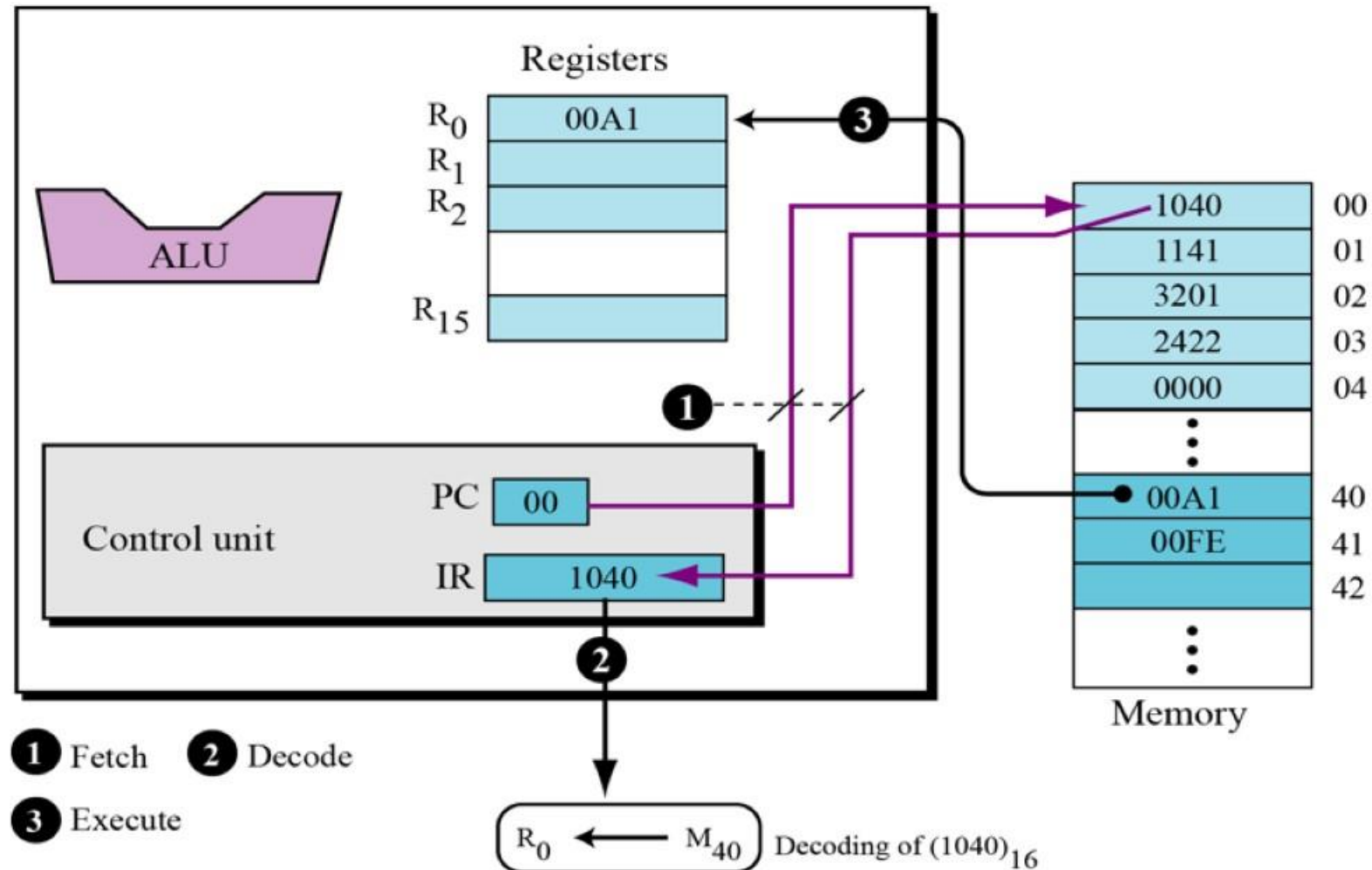


Figure 5.33 Status of cycle 2

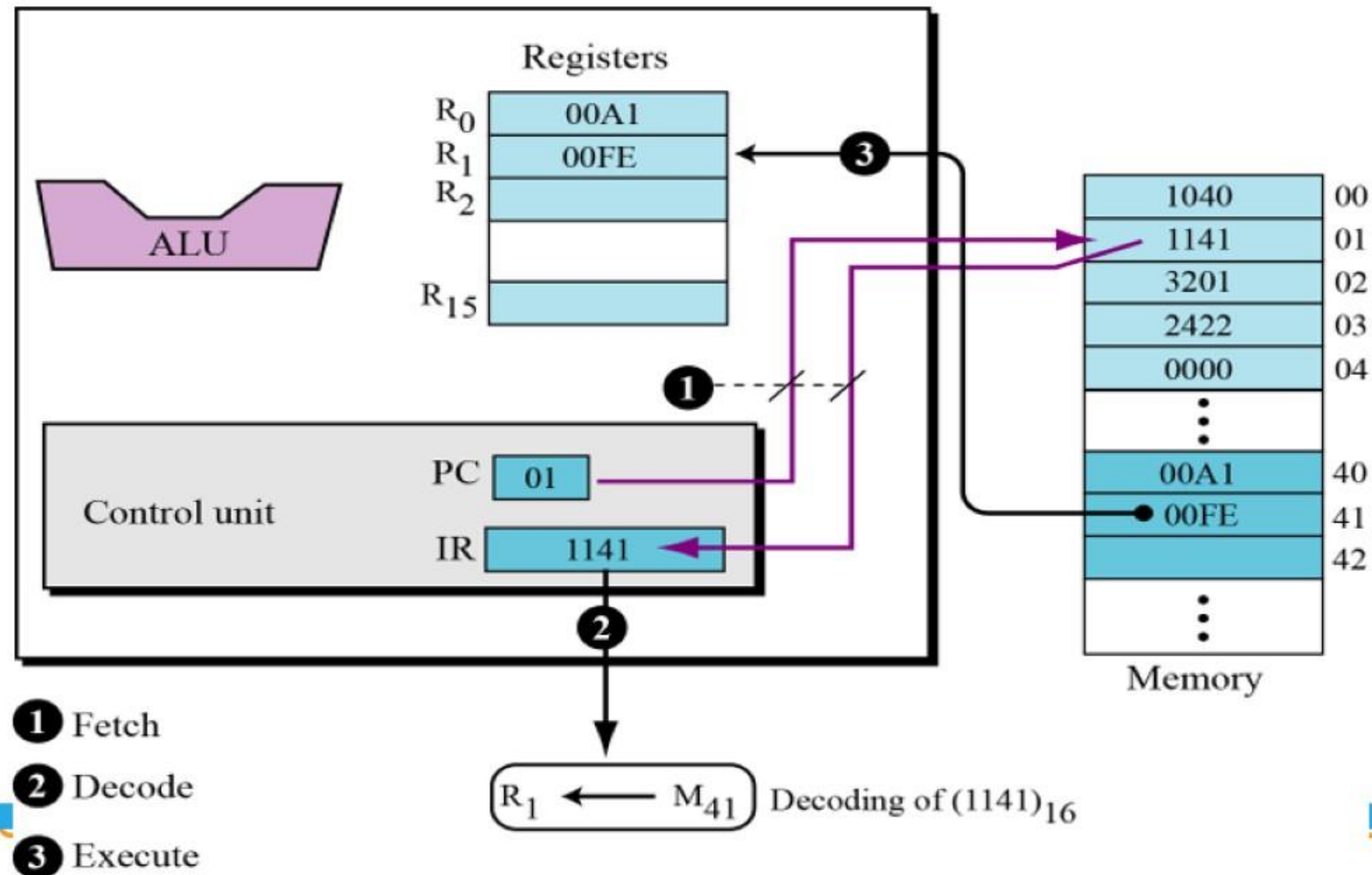


Figure 5.33 Status of cycle 3

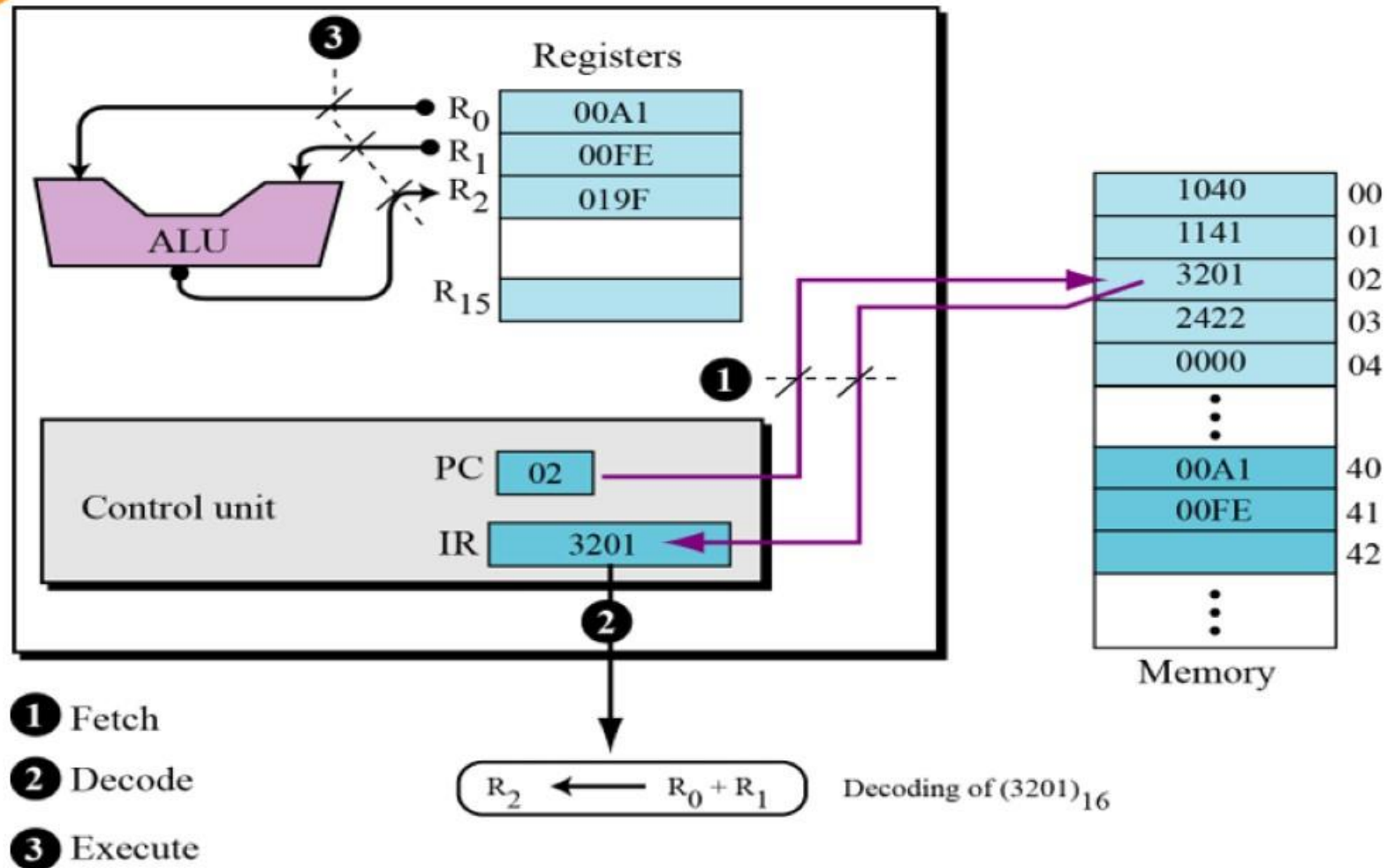


Figure 5.33 Status of cycle 4

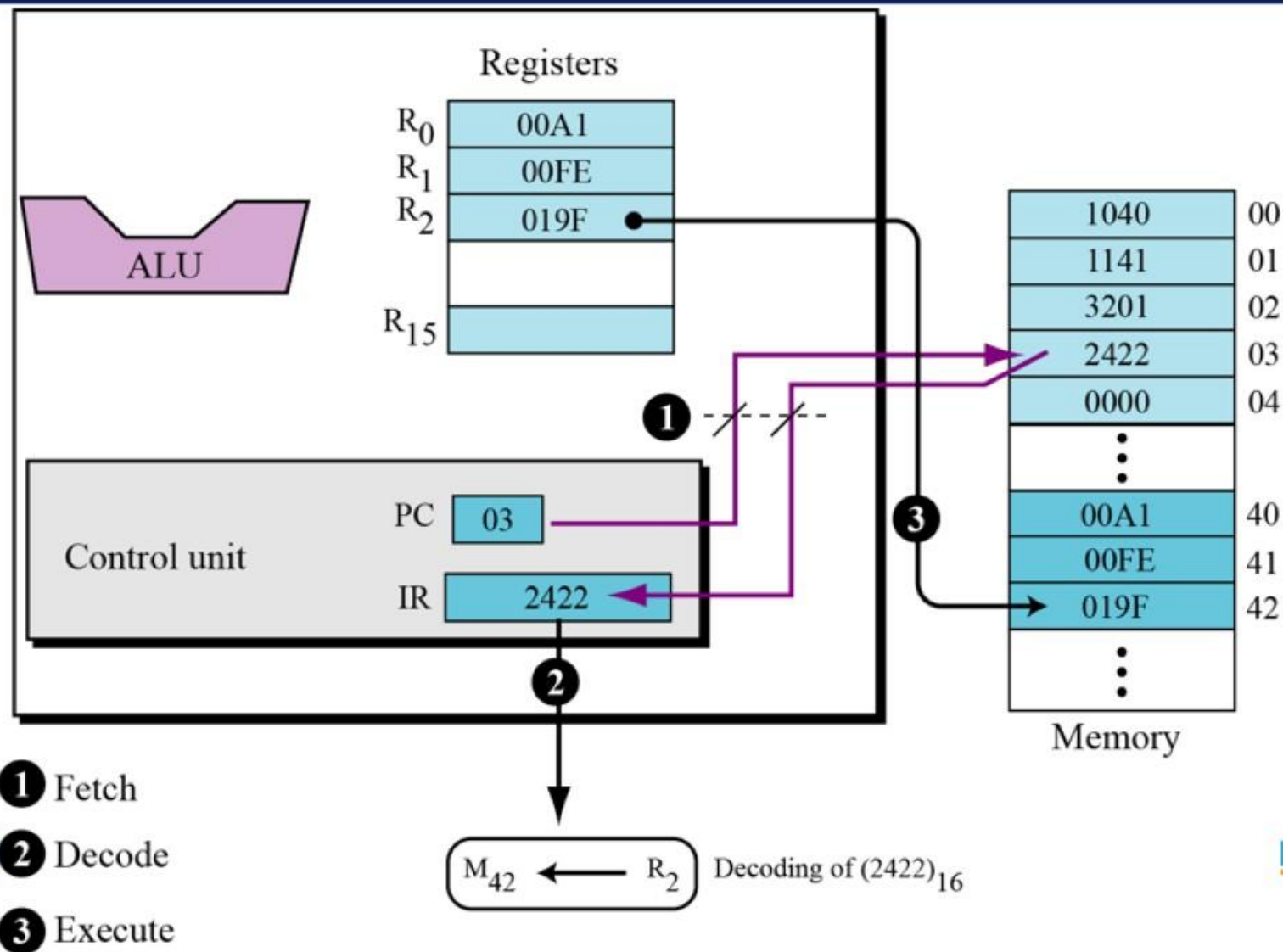


Figure 5.33 Status of cycle 5

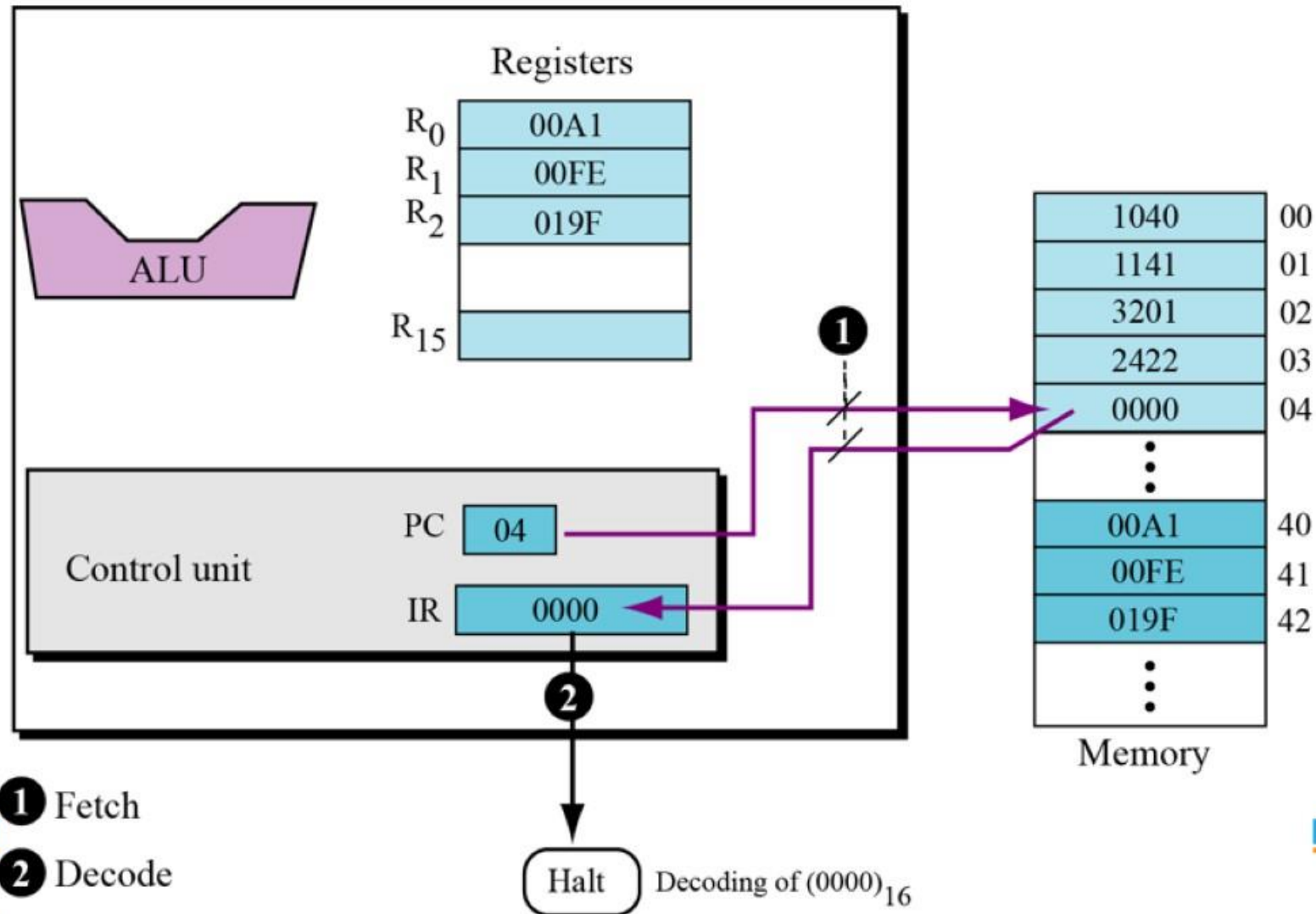


Figure 5.33 Status of cycle 4

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:



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. $M_{42} \leftarrow R_2$.
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:

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

The write operation is:

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

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.

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.

5-7 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.

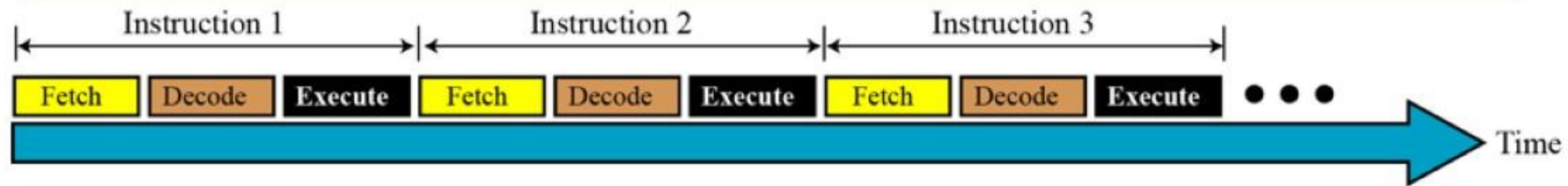
5.7.2 RISC

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.

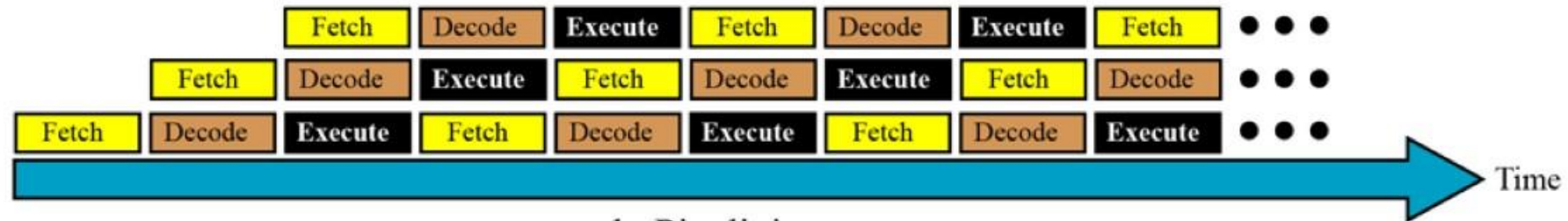
5.7.3 Pipelining

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



a. No pipelining

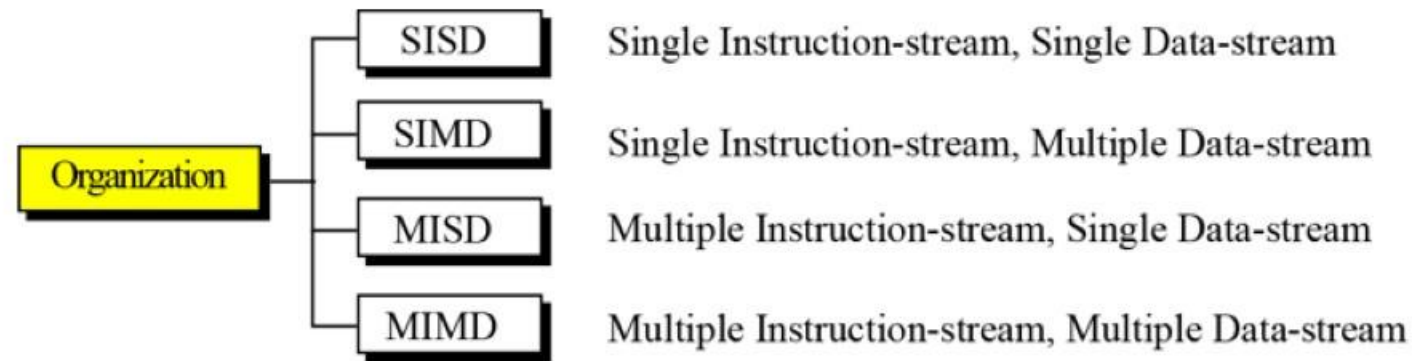


b. Pipelining

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.

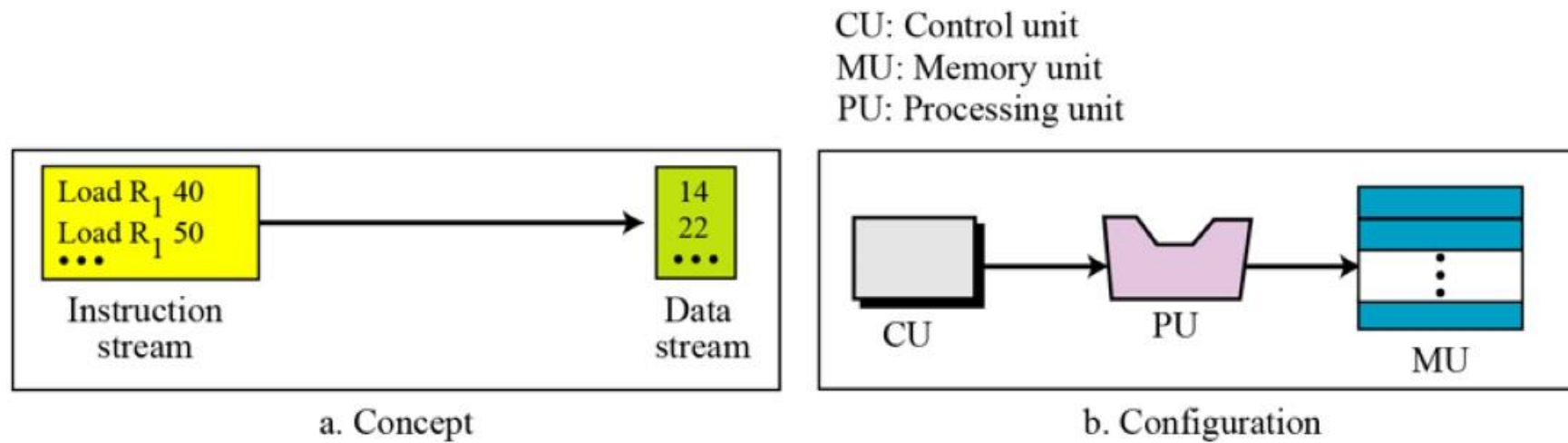
Figure 5.24 A taxonomy of computer organization



SISD organization

A **single instruction-stream, single data-stream (SISD)** organization represents a computer that has one control unit, one arithmetic logic unit, and one multiple memory units. The instructions are executed sequentially, and each instruction may access one or more data items in the data stream. Our simple computer introduced earlier in the chapter is an example of SISD organization.

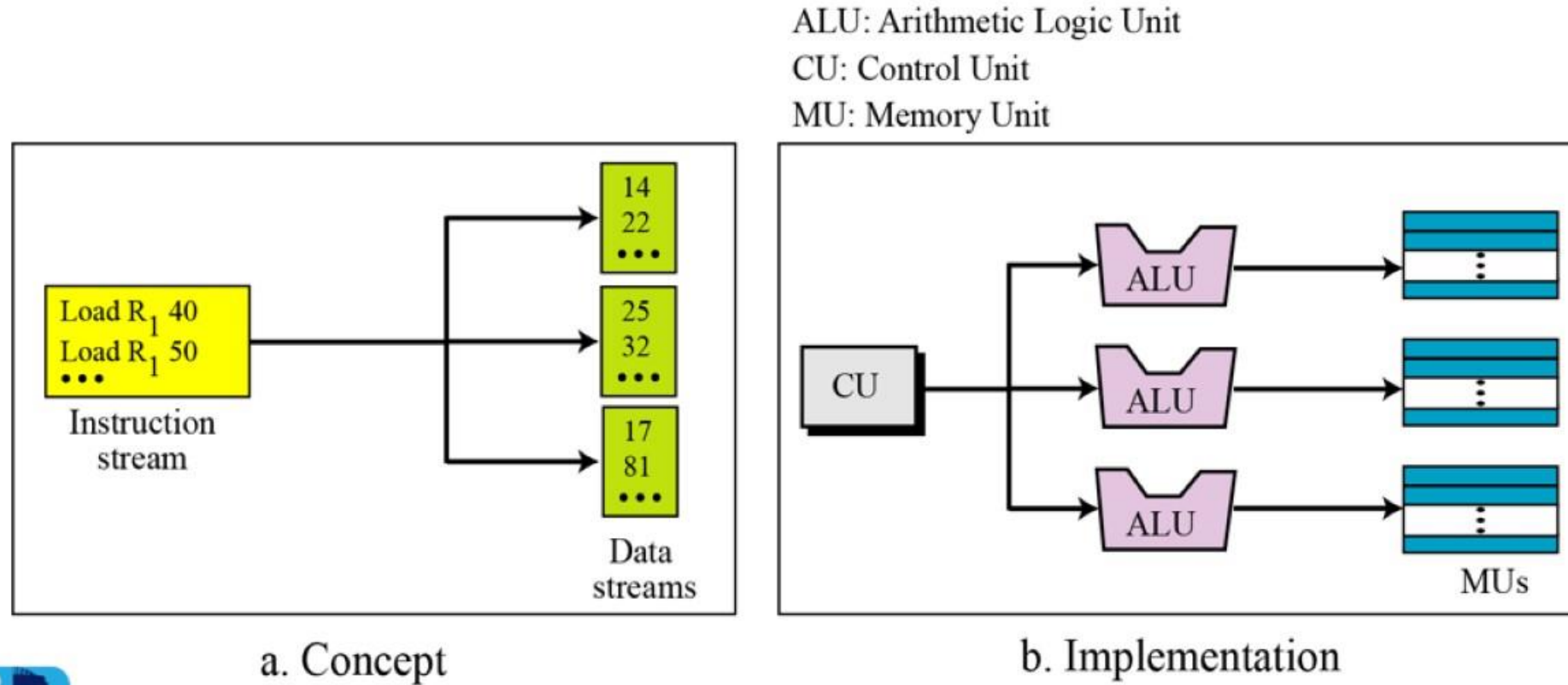
Figure 5.26 SISD organization



SIMD organization

A **Single Instruction-stream, Multiple Data-stream (SIMD) organization** represents a computer that has one control unit, multiple processing units, and multiple memory units. All processor units receive the same instruction from the control unit but operate on different items of data. An array processor that simultaneously operates on an array of data belongs to this category.

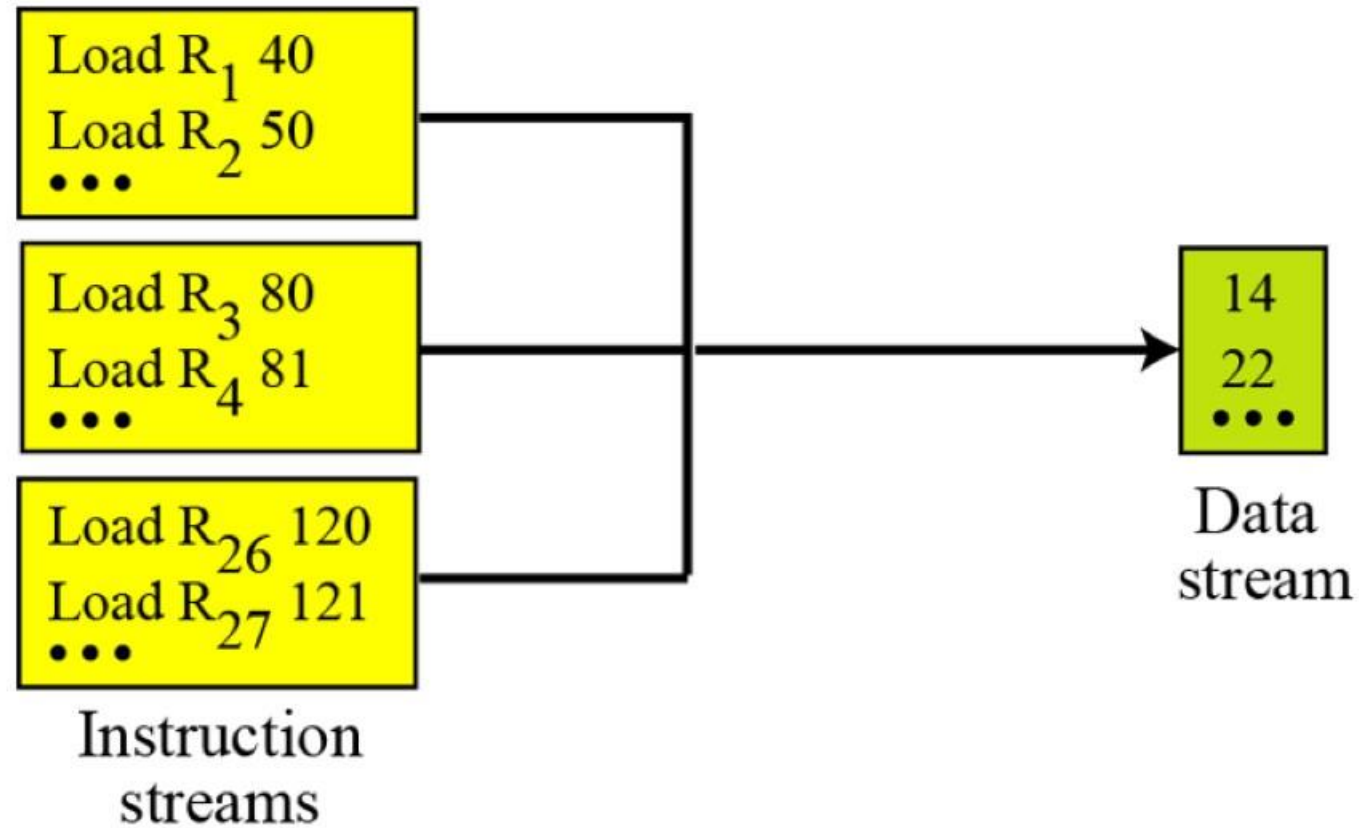
Figure 5.27 SIMD organization



MISD organization

A multiple instruction-stream, single data-stream (MISD) architecture is one in which several instructions belonging to several instruction streams simultaneously operate on the same data stream.

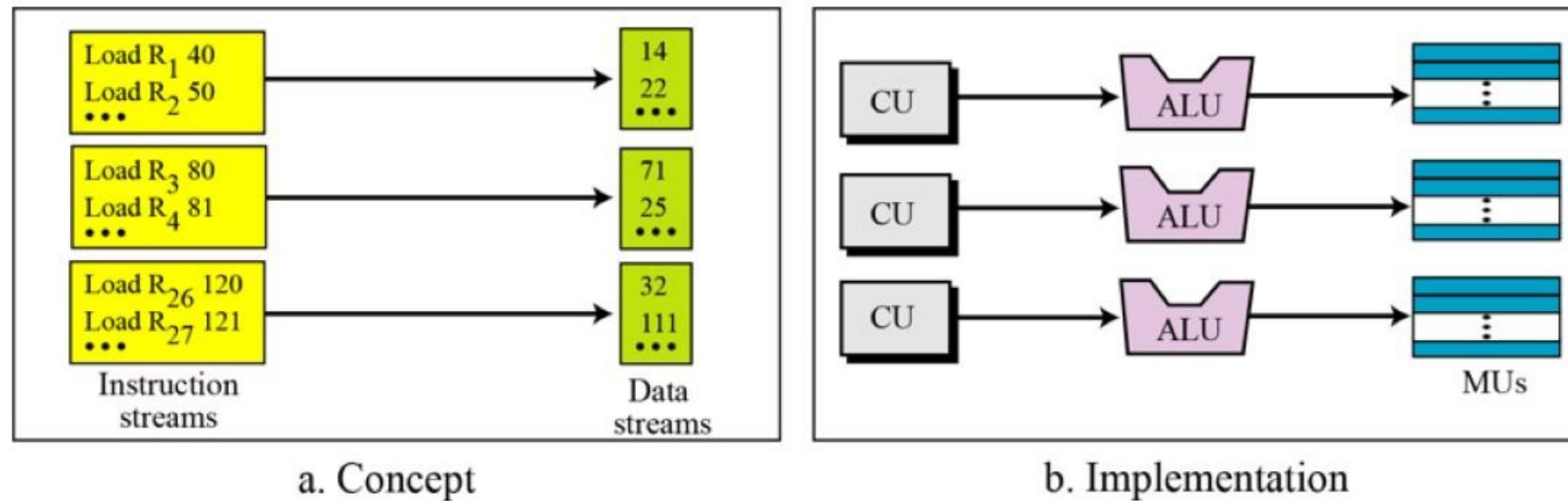
Figure 5.28 MISD organization



MIMD organization

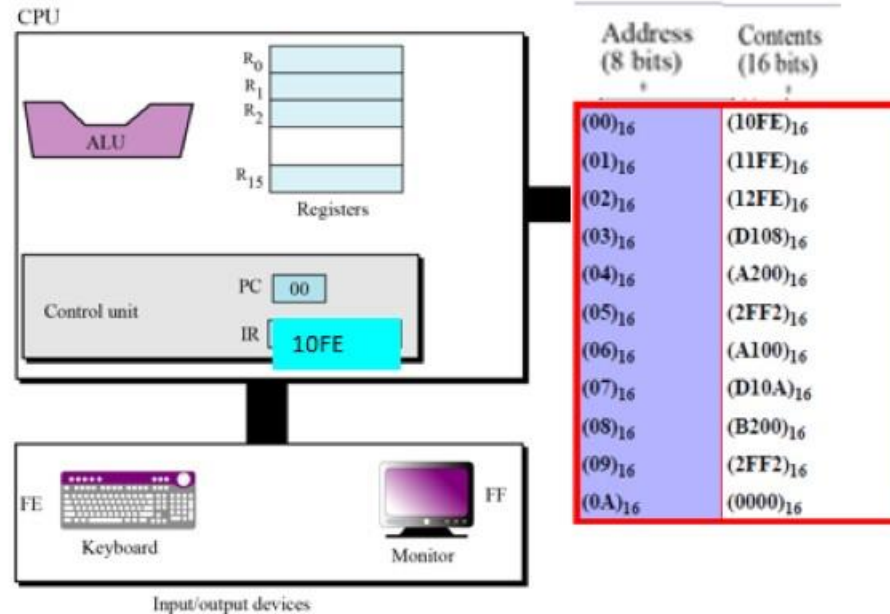
A **multiple instruction-stream, multiple data-stream (MIMD) architecture** is one in which several instructions belonging to several instruction streams simultaneously operate on several data streams (each instruction on one data stream). MIMD organization is considered as a true parallel processing architecture by some experts. In this architecture several tasks can be performed simultaneously. The architecture can use a single shared memory or multiple memory sections.

Figure 5.29 MIMD organization



Example

(00) ₁₆	(10FE) ₁₆
(01) ₁₆	(11FE) ₁₆
(02) ₁₆	(12FE) ₁₆
(03) ₁₆	(D108) ₁₆
(04) ₁₆	(A200) ₁₆
(05) ₁₆	(2FF2) ₁₆
(06) ₁₆	(A100) ₁₆
(07) ₁₆	(D10A) ₁₆
(08) ₁₆	(B200) ₁₆
(09) ₁₆	(2FF2) ₁₆
(0A) ₁₆	(0000) ₁₆



What is the output if we input 3 integer: 1, 2, 3

Solution:
Output = 2

Table 5.4 List of instructions for the simple computer

Instruction	Code				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 ← M _S
STORE	2	M _D		R _S	M _D ← R _S
ADDI	3	R _D	R _{S1}	R _{S2}	R _D ← R _{S1} + R _{S2}
ADDF	4	R _D	R _{S1}	R _{S2}	R _D ← R _{S1} + R _{S2}
MOVE	5	R _D	R _S		R _D ← R _S
NOT	6	R _D	R _S		R _D ← $\bar{R}_S$
AND	7	R _D	R _{S1}	R _{S2}	R _D ← R _{S1} AND R _{S2}
OR	8	R _D	R _{S1}	R _{S2}	R _D ← R _{S1} OR R _{S2}
XOR	9	R _D	R _{S1}	R _{S2}	R _D ← R _{S1} XOR R _{S2}
INC	A	R			R ← R + 1
DEC	B	R			R ← R - 1
ROTATE	C	R	n	0 or 1	Rot _n R
JUMP	D	R	n		IF R ₀ = 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
 d₁, d₂, d₃, d₄: First, second, third, and fourth hexadecimal digits



Summary

- ☐ **Simple Computer**
- ☐ **Distinguish the two major trends in the design of computers.**
- ☐ **Understand how computer throughput can be improved using pipelining and parallel processing.**

Thank You 😊