



Professional Diploma of Software Engineering

Computer Systems Principles & Programming (SE101)


Lecture 2

Signed Binary Fractional Numbers Representation

- Numbers used in scientific calculations are designed by a sign, by the magnitude of the number, and by the position of the radix point.
- The position of the radix point is required to represent fractions, integers, or mixed integer-fraction numbers.
- There are two ways of specifying the position of the radix point which are:
 1. Fixed-point representation (which we have studied up till now) .
 2. Floating-point representation.

Fixed-Point Representation

Old position	7	6	5	4	3	2	1	0
New position	4	3	2	1	0	-1	-2	-3
Bit pattern	1	0	0	1	1	1	0	1
Contribution	2^4			2^1	2^0	2^{-1}	2^{-2}	2^{-3}
<hr/>								
								$=19.625$



Radix Point

■ **Limitation of Fixed-Point Representation:**

To represent very large or very small numbers we **need a very long sequences of bits**, this is because we have to give bits **to both the integer part and the fraction part**.

Floating-Point Representation

■ In Floating-point representation, there are two ways for positioning the radix point:

- 1- Putting the radix point at the extreme left of the number.
- 2- Putting the radix point at the extreme right of the number.

■ According to the first way, any number can be expressed as a fraction (mantissa) and a positive exponent. (e.g., $255.489 = 0.255489 * 10^{+3}$).

■ According to the second way, any number can be expressed as an integer and a negative exponent. (e.g., $255.489 = 255489.0 * 10^{-3}$).

Floating-Point Representation (2)

In decimal system, the floating-point notation; which is also called “scientific notation”; of any real number can be represented as:

$$(N)_{10} = .F \times 10^E$$

$(N)_{10}$: is the real number in decimal.

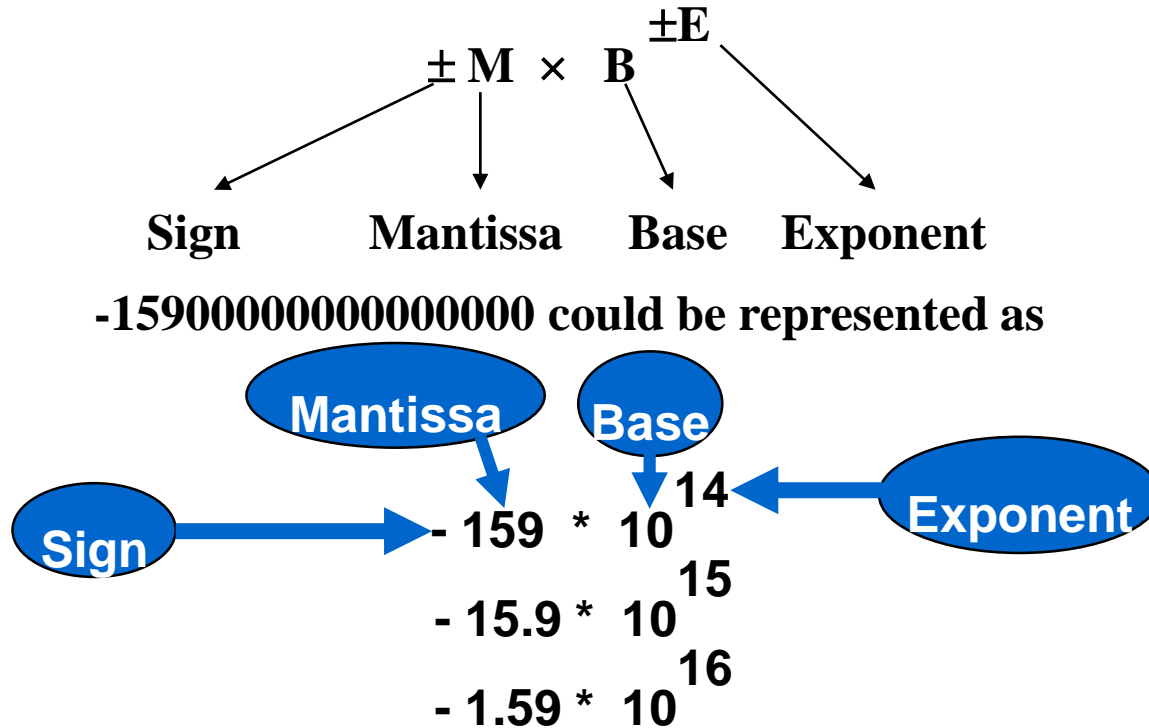
F : is a fraction(mantissa).

E : is an exponent.

Example: The real number **+6132.789** can be expressed using floating-point notation:

Fraction (F)	Exponent (E)
+ . 6132789	+ 04

Floating Point Representation (3)



Number	Scientific Notation	Floating-Point Notation
1,245,000,000,000	1.245 x10 ^{12}	0.1245 x10 ^{13}
0.0000001245	1.245 x10 ^{-7}	0.1245 x10 ^{-6}
-0.0000001245	-1.245 x10 ^{-7}	-0.1245 x10 ^{-6}

IEEE-32Bits Single Precision Method

In this method, the single precision floating-point number is expressed using 32-bits



The **sign bit (s)** is **0** for **positive** numbers and **1** for **negative** numbers.

The **exponent part (e)** is expressed using **8-bits in excess-127 notation**, i.e., if the exponent is (+4), the exponent will be $(+4) + 127 = 131$ (10000011).

The **fraction or mantissa part (f)** is expressed using **23-bits in normalized form** such that it is **considered to be 1.f**, with **1 is hidden and appears only during conversion to decimal**, e.g., if fraction is **.01101**, then fraction would be **1.01101** when it is converted to decimal.

IEEE-32Bits Single Precision Method (2)

■ Conversion from decimal number to IEEE-32bits binary floating-point number:

1. Take the integer part of the number and generate its binary equivalent.
2. Take the fractional part and generate its binary equivalent.
3. Put the two parts together with radix point in the middle between them.
4. Normalise the whole binary number to the form (**1.f**) by moving the radix point whether to the left or to the right.
5. Calculate the decimal exponent as the number of radix point movements.
6. Represent the decimal exponent into binary in excess-127 by adding it to 127, and then convert the addition result as unsigned integer to binary.
7. Fill in the sign bit with 0 if the decimal number is positive; otherwise with 1.
8. Fill in exponent part with the binary exponent in excess-127.
9. Fill in the fraction (mantissa) part with (**.f**) normalized binary bits.

IEEE-32Bits Single Precision

Method (3)

Example: Code the decimal **+0.001275** as IEEE-32bits binary floating-point pattern.

Solution:

+ (.001275) is a positive number then the sign bit is 0.

$$+ (.001275) = + (.0000000000101) = + (1.01) \times 2^{-10}$$

The decimal exponent is -10, then its binary in excess-127 is:

$$-10 + 127 = 117 = 01110101$$

The mantissa is (1.01), then the normalised mantissa is (.01)

1-bit Sign of Fraction (f)	8-bits Exponent (e)	23-bit Fraction (f)
0	01110101	01000000000000000000000

2 X	.001275	
	.00255	0
	.0051	0
	.0102	0
	.0204	0
	.0408	0
	.0816	0
	.1632	0
	.3264	0
	.6528	0
	.3056	1
	.6112	0
	.2224	1

The binary pattern of floating-point format of $(+.001275)_{10}$ is $(0 \ 01110101 \ 01000000000000000000000)_2 = (3AA00000)_{16}$

IEEE-32Bits Single Precision

Method (4)

Example: Code the decimal **-6.75** as IEEE-32bits binary floating-point pattern.

Solution:

-(6.75) is a negative number then the sign bit is 1.

$$-(6.75) = -(110.11) = -1.1011 * 2^{+2}$$

The decimal exponent is +2, then its binary in excess-127 is:

$$+2 + 127 = 129 = 10000001.$$

The mantissa is (1.1011), then the normalised mantissa is (.1011)

1-bit Sign of Fraction (f)	8-bits Exponent (e)	23-bit Fraction (f)
1	10000001	10110000000000000000000

The binary pattern of floating-point format of $(-6.75)_{10}$ is
 $(1\ 10000001\ 10110000000000000000000)_2 = (C0D80000)_{16}$

IEEE-32Bits Single Precision Method (5)

Possible Representations of Exponent

Binary	Unsigned Integer	Excess-127
00000000	0	-127 {reserved}
00000001	1	-126
00000010	2	-125
01111110	126	-1
01111111	127	0
10000000	128	1
10000001	129	2
11111110	254	127
11111111	255	128 {reserved}

IEEE-32Bits Single Precision Method (6)

The maximum value (11111111) and the minimum value (00000000) of the e-field are reserved for the following exceptional conditions:

1. $e = 255$ and $f = 0$

In this case, the number represents plus or minus infinity.

2. $e = 255$ and $f \neq 0$

In this case, the representation is considered to be "not a number", i.e. NaN, regardless of the sign value. NaNs are used to signify invalid operations such as multiplication of zero with infinity.

3. $e = 0$ and $f = 0$

In this case, the number denotes plus or minus zero.

4. $e = 0$ and $f \neq 0$

In this case, the number has a magnitude which is less than the allowed minimum value.

IEEE-32Bits Single Precision Method (7)

Reserved Special Cases:

Positive Zero

0 00000000 000000000000000000000000

Negative Zero

1 00000000 000000000000000000000000

Positive Infinity

0 11111111 000000000000000000000000

Negative Infinity

1 11111111 000000000000000000000000

Positive Underflow

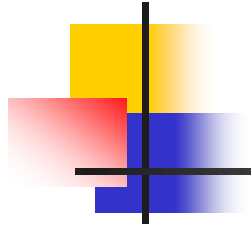
0 00000000 xxxxxxxxxxxxxxxxxxxxxxxxxxxx

Negative Underflow

1 00000000 xxxxxxxxxxxxxxxxxxxxxxxxxxxx

Not A Number (NaN)

x 11111111 xxxxxxxxxxxxxxxxxxxxxxxxxxxx



Complements



Diminished Radix Complement

$(r-1)$'s complement

- Given a number N in base r having n digits, the $(r-1)$'s complement of N is defined as $(r^n - 1) - N$.

The 9's complement of 546700 is $999999 - 546700 = 453299$

The 1's complement of 101100 is $111111 - 101100 = 010011$

- Alternatively, the 1's complement of a binary number is formed by changing each 1 to 0 and each 0 to 1
- Note: The $(r-1)$'s complement of octal or hexadecimal numbers is obtained by subtracting each digit from 7 or F (decimal 15), respectively



Radix Complement (**r 's complement**)

- Given a number N in base r having n digit, the r 's complement of N is defined as $(r^n - N)$ for $N \neq 0$ and as 0 for $N=0$.

The 10's complement of 246700 is 753300

The 2's complement of 1011000 is 0101000

- Alternatively, the r 's complement of a given number N can be obtained by first computing the $(r-1)$'s complement of N and adding 1 to it
- If the number N contains a radix point, the point should be removed temporarily to form any complement and then it is restored to the complemented number in the same position



Subtraction with Complement

- The subtraction of two n -digit unsigned numbers $M - N$ in base r can be done as follows:
 $M + (r^n - N)$, note that $(r^n - N)$ is r 's complement of N .
- If $M \geq N$, the sum will produce an end carry x , which can be discarded; what is left is the result $M - N$.
- If $M < N$, the sum does not produce an end carry and the result is $N - M$. Take the r 's complement of the sum and place a negative sign in front.



Example:

- Using 10's complement subtract $72532 - 3250$

$$M = 72532$$

$$10\text{'s complement of } N = 96750$$

$$\text{sum} = 169282$$

$$\text{Discarded end carry } 10^5 = -100000$$

$$\text{answer: } 69282$$



Example:

- Using 10's complement subtract $3250 - 72532$

$$M = 03250$$

$$10\text{'s complement of } N = 27468$$

$$\text{sum} = \textcircled{0}30718$$

$$\text{Discarded end carry } 10^5 = \underline{-100000}$$

$$\text{answer: } -(100000 - 30718) = -69282$$

- The answer is $-(10\text{'s complement of } 30718) = -69282$



Example

- Using 2's complement subtract (a) $1010100 - 1000011$

$$M = 1010100$$

$$N = 1000011, \text{ 2's complement of } N = 0111101$$

$$\begin{array}{r} 1010100 \\ + \quad 0111101 \\ \hline \text{Sum} = 10010001 \\ \text{Discarded end carry } 2^7 = -10000000 \\ \hline \text{answer: } 00010001 \end{array}$$



Example

- Using 2's complement subtract (b) $1000011 - 1010100$

$$M = 1000011$$

$$N = 1010100, \text{ 2's complement of } N = \textcolor{red}{0101100}$$

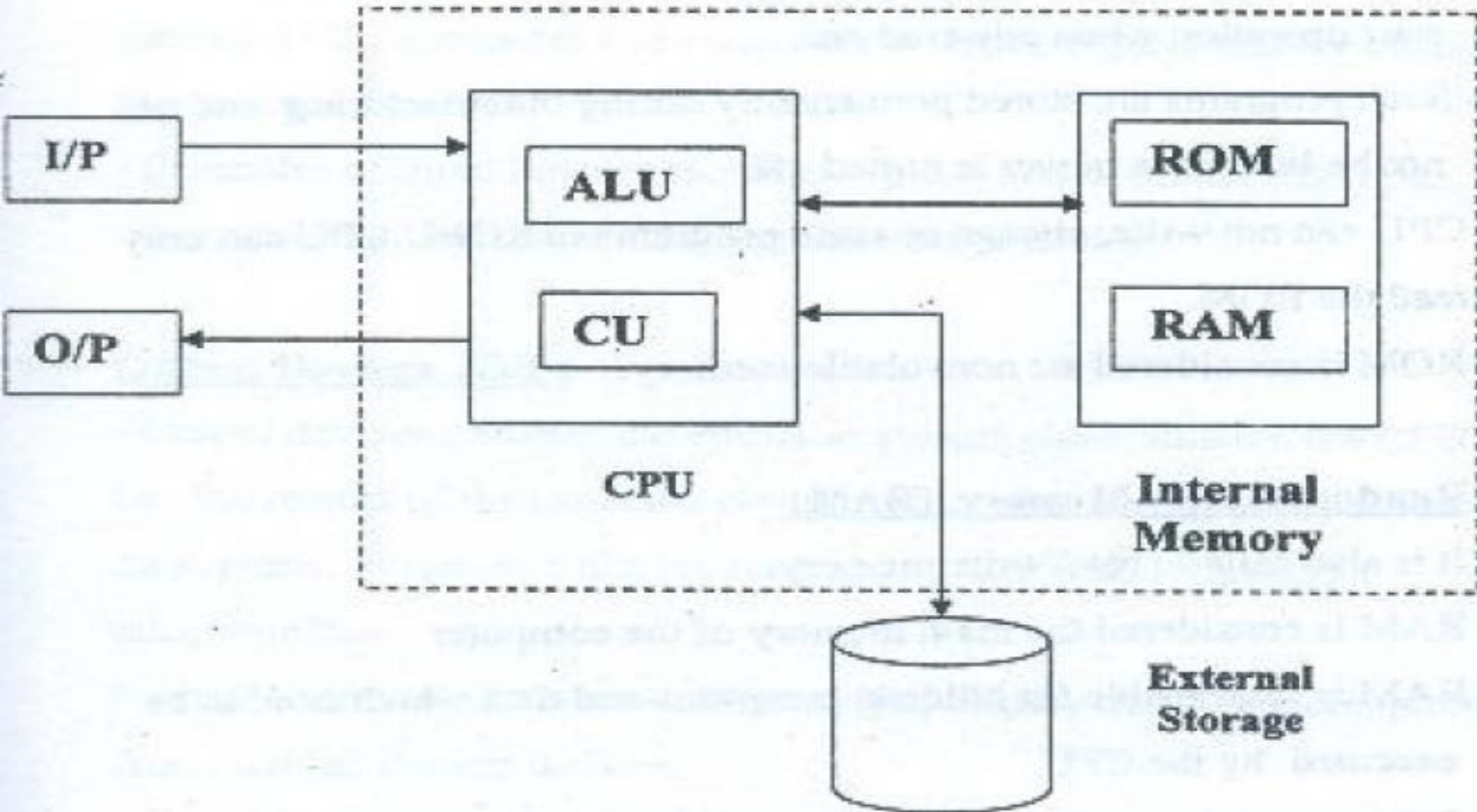
$$\begin{array}{r} 1000011 \\ + \quad \textcolor{red}{0101100} \\ \hline \text{sum} = 1101111 \end{array}$$

$$\text{answer: } - (1000000 - 1101111) = - \underline{\underline{0010001}}$$

- The answer is $-(2\text{'s complement of } 1101111) = -0010001$

Computer Hardware

Architecture of Modern Computer Machine:



Architecture of Modern Computer Machine

➤ **Central Processing Unit (CPU):** is the main unit of the computer which is responsible for executing programs stored in the internal memory. It consists of two parts:

- 1) Control Unit (CU) .
- 2) Arithmetic and Logic Unit (ALU) .

- 1) **Control Unit (CU):** It controls all the operations of the CPU, such as:
 - Decoding the instructions of programs stored in the internal memory.
 - Controlling the flow of information through the ALU, I/O, and internal memory.
- 2) **Arithmetic and Logic Unit (ALU):** performs all the following operations types:
 - Arithmetic: Addition, Subtraction, Multiplication, Division, Modulus, Power,... etc.
 - Logical: AND, OR, XOR, SHIFT, ROTATE,...etc.

Architecture of Modern Computer Machine (2)

➤ **Internal Memory:** It's composed of chips of integrated circuits which are capable of quickly storing and retrieving data. There are two types of internal memory:

- 1) Read Only Memory (ROM).
- 2) Random Access Memory (RAM).

1) **Read Only Memory (ROM):**

- It contains built-in programs (e.g., BIOS) that are needed by the computer to start operation (booting up) when powered on.
- These programs are stored permanently during the manufacturing and can't be lost when power is turned off (*nonvolatile memory*)
- The CPU can **only read from** the ROM.

Architecture of Modern Computer Machine (3)

2) Random Access Memory (RAM):

- It's the main memory of the computer.
- It stores the programs and their data which are needed to be executed by the CPU.
- The CPU can **read from and write to** RAM .
- When the power is off, all the stored data in RAM are lost (*volatile memory*).
- When you buy a computer, you pay for RAM not ROM.

➤ External Storage (Auxiliary Memory):

- It's a secondary memory (*nonvolatile storage*).
- It holds the programs and data permanently even if the power is off.
- It has many forms such as Compact Disks (CDs), Digital Versatile Disks (DVDs), and Flash Memories.

Architecture of Modern Computer Machine (4)

➤ **Input Devices:** are devices that allow a user to enter data to a computer, and convert it into electric digital signals to be suitable for processing. Examples of these devices such as **keyboard, mouse, joystick, light-pen, scanner, microphone and ...etc.**

➤ **Output Devices:** are devices that convert the electric digital signals resulted from the CPU into letters, numbers, symbols, images, videos, sounds and paintings. Examples of these devices are **screen (monitor), speaker, printer, plotter (computer printer for printing vector graphics) and ...etc.**

Storage Capacity Measurement

The units that are usually used for measuring storage capacities:

UNIT	ABBREVIATION	MEANING
bit	b	1 binary digit
Byte	B	8 bits
Kilobit	Kb	2^{10} b = 1024 bits
Kilobyte	KB	2^{10} B = 1024 bytes
Megabit	Mb	2^{20} b = 1024 Kb
Megabyte	MB	2^{20} B = 1024 KB
Gigabit	Gb	2^{30} b = 1024 Mb
Gigabyte	GB	2^{30} B = 1024 MB
Terabyte	TB	2^{40} B = 1024 GB

Internal Memory

■ There are 7-types of internal memory inside a digital computer:

1. Random Access Memory (RAM).
2. Read Only Memory (ROM).
3. CACHE Memory.
4. CPU Internal Registers.
5. Video Random Access Memory (VRAM).
6. Complementary Metal Semiconductor Memory (CMOS).
7. Virtual Memory.

RAM Organization

■ The RAM of digital computers consists of a large number of “**memory cells**” where each **memory cell** is called a “**word**”.

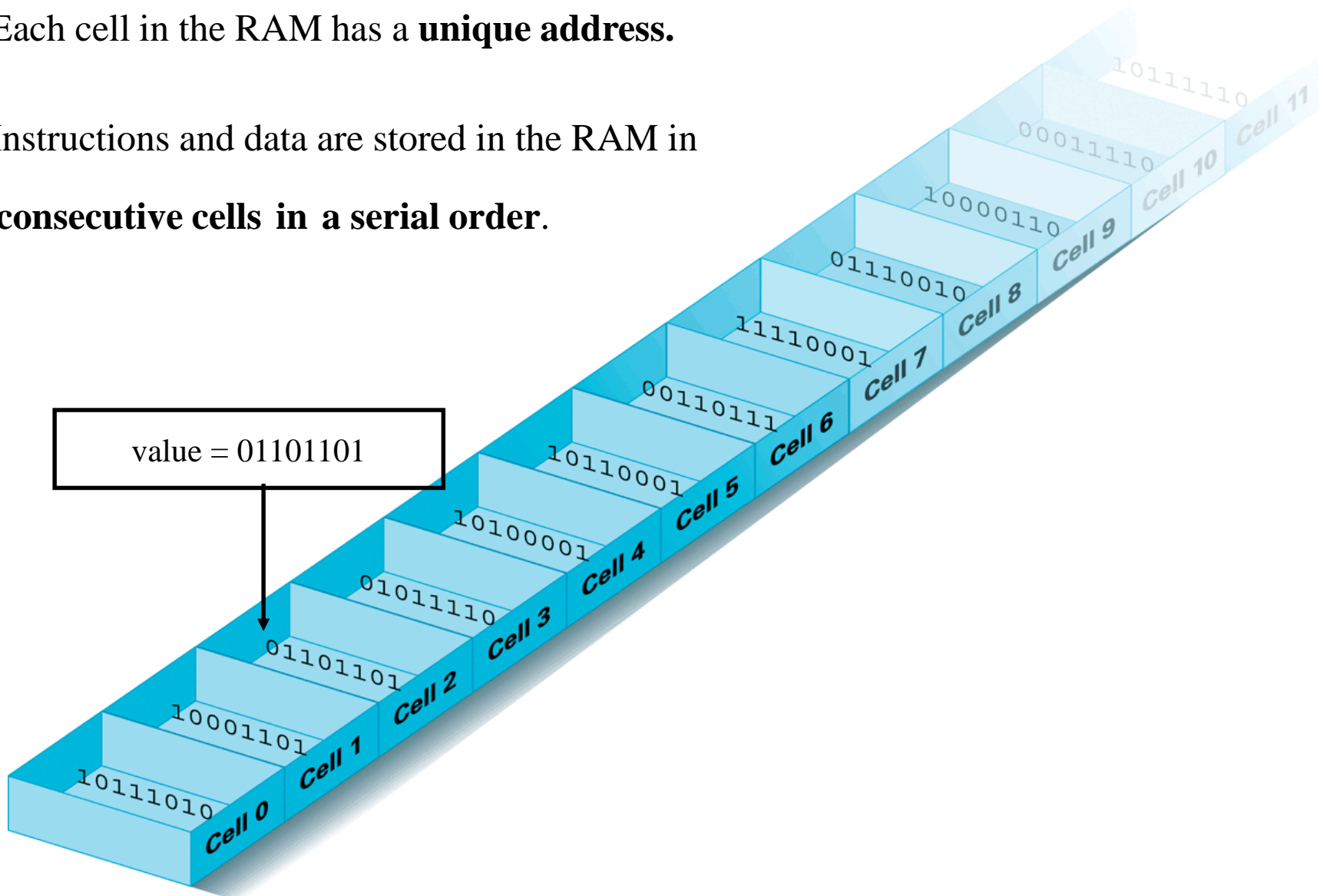
■ With this memory organization, **each memory cell is addressed, accessible and modified individually.**

■ A “**word**” or a cell in memory is an entity of bits (group of 1’s and 0’s) that can be moved in and out of the memory as a unit.

■ The size of a memory word is multiples of 8-bits in length: **8-bits (i.e.,1 byte), 16-bits (i.e., 2 bytes) , or 32-bits (i.e., 4 bytes),** and so on .

RAM Organization (2)

- Each cell in the RAM has a **unique address**.
- Instructions and data are stored in the RAM in **consecutive cells in a serial order**.



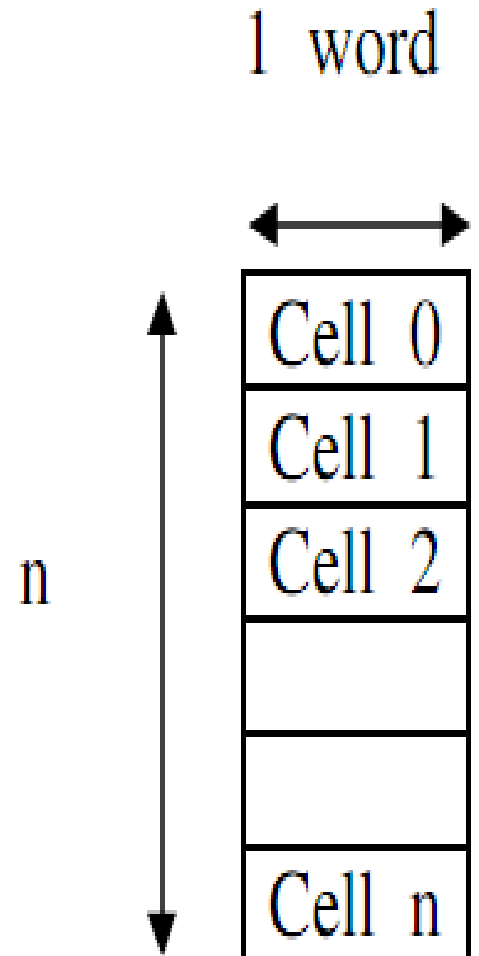
RAM Organization (3)

The capacity of the RAM is simply the total number of bytes that it can store.

The capacity of memory = n words

$$= n \times m \text{ bytes}$$

$$= n \times m \times 8 \text{ bits}$$



RAM Organization (4)

Example(1): The following memory capacities are measured in words, obtain the equivalent capacities in terms of bytes and bits.

- i- 1K words with 16 bits each
- ii- 16 M words with 32 bits each
- iii- 2048 words with 8 bits each

Solution:

- i- Memory capacity = $1\text{ K} \times 2\text{ B} = 2\text{ K B} = 2 \times 1024\text{ B} = 2048\text{ B}$
 $= 1\text{ K} \times 16\text{ b} = 16\text{ K b} = 16 \times 1024\text{ b} = 16384\text{ b}$
- ii- Memory capacity = $16\text{ M} \times 4\text{ B} = 64\text{ MB} = 64 \times 1024 \times 1024\text{ B}$
 $= 67108864\text{ B} = 67108864 \times 8\text{ b} = 536870912\text{ b}$
- iii- Memory capacity = $2 \times 1024 \times 1\text{ B} = 2048\text{ B} = 2\text{KB}$
 $= 2\text{ K} \times 8\text{ b} = 16\text{ K b} = 16384\text{ b}$

RAM Addressing

■ The CPU can access each memory cell (word) by a unique address.

■ **Example 1:** Go to memory cell 1000 (Absolute Address)

Go to memory cell next to memory cell 1000 by 6 cells (Relative Address)

■ A CPU with n-address line (**the width of the address bus of a CPU is n bits**) can access memory cells up to 2^n cells. In this case, the address range from 0 to $2^n - 1$, so the first cell address is **0....000000000** and the last cell address is **1....11111111**.

■ The CPU can access all memory cells equivalently without any differences in accessibility between “low address” and “high address” cells. Therefore, main memory is usually termed **RAM (i.e., Random Access Memory)**, which means that all memory cells are accessible just by address.

RAM Addressing (2)

Example 2: What is the total memory capacity that can be addressed using a CPU with 16-address line?

Solution: The total memory capacity = $2^{16} = 2^6 \times 2^{10} = 64 \text{ Kilo word (KW)}$. If each word = 1 byte, then the total memory capacity = 64 KB. The address of the first memory cell = 0000 0000 0000 0000. The address of the last memory cell = 1111 1111 1111 1111.

In other words , using hexadecimal notation, the address range is: 0000 to FFFF.

Example 3: what is the total number of address lines needed by a CPU to address memory of capacity of 64 MB?

Solution: The total memory capacity = $2^6 \times 2^{20} = 2^{26} \text{ B}$. If each word (cell) = 1 byte, then the total memory capacity = $2^{26} \text{ word (cell)}$, thus the number of address lines is 26.

Read Only Memory (ROM)

- Contains small set of **never-changed** instructions called “**Basic Input/Output System**” (**BIOS**) which tell the computer how to access the hard disk, find the operating system and load it into RAM.

- Use also random addressing as RAM but **only for reads**.

- **Types of ROM:**

- A. Programmable ROM (PROM):** written and programmed once.

- B. Erasable Programmable-ROM (EPROM):** erased and reprogrammed many times.

- C. Flash Memory:** a special type of EPROM used as storage device in digital cameras, cell phones and home video game consoles.

Read Only Memory (ROM) (2)

ROM vs RAM



X



CACHE Memory

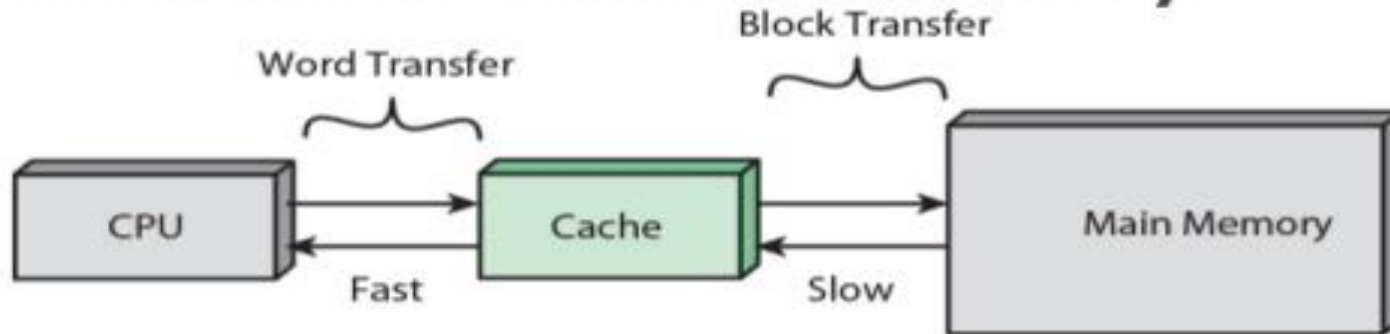
- **Very high-speed** memory acts as **temporary area (volatile memory)** between the CPU and RAM.
- This memory is typically **integrated directly with the CPU chip** or placed on a **separate chip** that has a separate bus interconnect with the CPU.
- The basic purpose of cache memory is **to store program instructions and data that are frequently or routinely re-referenced by the program during operation**. So fast access to these instructions and data increases the overall speed of the program execution.
- There are **three levels of CACHE memory** inside a digital computer: **L1, L2 and L3**.

CACHE Memory (2)

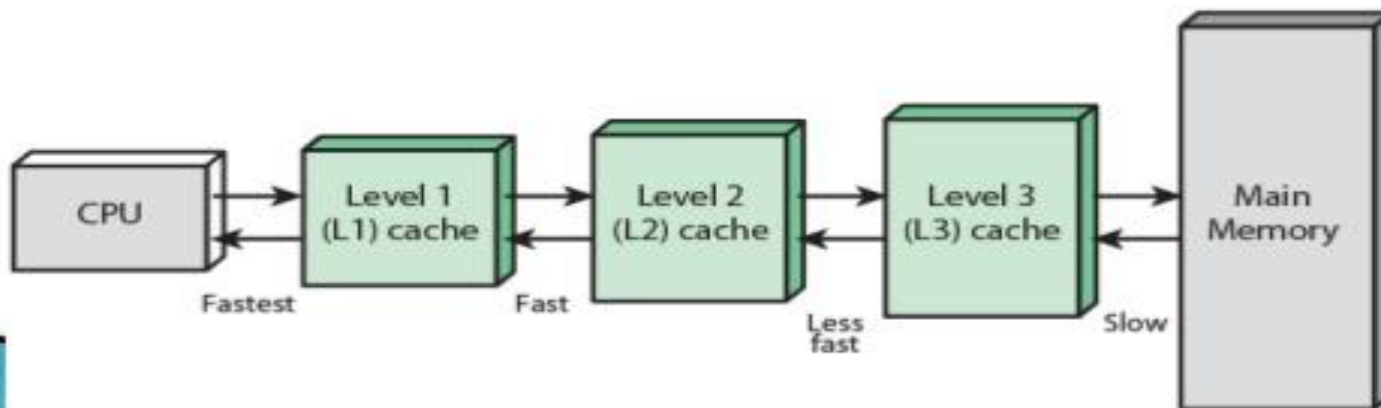
- **L1 cache** is extremely fast but relatively small, and is usually embedded in the processor chip (CPU).
- **L2 cache** is often **larger than L1**; it may be located on the CPU or on a separate chip with a high-speed alternative system bus interconnecting the cache to the CPU, so as not to be slowed by traffic on the main system bus.
- **L3 cache** is typically specialized memory that **works to improve the performance of L1 and L2**. It can be significantly **slower than L1 or L2**, but is usually **larger than them and double the speed of RAM**. In the case of multi-core processors, each core may have its own dedicated L1 and L2 cache, but share a common L3 cache.

CACHE Memory (3)

Cache and Main Memory



(a) Single cache



(b) Three-level cache organization

CACHE Memory (4)



CPU request a byte
from address X



L1 cache reads 64
bytes into cache



L2 cache reads 256
bytes into cache



L3 cache reads 1024
bytes into cache

CPU Internal Registers

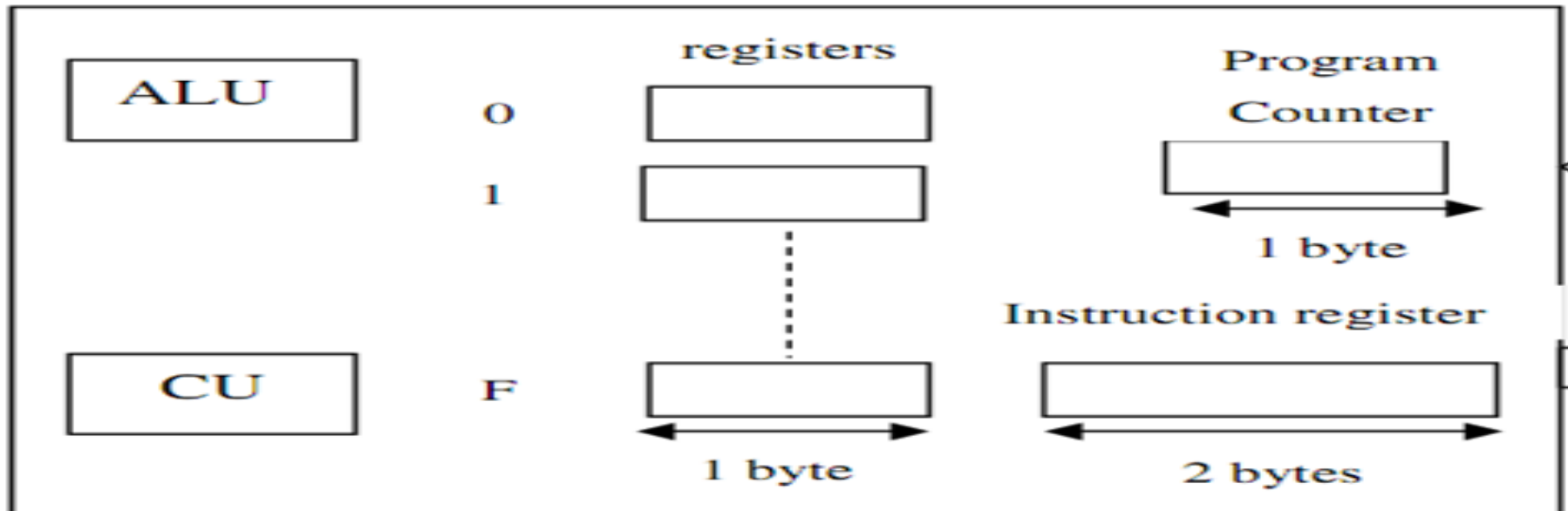
- The microprocessor or processor is an integrated circuit that contains the computer's central processing unit (CPU) which executes programs that are **stored only** in the internal main memory (RAM) of the computer.
- The CPU contains internal temporary storage cells, called **registers**, that are similar to main memory cells, to hold data that is being processed.
- The size and number of registers are critical in determining the speed and power of a computer.

CPU Internal Registers (2)

There are **two types of registers** (volatile memory) inside the CPU:

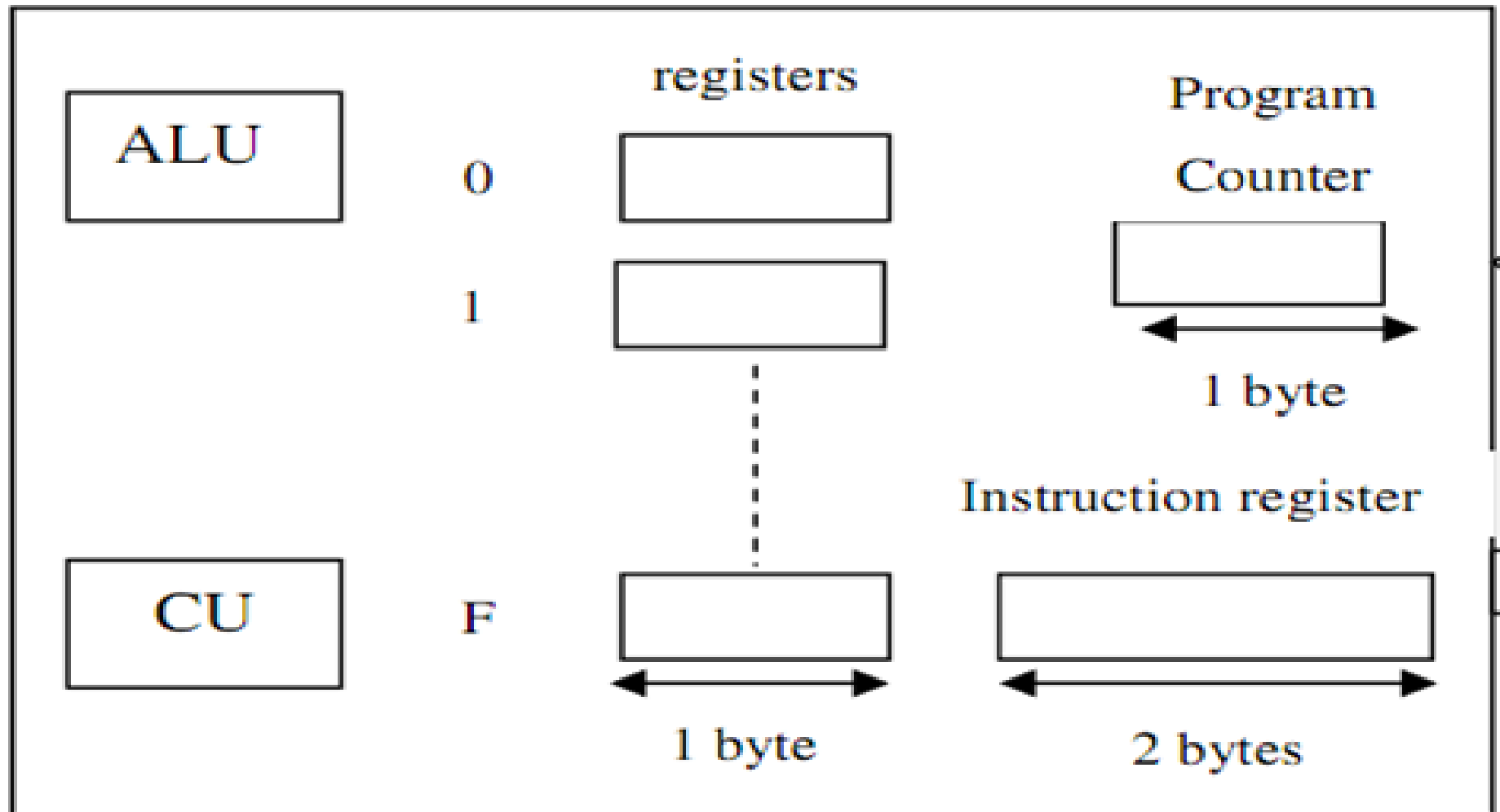
- 1- **General-purpose registers:** which serves as temporary holding place of the processed data by the ALU.
- 2- **Special-purpose registers:** which are responsible for proper execution of the programs, e.g. “Program Counter Register (PCR)” and “Instruction Register (IR)”.

CPU



Microprocessor (1)

CPU



Microprocessor (2)

■ **A microprocessor has two main components:**

1. **Control Unit (CU)** : is responsible for coordinating the computer's activities.
2. **Arithmetic & Logic Unit (ALU)**: is responsible for performing arithmetic (Addition, Subtraction, Multiplications, Division, Power, Modulus,...) and logical (OR, AND, NOT, XOR, SHIFT, ROTATE...) operations.

■ The microprocessor has a finite set of instructions (**instruction set**) that can be performed, **such instructions are hard-wired into the processor's circuitry.**

■ **The instruction set include:**

1. **Basic arithmetic and logic instructions**
2. **Data transfer instructions.**
3. **Control instructions.**

Types Of Microprocessor

■ **Microprocessor can be classified according to the following criteria:**

1. **The technology of instruction set:**

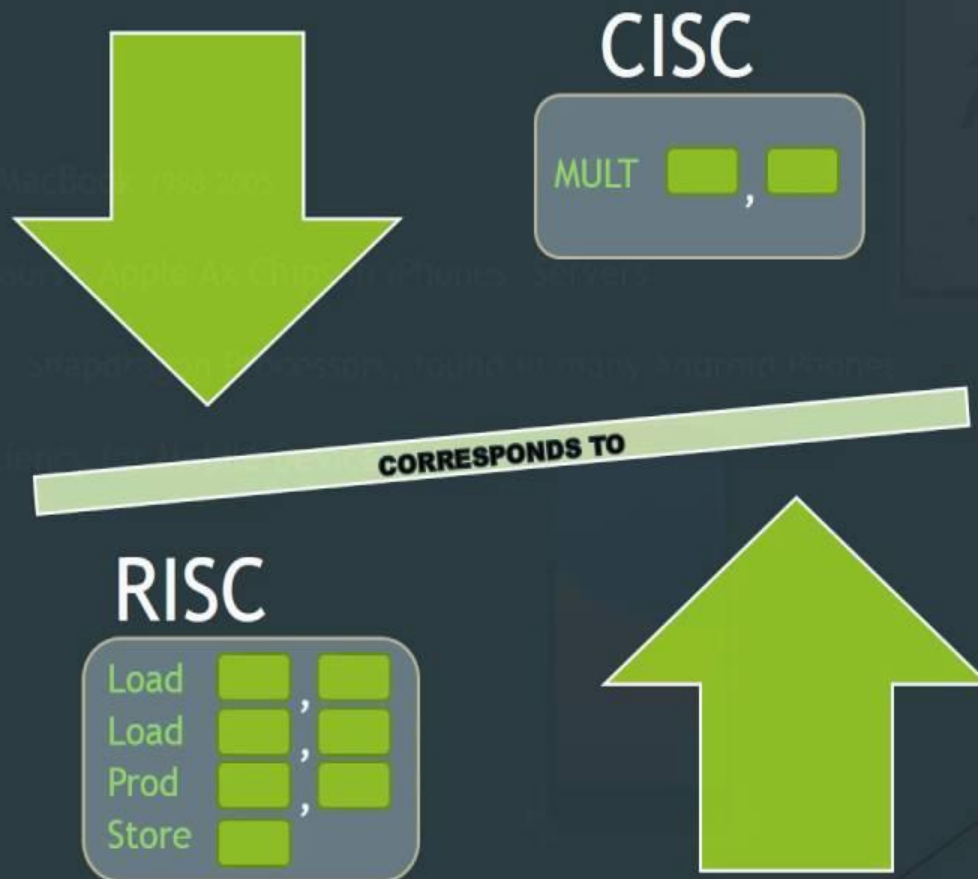
A. Complex instruction set computer (CISC): The set includes large number of complex instructions each requires several clock cycles for execution.

B. Reduced instruction set computer (RISC): The set includes limited number of simple instructions each requires fewer clock cycles for execution.

■ Although a RISC processor executes faster than a CISC one but it might needs more instructions to complete the task than a CISC processor needs for the same task.

Types Of Microprocessor (2)

Comparison of Instruction Sets



Types Of Microprocessor (3)

CISC(complex instructions)

```
10101011001010100000100  
000100011010011101011010
```

```
010000111100101000010111  
110100101001001110110010  
100010101001110101010100  
10100101010001010101001
```

```
11111110001010101100
```

RISC(simple instructions)

```
101010101010101101010  
1010101111111101110101
```

```
101010110010101000001  
0011111111111111110101
```

```
101010110010101000001  
0011111111111111110101
```

```
101010000000001000001  
0010100101011001110101
```

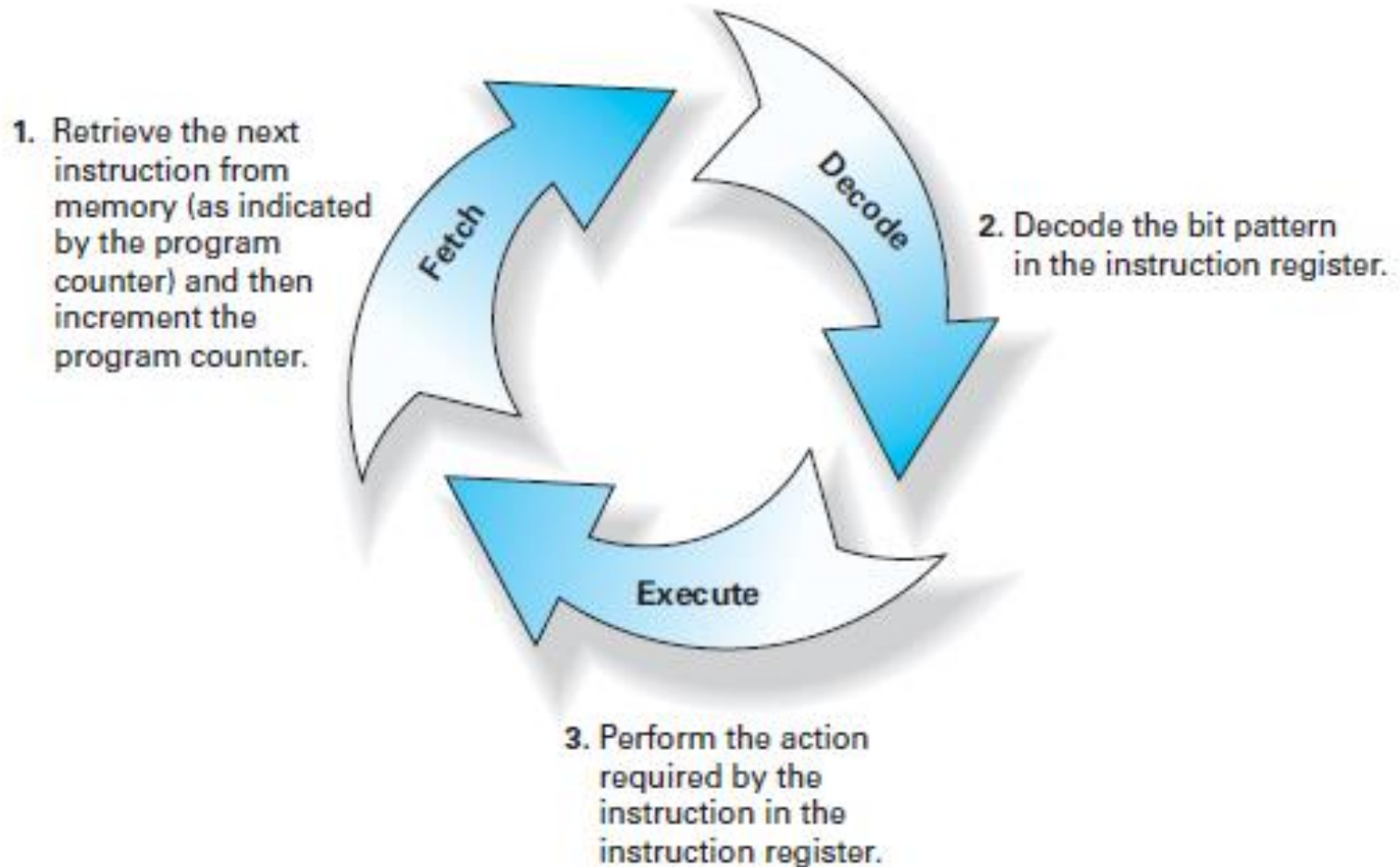
Types Of Microprocessor (4)

■ **Microprocessor can be classified according to the following criteria:**

2. **The technology of executing instructions:**
 - A. **Sequential:** The processor must complete all steps in the instruction cycle before it begins to execute the next instruction.
 - B. **Pipelining:** The processor can begin executing an instruction before it completes the previous one.
 - C. **Parallel:** The processor can execute multiple instructions at the same time. Such a processing ability can be achieved using either of the following technologies:
 - I. **Hyper-Threading:** Which is a circuitry that enables the processor to simulate two processors per core.
 - II. **Multi-Core:** Which allows a single chip containing the circuitry of multiple processors (2,4,8 or 16) with a shared cache memory.

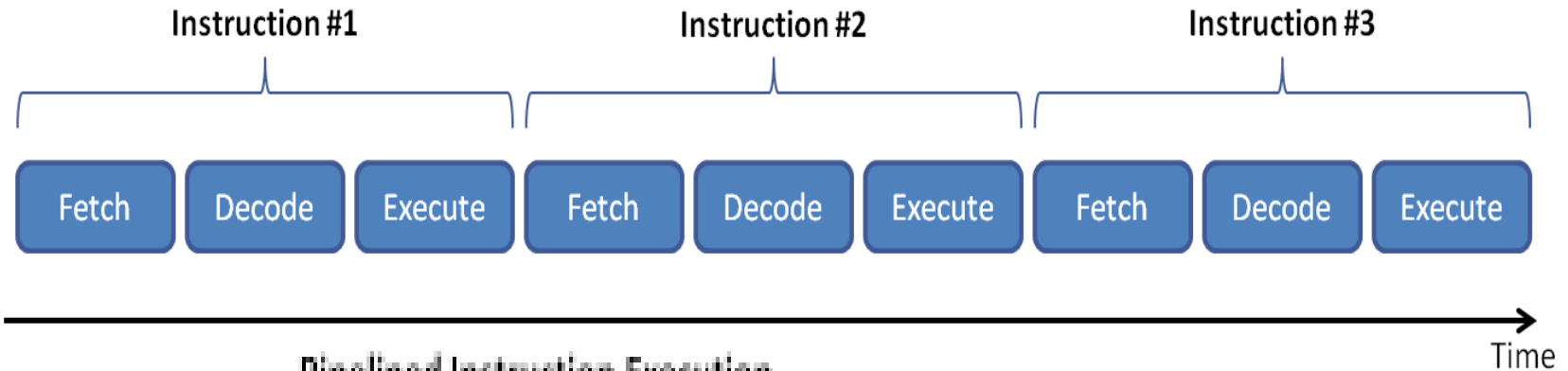
Instruction Execution Cycle

Figure 2.8 The machine cycle

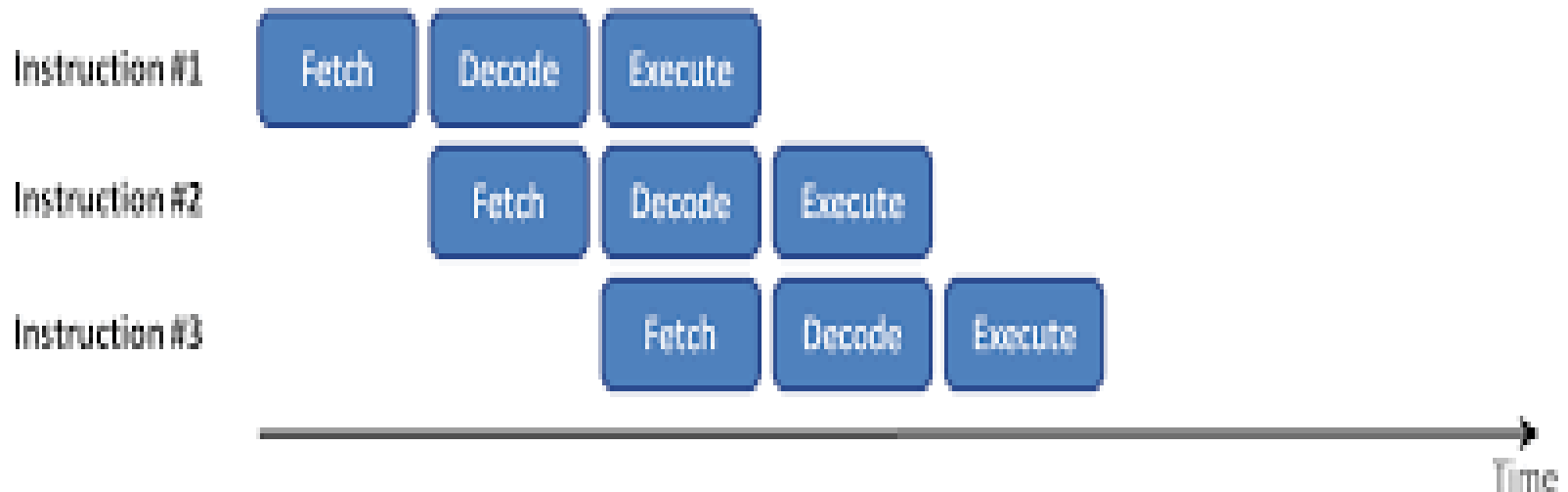


Types Of Microprocessor (5)

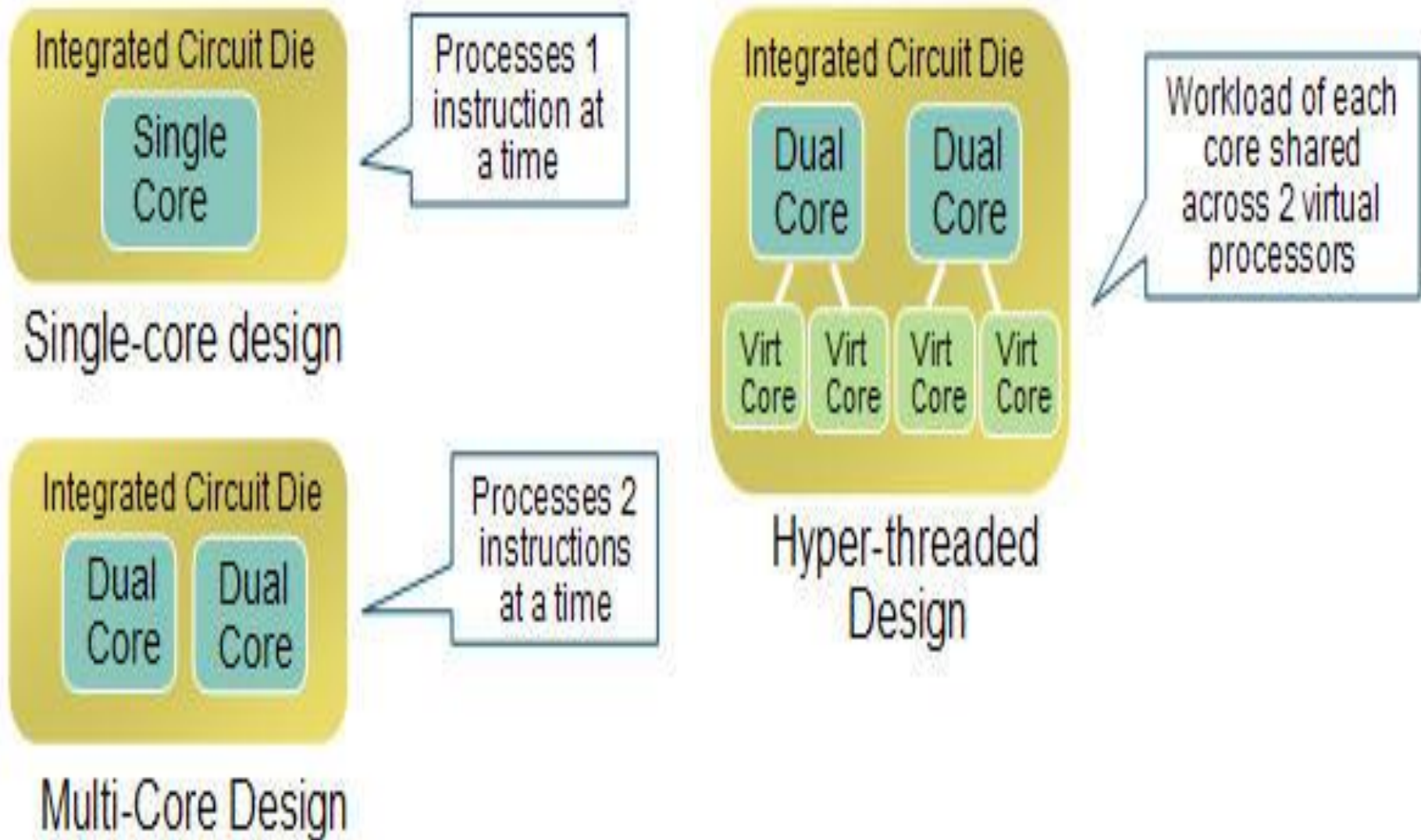
Sequential Instruction Execution



Pipelined Instruction Execution



Types Of Microprocessor (6)



Performance Of Microprocessor

■ **The performance of microprocessor is affected by the following factors:**

1. **Clock speed (or clock rate):** It is the speed at which a computer's microprocessor is able to process data. It is usually measured in hertz, generally in GHz, (i.e. billion cycles per second). For example, a CPU with a clock rate of 3 GHz can perform 3,000,000,000 billion clock cycles of (fetch-decode-execute) per second.
2. **Word Size:** It is the number of bits that a microprocessor can manipulate at one time. It is based on the size of the registers of the ALU as well as the width of the data bus. For example, 32-bit microprocessor can process 4-bytes at a time, while 8-bytes microprocessor can process 8 bytes at a time.
3. **Instruction Set:** Which is the technology used for designing the microprocessor instructions set. Instructions set is usually: CISC or RISC technology.

Performance Of Microprocessor (2)

■ **The performance of microprocessor is affected by the following factors:**

4. **The speed of the front side bus:** Front-side bus (FSB) refers to the circuitry that transports data between the microprocessor and RAM. A fast FSB allows the processor to work at full capacity.
5. **Cache Size:** which refers to level-1 cache (L1) (inside the microprocessor chip), level-2 cache (L2) and level-3 cache (L3) (on a separate chip connected to the microprocessor through a special bus that is called as cache bus, back-side bus or BSB).
6. **Processing Technique:** which is the technology used for executing instructions. It is usually serial, pipelining, or parallel processing.

Data Manipulation

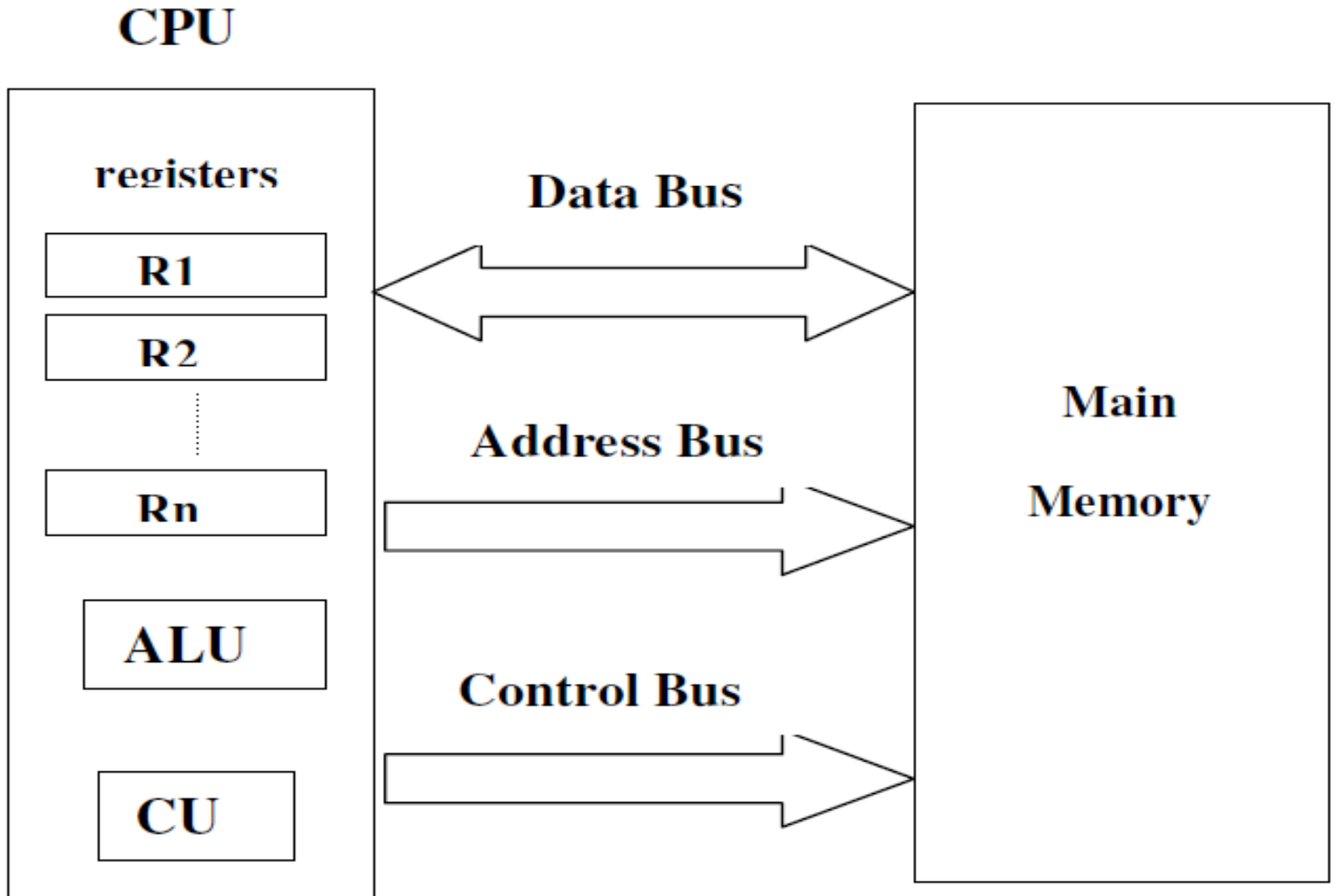
Data manipulation in an algorithmic machine means:

- The ability to perform operations on the data stored on memory.
- Coordinating the sequence of these operations.

To perform an operation on data stored in main memory:

1. CU transfers the data from the main memory into the general-purpose register.
2. CU informs the ALU which general-purpose register hold the data.
3. CU activates the appropriate circuitry within the ALU based the content of the instruction register.
4. CU tells the ALU which general-purpose register should receive the result.

Data Manipulation (2)



Data Manipulation (3)

■ To write (or store) data in a memory cell:

1. CPU sends the address of the memory cell through the address bus.
2. CPU sends the data to be stored through the data bus.
3. CPU sends a write control signal through the control bus.

■ To read data from a memory cell:

1. CPU sends the address of the memory cell through the address bus.
2. CPU sends a read control signal through the control bus.
3. CPU receives the contents of the required memory cell through the data bus and holds it in a general-purpose register.

Data Manipulation (4)

Machine Instructions

- The general format of a machine instruction consists of two parts:

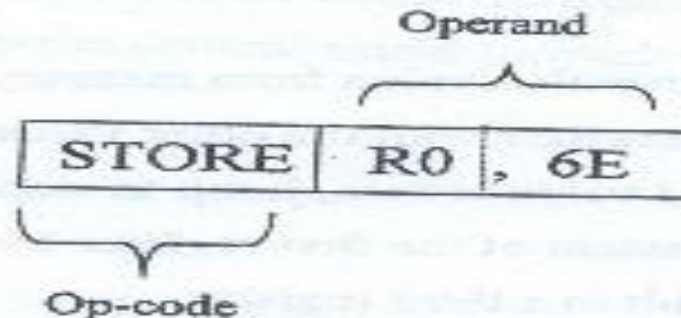
1- The op-code:

Which is the operation code of the instruction, such as: load, store, add.

2- The operand

Which provides more information about the operation specified by the op-code field.

Example (2): The format of the store instruction can be explained as follows:



Store the contents of register number 0 in memory cell of address 6E

Data Manipulation (5)

Machine Instructions

The instructions set of a processor can be grouped into 3 categories as:

“data transfer instructions”:

e.g. LOAD , STORE

LOAD R0, AB ➔ $R0 \leftarrow AB$

LOAD R1 , M[AB] ➔ $R1 \leftarrow M[AB]$

STORE R5, F5 ➔ $M[F5] \leftarrow R5$

▪ “arithmetic & logic instructions”:

ADD, AND, OR, XOR, SHIFT, ROTATE

▪ “control instructions”

JUMP (or BRANCH), HALT (or STOP)

Data Manipulation (6)

Machine Instructions

Example (1): The following steps explain how a machine language program can be written to divide two values stored in memory:

step 1: load a register with a value from memory

step 2: load another register with the other value from memory

step 3: IF this second value is zero, jump to step 6

step 4: Divide the content of the first register by the second register and leave the result in a third register.

step 5: store the contents of the third register in memory.

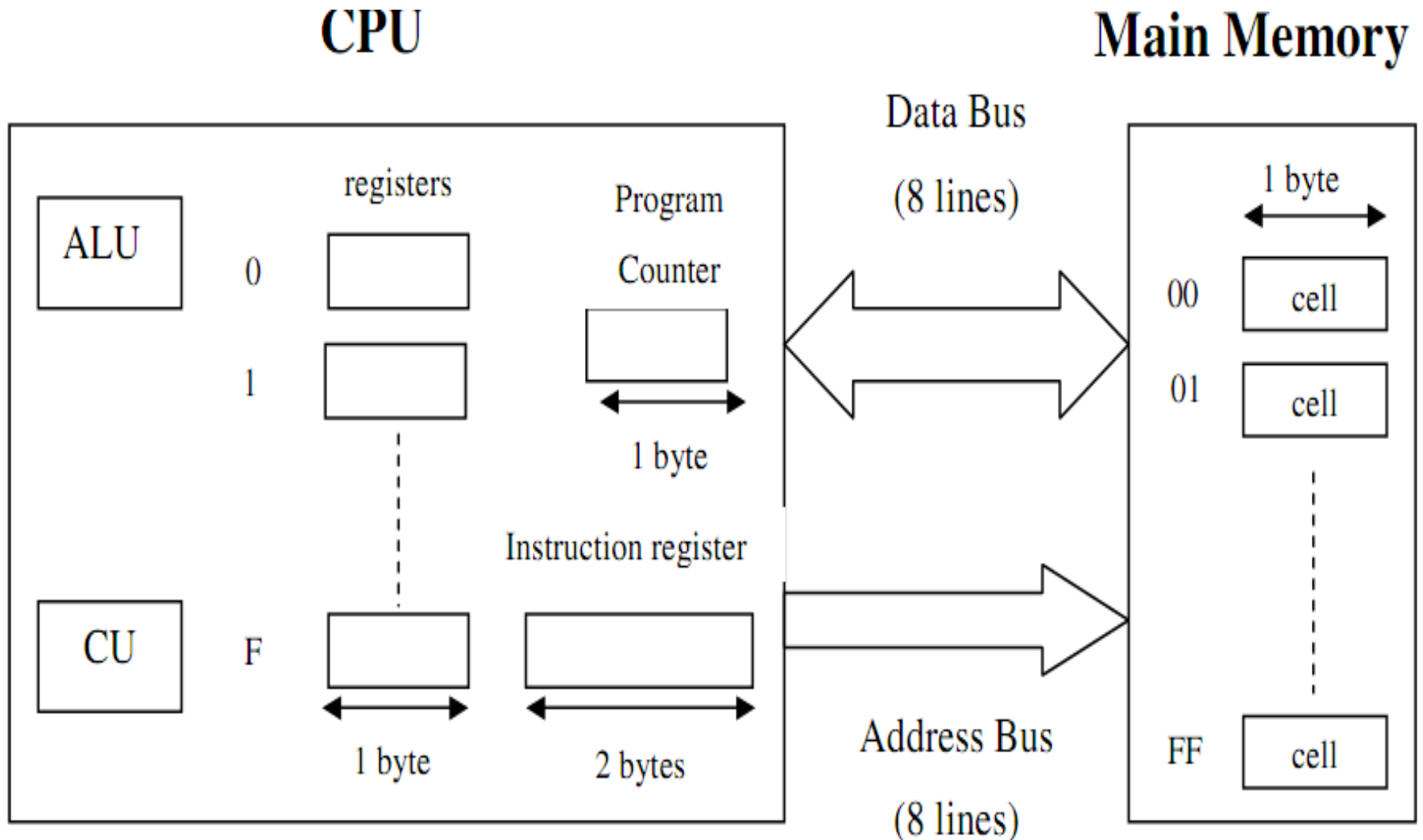
step 6: stop

It is clear that:

steps: 1, 2, 5	➔	data transfer instructions
steps: 3, 6	➔	control instructions
step 4	➔	arithmetic instruction

Data Manipulation (7)

Hypothetical Algorithmic Machine



Data Manipulation (8)

The Instructions Set of The Hypothetical Algorithmic Machine

Op-code	Operand	Description
1	RXY	<i>LOAD</i> the register R with the bit pattern found in the memory cell whose address is XY. <u>Example:</u> 14A3 would cause the contents of the memory cell located at address A3 to be placed in register 4.
2	RXY	<i>LOAD</i> the register R with the bit pattern XY. <u>Example:</u> 20A3 would cause the value A3 to be placed in register 0.
3	RXY	<i>STORE</i> the bit pattern found in register R in the memory cell whose address is XY. <u>Example:</u> 35B1 would cause the contents of register 5 to be placed in the memory cell whose address is B1.
4	0RS	<i>MOVE</i> the bit pattern found in register R to register S. <u>Example:</u> 40A4 would cause the contents of register A to be copied into register 4.
5	RST	<i>ADD</i> the bit patterns in registers S and T as though they were two's complement representations and leave the result in register R. <u>Example:</u> 5726 would cause the binary values in registers 2 and 6 to be added and the sum placed in register 7.

Data Manipulation (9)

The Instructions Set of The Hypothetical Algorithmic Machine

6	RST	<i>ADD</i> the bit patterns in registers S and T as though they represented values in floating-point notation and leave the floating-point result in register R. <u>Example:</u> 634E would cause the values in registers 4 and E to be added as floating-point values and the result to be placed in register 3.
7	RST	<i>OR</i> the bit patterns in registers S and T and place the result in register R. <u>Example:</u> 7CB4 would cause the result of ORing the contents of registers B and 4 to be placed in register C.
8	RST	<i>AND</i> the bit patterns in registers S and T and place the result in register R. <u>Example:</u> 8045 would cause the result of ANDing the contents of registers 4 and 5 to be placed in register 0.
9	RST	<i>EXCLUSIVE OR</i> the bit patterns in registers S and T and place the result in register R. <u>Example:</u> 95F3 would cause the result of EXCLUSIVE ORing the contents of registers F and 3 to be placed in register 5.
A	R0X	<i>ROTATE</i> the bit pattern in register R one bit to the right X times. Each time place the bit that started at the low-order end at the high-order end. <u>Example:</u> A403 would cause the contents of register 4 to be rotated 3 bits to the right in a circular fashion.
B	RXY	<i>JUMP</i> to the instruction located in the memory cell at address XY if the bit pattern in register R is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution. <u>Example:</u> B43C would first compare the contents of register 4 with the contents of register 0. If the two were equal, the execution sequence would be altered so that the next instruction executed would be the one located at memory address 3C. Otherwise, program execution would continue in its normal sequence.
C	000	<i>HALT</i> execution. <u>Example:</u> C000 would cause program execution to stop.

Data Manipulation (10)

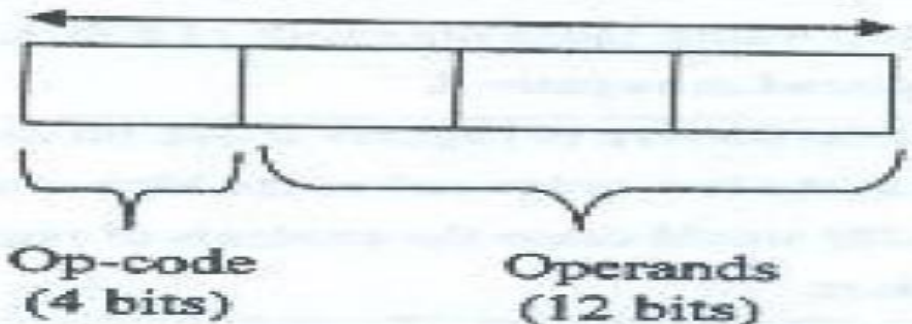
Program Execution

The fetch-decode-execute cycle continues until the “HALT” instruction is decoded in the **instruction register**, then before completing its execution the machine is informed that the program is completed and the machine cycle stops.

Since the instruction in the hypothetical CPU is 16 bits, and the size of each memory cell is 8 bits, then **each instruction of the program occupies two consecutive memory cells.**

The format of the machine instruction is as follows:

4 Hexadecimal Digits (2 bytes)



Binary Logic

- ★ We study binary logic as a foundation for analyzing and designing digital systems.
- ★ Binary logic consists of binary logical variables and a set of binary logical operations.
- ★ Binary logical variables take only one of two discrete values: **1** or **0**.
- ★ Binary logical variables are designated by letters of the alphabet, such as A, B, C, x, y, z, etc.

Logical Operations

- ★ There are three basic binary logical operations: **AND**, **OR** and **NOT**.
- ★ Each operation produces a binary result of **1** or **0**.
- ★ **AND** is denoted by a dot (\cdot).
- ★ **OR** is denoted by a plus ($+$).
- ★ **NOT** is denoted by an over bar ($\bar{}$) the variable.

Notation Examples

★ Examples:

❖ $Y = A \cdot B$ is read “Y is equal to A AND B.”

❖ $Z = X + Y$ is read “z is equal to x OR y.”

❖ $X = \bar{A}$ is read “X is equal to NOT A.”

★ Note: The statement:

$1 + 1 = 2$ (is read “one plus one equals two”)

is not the same as

$1 + 1 = 1$ (is read “1 or 1 equals 1”).

Operator Definitions

★ Operations are defined on the values "0" and "1" for each operator:

AND

$$0 \cdot 0 = 0$$

$$0 \cdot 1 = 0$$

$$1 \cdot 0 = 0$$

$$1 \cdot 1 = 1$$

OR

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 1$$

NOT

$$\bar{0} = 1$$

$$\bar{1} = 0$$

Truth Tables

- ★ Tabular listing of the values of a function for all possible combinations of values on its arguments
- ★ Example: Truth tables for the basic logic operations:

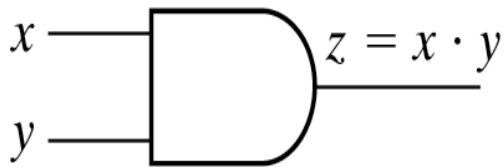
AND		
X	Y	$Z = X \cdot Y$
0	0	0
0	1	0
1	0	0
1	1	1

OR		
X	Y	$Z = X + Y$
0	0	0
0	1	1
1	0	1
1	1	1

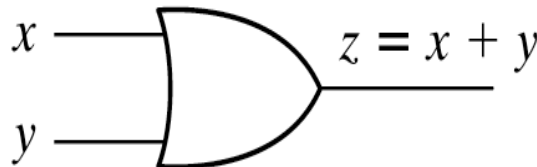
NOT	
X	$Z = \overline{X}$
0	1
1	0

Logic Gates

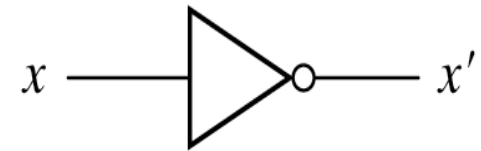
- ★ The logical operations are represented by logic gates.
- ★ The logic gate is an electronic circuit that operates on one or more input signals to produce an output signal.
- ★ Logic gates have special symbols:



(a) Two-input AND gate



(b) Two-input OR gate



(c) NOT gate or inverter

Fig. 1-4 Symbols for digital logic circuits

Exclusive OR / Exclusive NOR

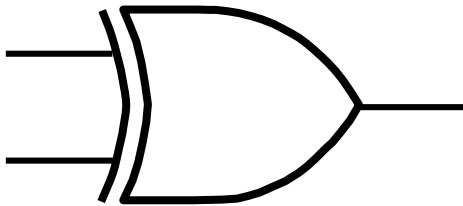
- The **eXclusive-OR (XOR)** function is an important Boolean function used extensively in logic circuits
- The XOR function may be:
 - Implemented directly as an electronic circuit (true gate)
 - Implemented by interconnecting other gate types (XOR is used as a convenient representation)
- The **eXclusive-NOR (XNOR)** function is the complement of the XOR function
- XOR and XNOR gates are complex gates

XOR / XNOR Tables and Symbols

XOR

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

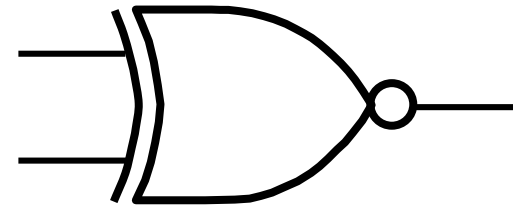
XOR Symbol



XNOR

X	Y	$\overline{X \oplus Y}$
0	0	1
0	1	0
1	0	0
1	1	1

XNOR Symbol



- The XNOR is also denoted as **equivalence**

Data Manipulation (11)

Logical Operations

- The logic operations that can be performed by the ALU are: AND, OR and XOR.
- Each of such operations has its specific instruction of the CPU instruction set.
- Each of such operations can be illustrated as follows:

Data Manipulation (12)

AND:

$$\begin{array}{r} \text{AND} \quad 0 \\ \quad 0 \\ \hline 0 \end{array}$$

$$\begin{array}{r} \text{AND} \quad 0 \\ \quad 1 \\ \hline 0 \end{array}$$

$$\begin{array}{r} \text{AND} \quad 1 \\ \quad 0 \\ \hline 0 \end{array}$$

$$\begin{array}{r} \text{AND} \quad 1 \\ \quad 1 \\ \hline 1 \end{array}$$

OR:

$$\begin{array}{r} \text{OR} \quad 0 \\ \quad 0 \\ \hline 0 \end{array}$$

$$\begin{array}{r} \text{OR} \quad 0 \\ \quad 1 \\ \hline 1 \end{array}$$

$$\begin{array}{r} \text{OR} \quad 1 \\ \quad 0 \\ \hline 1 \end{array}$$

$$\begin{array}{r} \text{OR} \quad 1 \\ \quad 1 \\ \hline 1 \end{array}$$

XOR:

$$\begin{array}{r} \text{XOR} \quad 0 \\ \quad 0 \\ \hline 0 \end{array}$$

$$\begin{array}{r} \text{XOR} \quad 0 \\ \quad 1 \\ \hline 1 \end{array}$$

$$\begin{array}{r} \text{XOR} \quad 1 \\ \quad 0 \\ \hline 1 \end{array}$$

$$\begin{array}{r} \text{XOR} \quad 1 \\ \quad 1 \\ \hline 0 \end{array}$$

Data Manipulation (13)

Example (1):

```
  0 1 0 0 1 0 1 1
AND
  1 0 1 0 1 0 1 1
  -----
  0 0 0 0 1 0 1 1
```

```
  1 0 0 0 0 0 1 1
AND
  1 1 1 0 1 1 0 0
  -----
  1 0 0 0 0 0 0 0
```

```
  1 1 1 1 1 1 1 1
AND
  0 0 1 0 1 1 0 1
  -----
  0 0 1 0 1 1 0 1
```

Example (2):

```
  0 1 0 0 1 0 1 1
OR
  1 0 1 0 1 0 1 1
  -----
  1 1 1 0 1 0 1 1
```

```
  1 0 0 0 0 0 1 1
OR
  1 1 1 0 1 1 0 0
  -----
  1 1 1 0 1 1 1 1
```

```
  1 1 1 1 1 1 1 1
OR
  0 0 1 0 1 1 0 1
  -----
  1 1 1 1 1 1 1 1
```

Data Manipulation (14)

Example (3):

01001011
XOR
10101011

11100000

10000011
XOR
11101100

01101111

11111111
XOR
00101101

11010010

Note:

The above logical operations are also called “logical masks”, or “logical filters”

Logical
Operation

Operand

Mask (or filter)

Result operand

Data Manipulation (15)

From uppercase to lowercase:

Use The OR mask 0 0 1 0 0 0 0 0

Or The XOR mask 0 0 1 0 0 0 0 0

From lowercase to uppercase:

Use The AND mask 0 1 0 1 1 1 1 1

Or The XOR mask 0 0 1 0 0 0 0 0

Therefore, the simplest solution for both ways is the XOR mask: 00100000

Data Manipulation (16)

Shift Operations

- Shifting the binary number one location to the right means dividing the number by 2.
- Shifting the binary number one location to the left means multiplying the number by 2.

Example (1)

Shift the following binary numbers one, two, and three locations to the left respectively and get their decimal equivalents:

a) 101 b) 100 c) 11

Data Manipulation (17)

Solution:

a)

$$(101)_2 = (5)_{10}$$

Shift left

$$(101)_2 \longrightarrow (1010)_2 = (10)_{10} = 5 \times 2$$

1 - location

Data Manipulation (18)

b)

$$(100)_2 = (4)_{10}$$

Shift left

$$(100)_2 \longrightarrow (10000)_2 = (16)_{10} = 4 \times 2 \times 2$$

2 - locations

Data Manipulation (19)

c)

$$(11)_2 = (3)_{10}$$

Shift left

$$(11)_2 \longrightarrow (11000)_2 = (24)_{10} = 3 \times 2 \times 2 \times 2$$

3 - locations

Data Manipulation (20)

Example (2)

Shift the following binary numbers one, two, and three locations to the right respectively and get their decimal equivalents:

- a) 1010 b) 10000 c) 11000

Data Manipulation (21)

Solution:

a)

$$(1010)_2 = (10)_{10}$$

Shift right

$$(1010)_2 \longrightarrow (101)_2 = (5)_{10} = 10 \div 2$$

1 - location

Data Manipulation (22)

b)

$$(10000)_2 = (16)_{10}$$

Shift right

$$(10000)_2 \longrightarrow (100)_2 = (4)_{10} = 16 \div (2 \times 2)$$

2 - locations

Data Manipulation (23)

c)

$$(11000)_2 = (24)_{10}$$

Shift right

$$(11000)_2 \longrightarrow (11)_2 = (3)_{10} = 24 \div (2 \times 2 \times 2)$$

3 - locations

Data Manipulation (24)

Rotate Operations

Assuming binary strings of 8 bits, right and left rotations can be performed as follows:

Example (5)

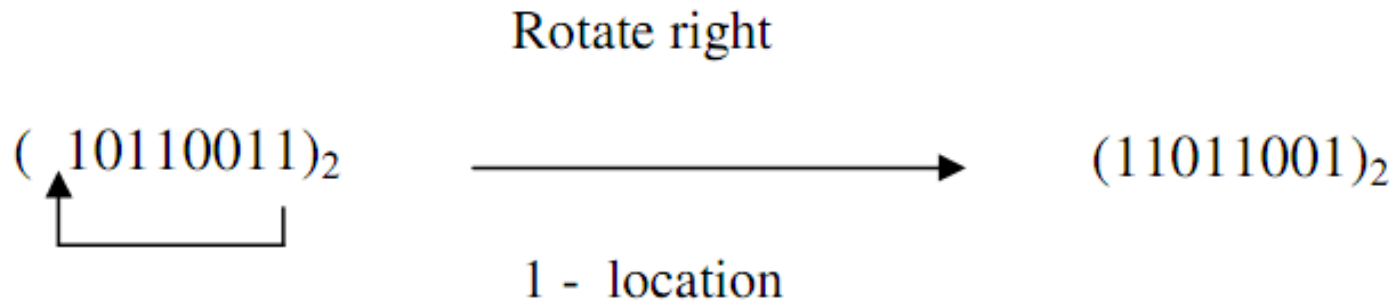
Perform a rotation operation one, two and three locations to the right for the following binary numbers:

- a) 10110011 b) 11110000 c) 00111100

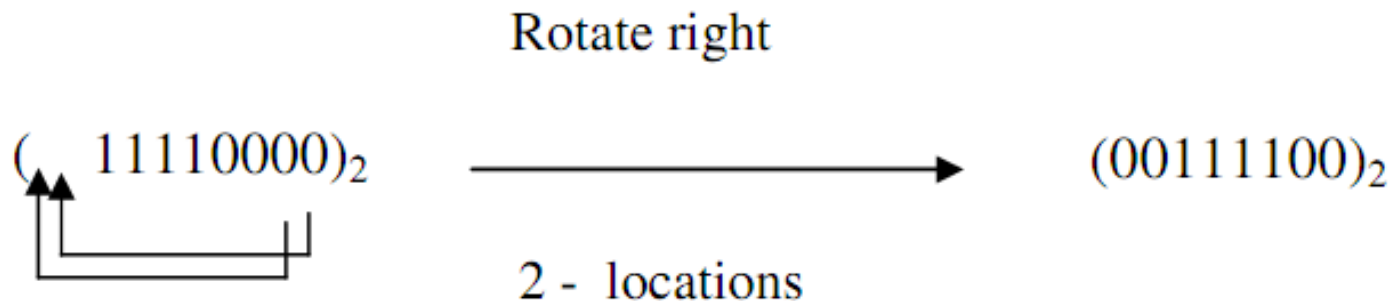
Data Manipulation (25)

Solution:

a)

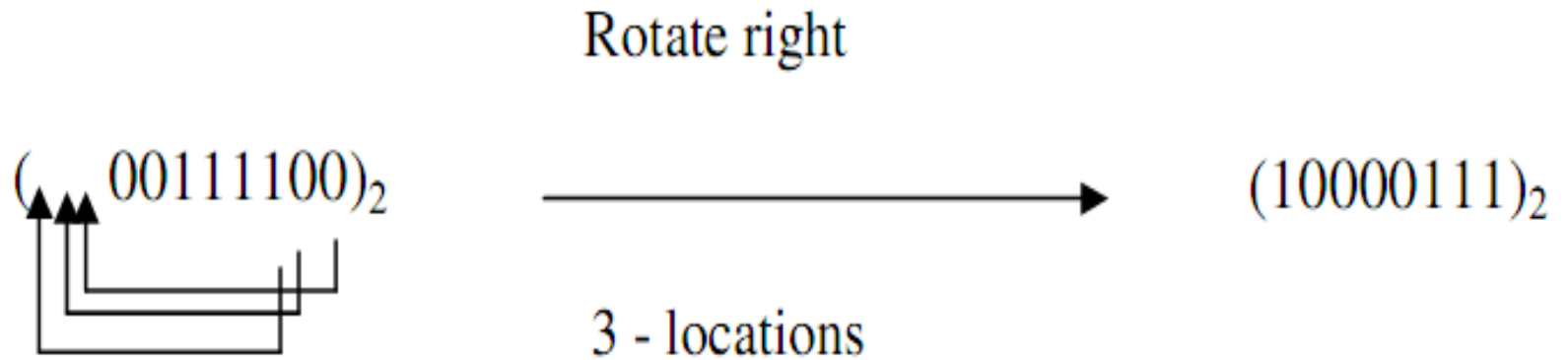


b)



Data Manipulation (26)

c)



Data Manipulation (27)

Note:

- Rotate right operation is also called “Circular right shift”
- Rotate left operation is also called “Circular left shift”

Example (6)

What is the result of performing one-bit left circular shift on the following hexadecimal numbers:

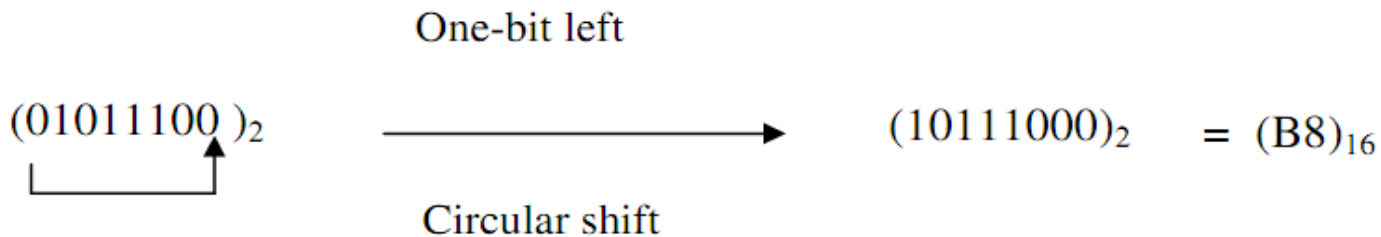
- a) 5C b) B7

Data Manipulation (28)

Solution:

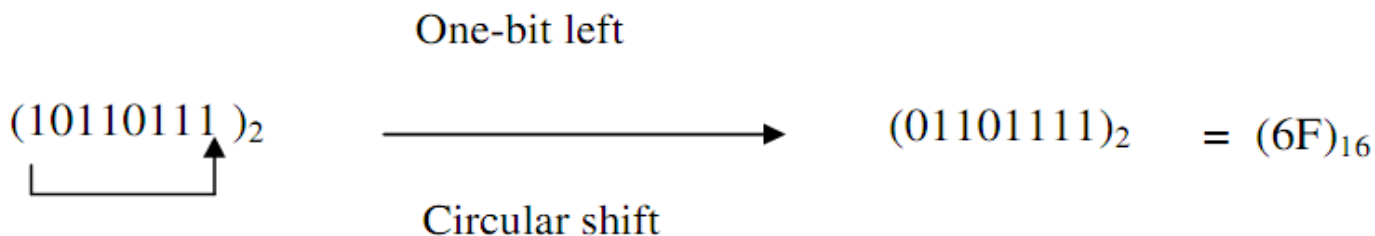
a)

$$(5C)_{16} = (01011100)_2$$



b)

$$(B7)_{16} = (10110111)_2$$



The End

Questions?

