An Introduction to Computer Architecture and Data Representation

Computer Architecture

Lectures 1-2

Mechatronics Engineering

Assistant Professor: Isam M. Asaad

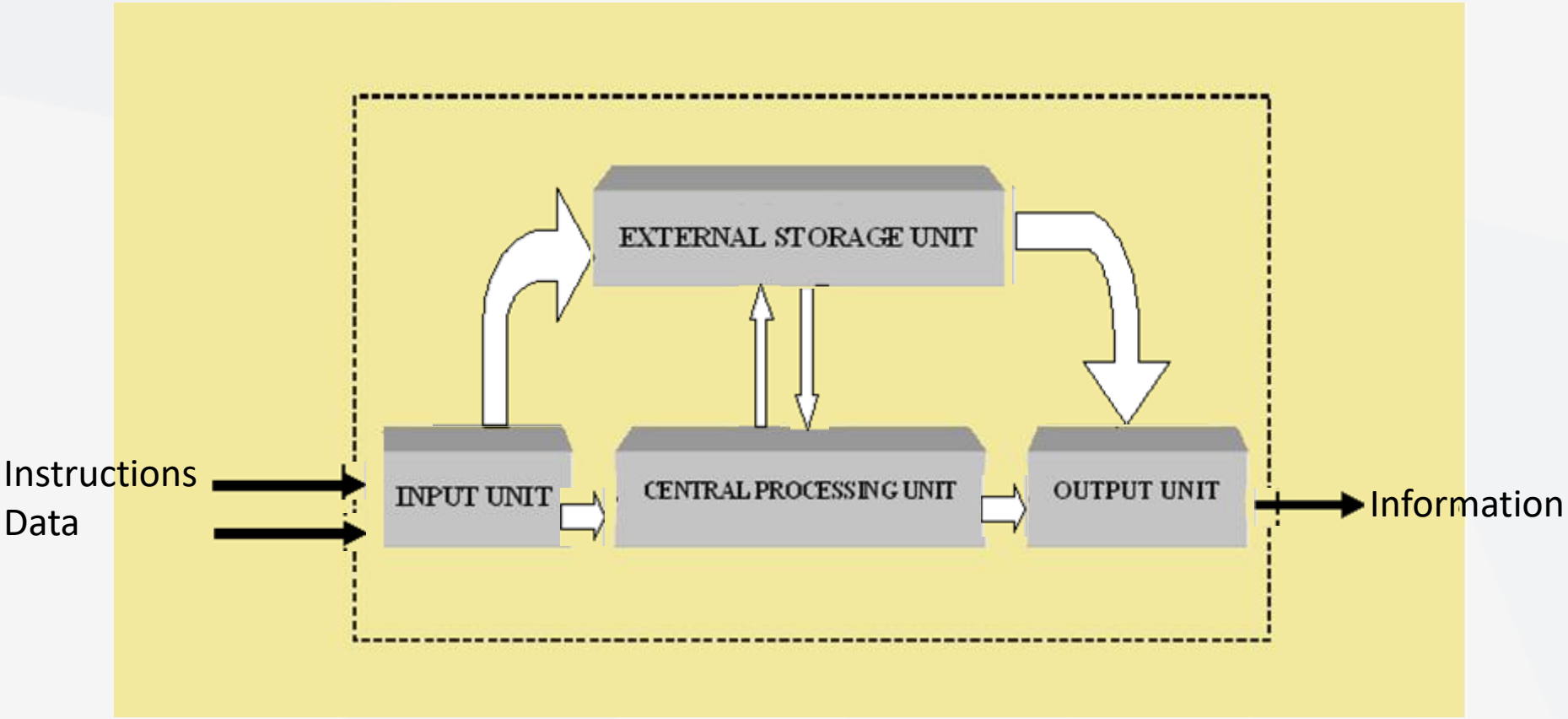
Lecture contents:

- What is a computer?
- Computer Generations:
- Characteristics of computer generations
- How data represented in a computer?
- Main Components of Computer System
- Computer Systems
- Programmer's View
- Architect's View
- Implementer's View
- Moore's Law
- Computer Architecture
- Memory Hierarchy
- Micro-processor Vs. Micro-controller
- Assembly vs. Machine Code
- Translating Languages
- Advantages of High-Level Languages
- Why assembly language?
- Instructions and Machine Language
- Instruction Fields
- Positional Number Systems
- Binary Numbers
- Signed Integers and Two's Complement
- Carry and Overflow
- Character Storage
- Printable ASCII Codes
- Control Characters
- Lecture References

What is a computer?

- An electronic device for storing and processing data, typically in binary form, according to instructions given to it in a variable program.
  - Changing the program changes the computer behavior! (solves a different problem!!!).
- A Program is simply a sequence of binary codes that represent instructions for the computer.
- The Program is stored in a Memory .
- External inputs to the Computer can also alter the behavior the computer.
- The computer will have Outputs that can be set/reset via program instructions.

How Computers Work ?



Computer System

Computer Generations:

The reality of the computer today did not come about by chance, but rather through developments in electrical and logic circuits and software. Here we present the most prominent stages of computer development, which began in the seventeenth century:

Ajax: The first attempt to automate arithmetic operations, a method still used to teach children counting, addition, and subtraction.

Napier's wooden machine: Using which the user can perform arithmetic operations (multiplication and long division).

Lenz's calculator: The user can perform multiplication, division, and extract square roots.

Babbage's difference machine: Using which the user can extract logarithms with great accuracy.

Printing adding machine: The user can perform addition and print the result on a strip of paper. The **Hollerith punch card reading** and **tabulation machine** is an electromechanical device invented by Herman Hollerith to facilitate the processing of large amounts of data, particularly for the 1890 U.S. Census. The machine relies on punch cards, representing information by the presence or absence of holes in specific locations.

Computer Generations:

1. **George Boole (1815–1864)**, known as the father of binary logic, his ideas on how to use symbols to process information forms, the foundation upon which all modern computers are based. In 1840, Boole announced the basic laws and rules of mathematics used to solve problems with logical formulas, known as Boolean algebra. This algebra served as the basis for the logical design of electronic computer circuits in the early 1940s, a hundred years after its announcement.
2. In **1937, Howard Aiken, John Presbert Eckart, and Aiken** built and put into operation the first electronic computer, called ENIAC. It could perform 500 additions or 300 multiplications per second (weighing 30 tons, 19,000 vacuum tubes, 1,500 square feet of space, 70,000 resistors, 10,000 capacitors, 6,000 switches, 500,000 solder points, and a power consumption of 200 kW).
3. In **1939, John Vest Atanasoff**, his assistant Clifford Berry, and Aiken developed prototypes of the electronic computer, and a machine consisting of 18,000 vacuum tubes was developed, performing arithmetic and logic operations by collecting electrical pulses generated at a rate of 100,000 pulses per second (e.g., EDVAC, EDSAC, and UNIVAC-1).
4. In **1955, Thomas Watson (son of IBM's founder)** moved his company into the computer age, launching the IBM-650.
5. **Between 1959 and 1960**, the second generation of computers appeared, replacing vacuum tubes with transistors.

Computer Generations:

In 1940, **von Neumann, Kolostin, and Lorks** formulated several **proposals that formed the foundation of computer design philosophy**:

1. The binary number system must be used in computer design.
2. The necessity of storing program instructions and data within the device.
3. The distinction between data and program instructions must not be made when representing them, as both are represented in binary numbers.

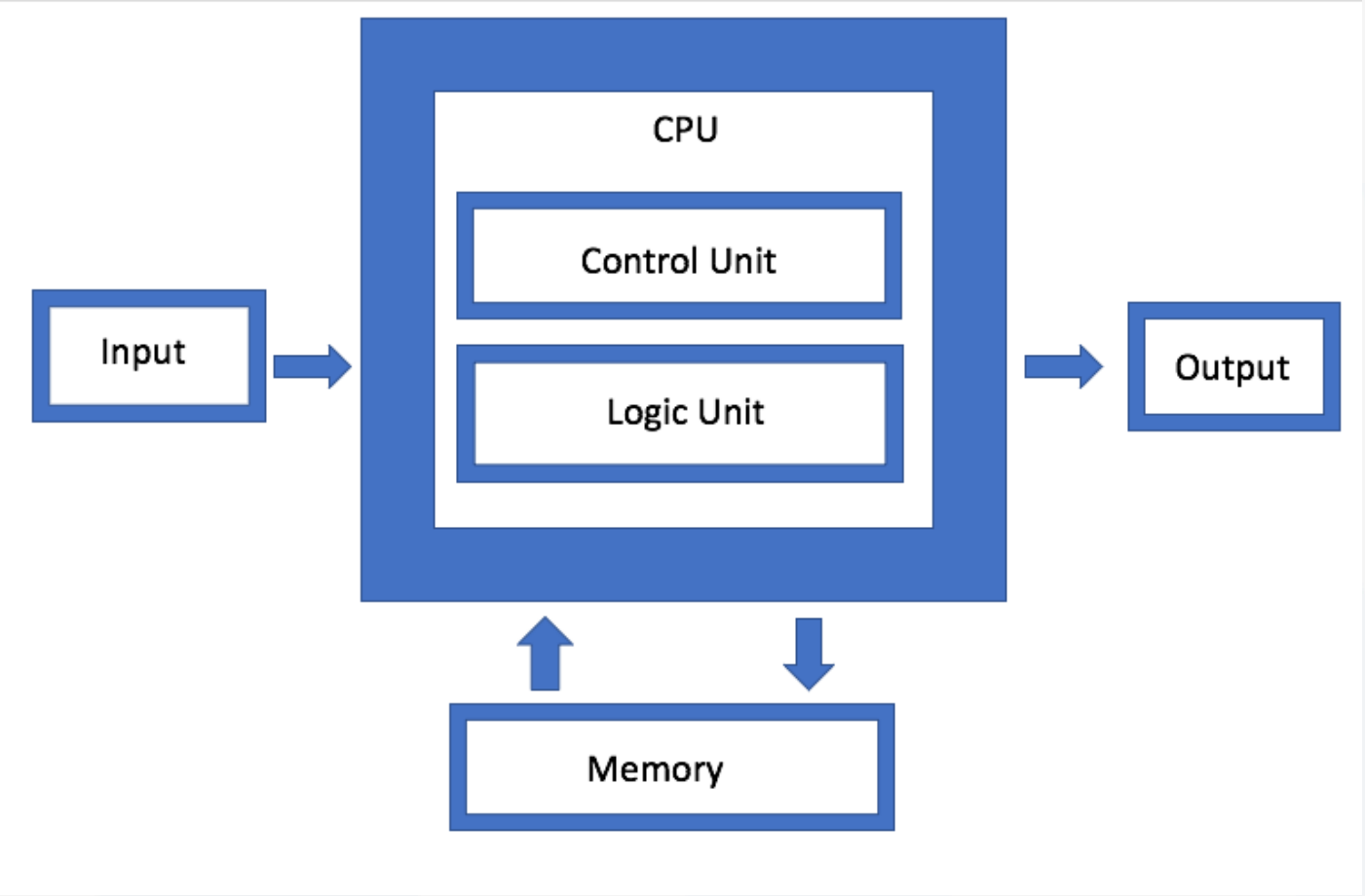
John von Neumann also **developed** the **internal architecture** of the **electronic computer system used to date**, which consists of:

1. The central processing unit (CPU).
2. The memory unit.
3. Storage devices.
4. Input/output peripherals.

The **von Neumann architecture** machine was based on several fundamental principles:

1. A centralized CPU, which controls every operation made in the computer system.
2. A centralized memory with fixed-length slots, whose cells occupied by multiple parts of computer system.
3. A limited number of input/output devices.

Computer Generations:

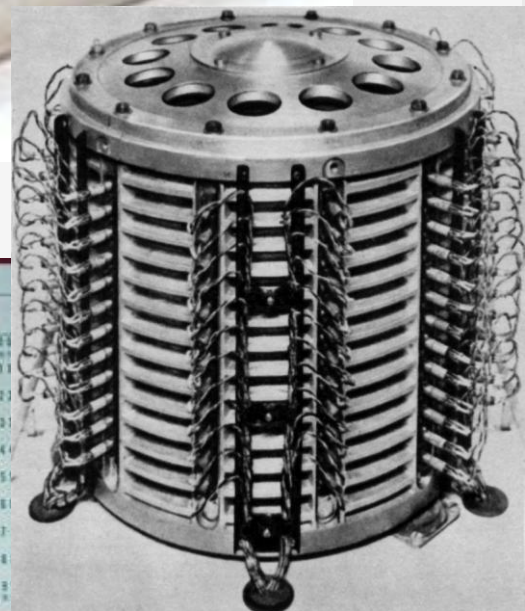


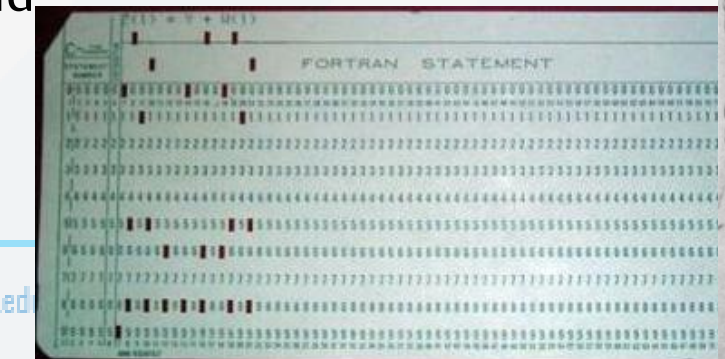
von Neumann architecture machine

Characteristics of computer generations

First Generation Computers (1951-1958):

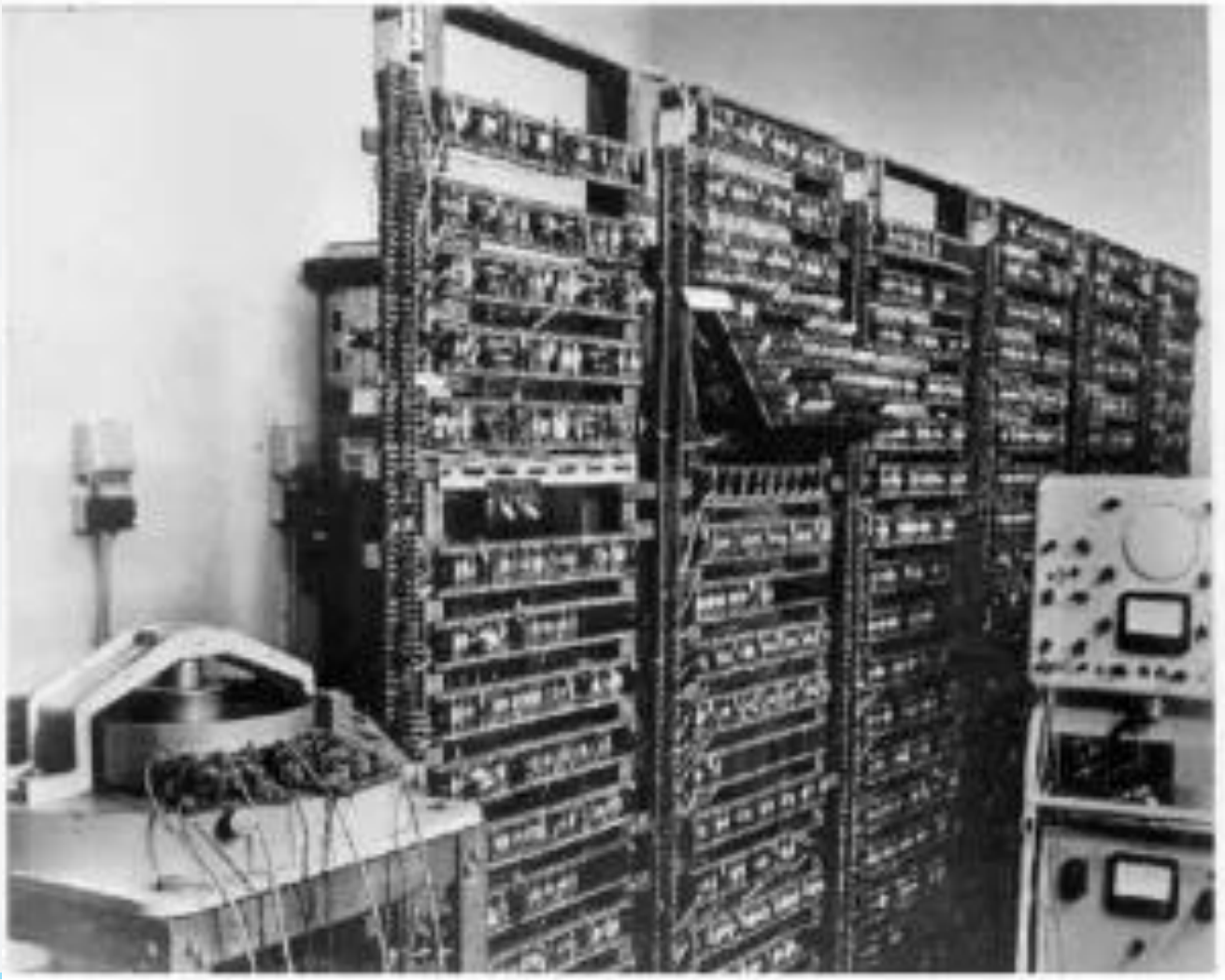
- Used vacuum tubes in electronic circuits.
- Used magnetic drums as the primary internal storage medium.
- Limited main storage capacity.
- Slow input and output (due to the use of punched cards).
- Programming in machine language and low-level (symbolic) languages.
- Faults were frequent, maintenance was difficult, and heat emission was problematic.
- Used for record-keeping and payroll processing applications.





Characteristics of computer generations

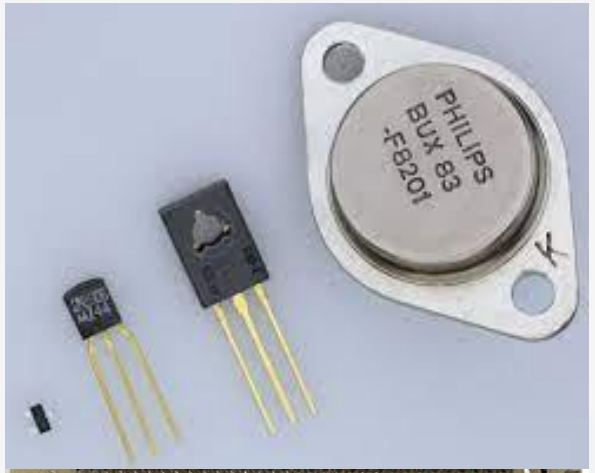
First Generation Computers (1951-1958):

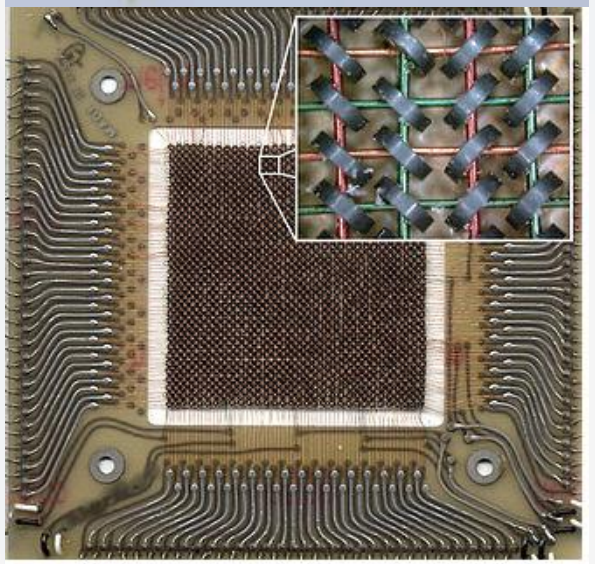


Characteristics of computer generations

Second Generation Computer (1959–1964)

- Used **transistors** for **internal operations**.
- Used a magnetic core as the primary internal storage medium.
  - Increased main storage capacity.
- Fast input and output (associated with magnetic tape).
- High-level programming languages (Fortran, COBOL).
- Significantly reduced size and heat emission.
- Increased speed and reliability.
- Used for batch processing applications (invoice and balance sheet calculations).





Magnetic-core_memory

Characteristics of computer generations

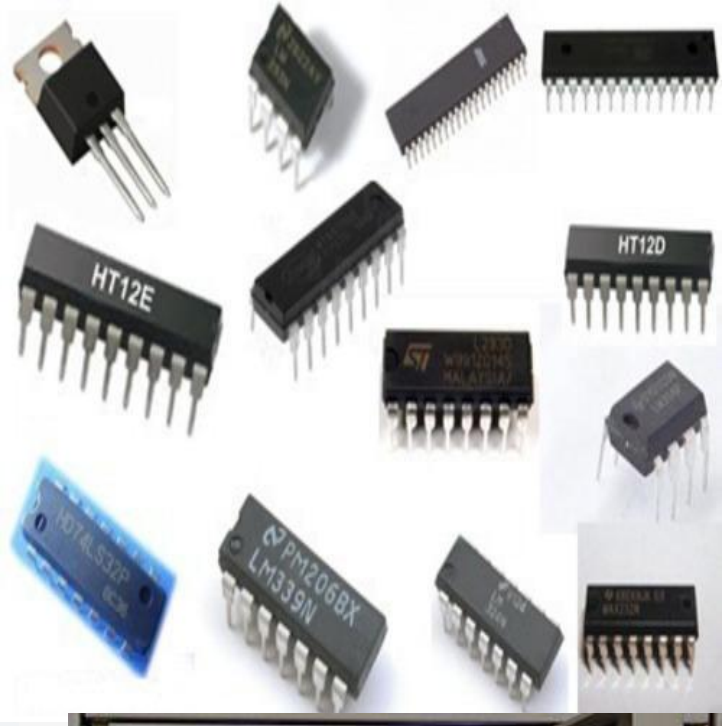
Second Generation Computer (1959–1964)

Property of Museum of History & Industry, Seattle

Characteristics of computer generations

Third Generation Computers (1965–1971):

- Use of **integrated circuits** (ICs).
- Use of magnetic cores and solid-state storage (SSS) for primary storage.
- Decreased size with improved performance and reliability.
- Extensive use of high-level languages (FORTRAN) and their development.
- Emergence of microcomputers (minicomputers).
- Emergence of time-sharing (multitasking) systems.
- Emergence of operating systems to monitor input and output and perform multiple tasks.
- Used for airline reservation systems and market forecasting applications.



Tape Drive

Characteristics of computer generations

Third Generation Computers (1965–1971):

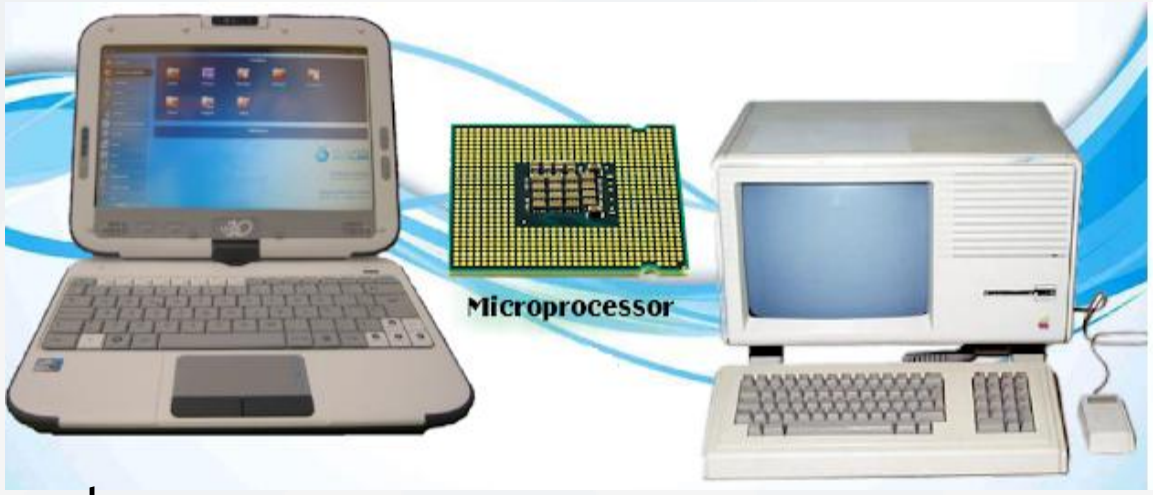
Third generation Computer

Integrated Circuit

Characteristics of computer generations

Fourth Generation Computers (1972 – early 1990s):

- The **emergence** of the **microprocessor**.
- The use of Very Large-Scale Integrated ICs (VLSI).
- Increased storage capacity and increased speed.
- The emergence of general-purpose software.
- The growing of use of minicomputers.
- A wide variety of input and output devices.
- The emergence of the microcomputer.
- Applications of mathematical models, simulations, and electronic transfers of bank deposits, and the use of microcomputers in homes (personal computers).



Characteristics of computer generations

Fifth Generation Computers (early 1990s to present):

- Use of Ultra Large Scale Integrated Circuits (ULSIs).
- Smaller in size and with significantly expanded memory.
- More efficient and faster.
- Equipped with artificial intelligence technologies to attempt to simulate the human mind.
- Knowledge-based information processing systems.
- Use of automated devices (robots) equipped with artificial intelligence.
- Ability to handle natural languages.
- Applications used have diversified and expanded dramatically (home, industrial, automotive, educational, mobile phones, etc.).

How data represented in a computer?

- Since a computer, as we mentioned, is an electronic device, it uses electrical signals (pulses). Computer data is binary digital data, meaning it is represented by only two values, each of which is called a binary digit (bit). The digit represents a specific electrical state of the signal (ON/OFF).
- When expressing these two states in a computer, two symbols are used: zero (0) and one (1), where zero represents the OFF state and one represents the ON state.

How data represented in a computer?

- **Computers do not understand data** that **humans process**, such as letters or numbers. Therefore, this data is encoded in binary digits that the computer processes and understands. The bit (zero or one) is the basic unit of any information.
- Using a set of binary digits, any desired information can be expressed.
- For example, a letter of the alphabet can be expressed using a series of binary digits (bits). The letter "a" can be represented by eight digits (11000110), the number 9 can be expressed by a series of four binary digits (1001), and so on.
- These binary numbers are represented inside the computer by a series of electrical voltage values depending on whether the number is zero or one.

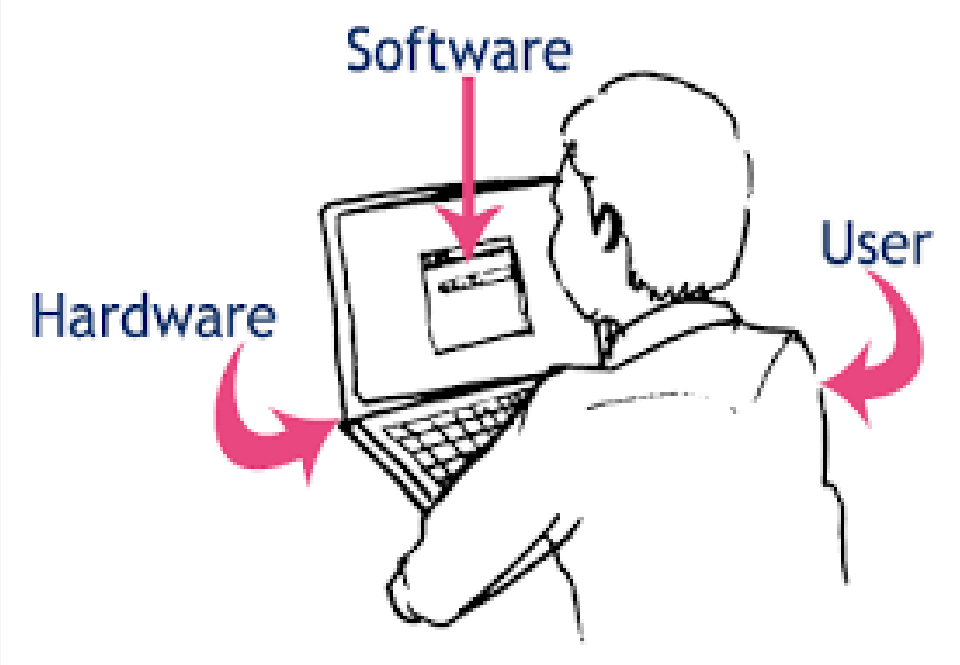
How data represented in a computer?

1 bit	Bit	=	0,1
1 byte	Byte	=	8 bit
1 Kilobyte	KB	=	1024 bytes
1 Megabyte	MB	=	1024 KB
1 Gigabyte	GB	=	1024 MB
1 Terabyte	TB	=	1024 GB
1 Petabyte	PB	=	1024 TB

Main components of a computer system

A computer system consists of three basic parts that work together to accomplish tasks:

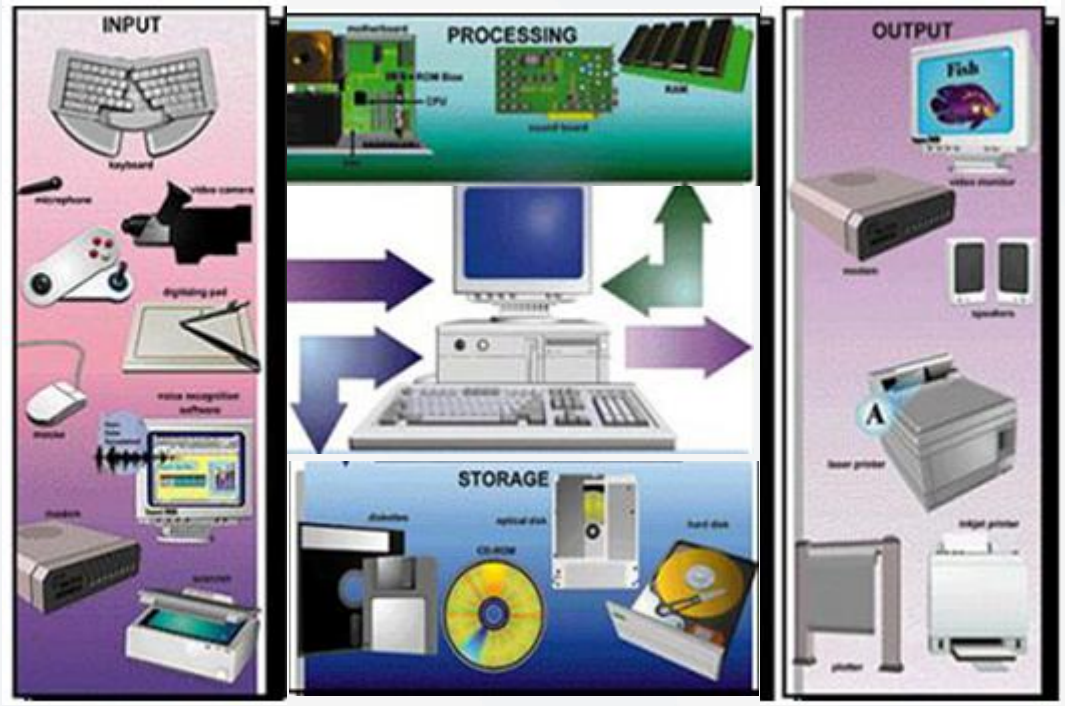
1. The hardware.
2. The software.
3. The users.



Hardware:

This term refers to the physical components that make up a computer system. They are:

1. The central processing unit.
2. The motherboard.
3. The memory unit.
4. The external storage units.
5. The input units.
6. The output units.
7. The communication buses between these units.



Software:

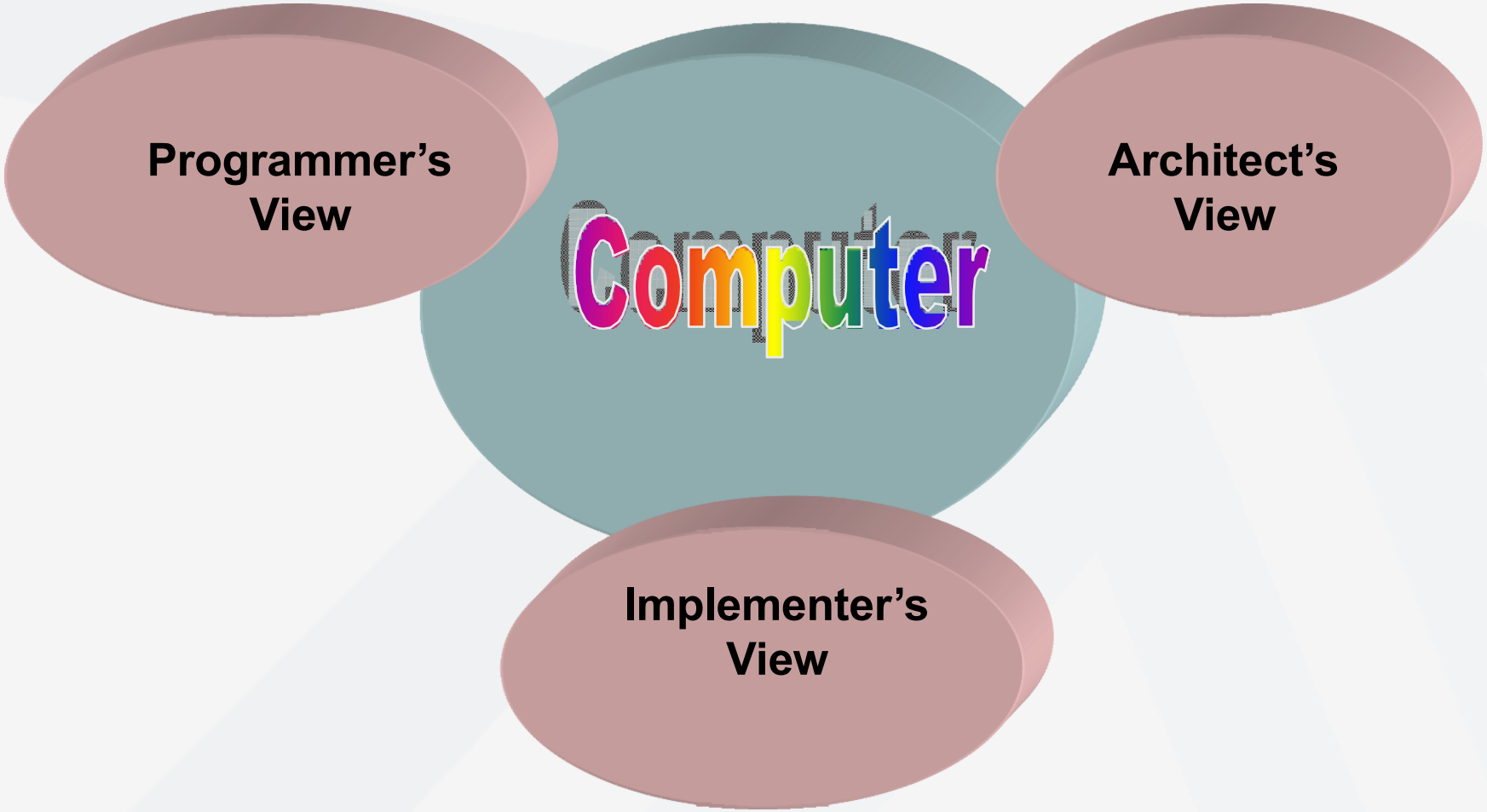
System Software

- Operating Systems
  - Utility Programs
  - Device Drivers
  - Memory Management
  - CPU Management
- Programming Languages
- Compilers and Assemblers

Application Software

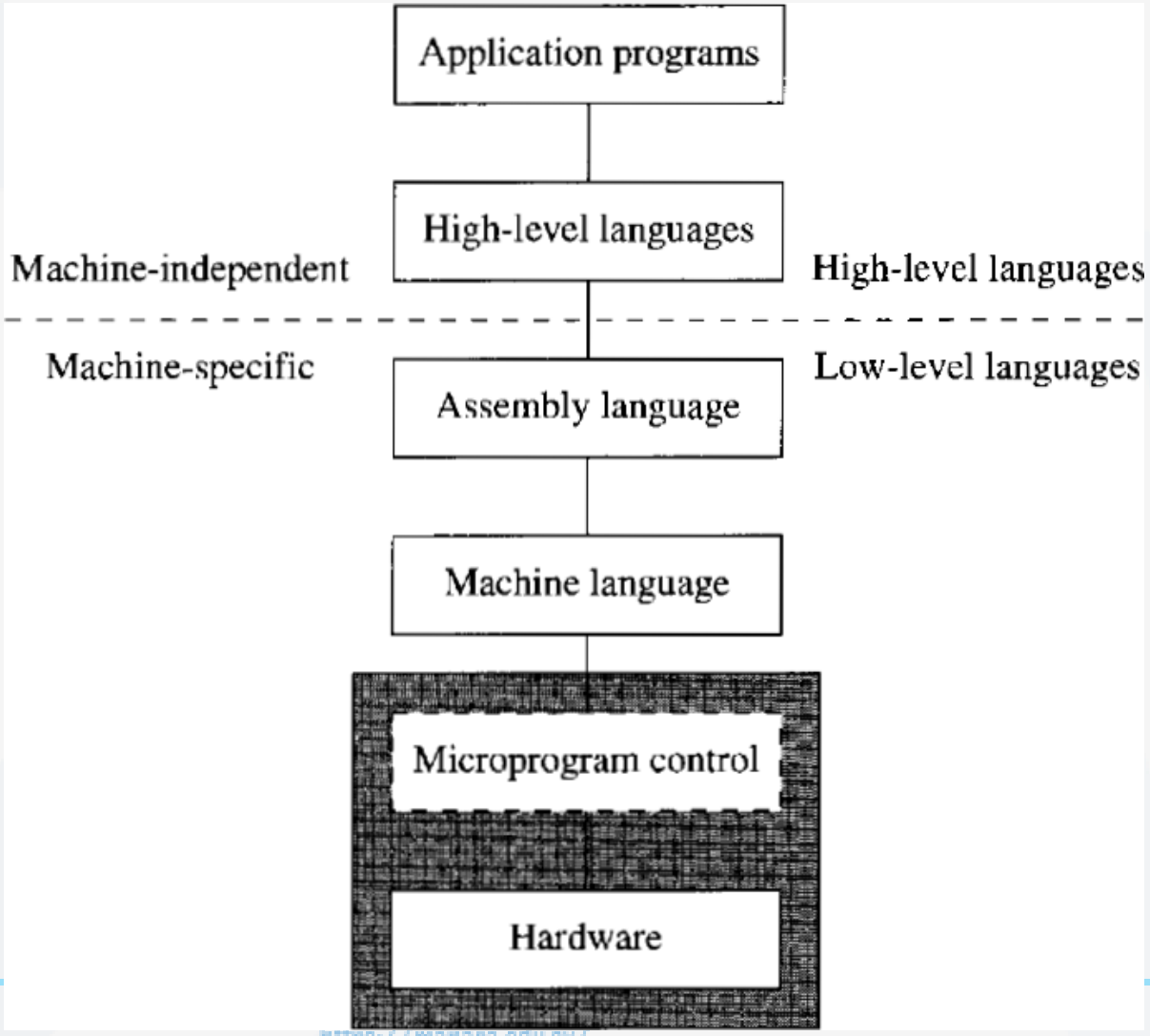
- Text Processing Programs
- Spreadsheet Programs
- Database Programs
- Custom Software
- Library Management System
- Repository System

Computer Systems



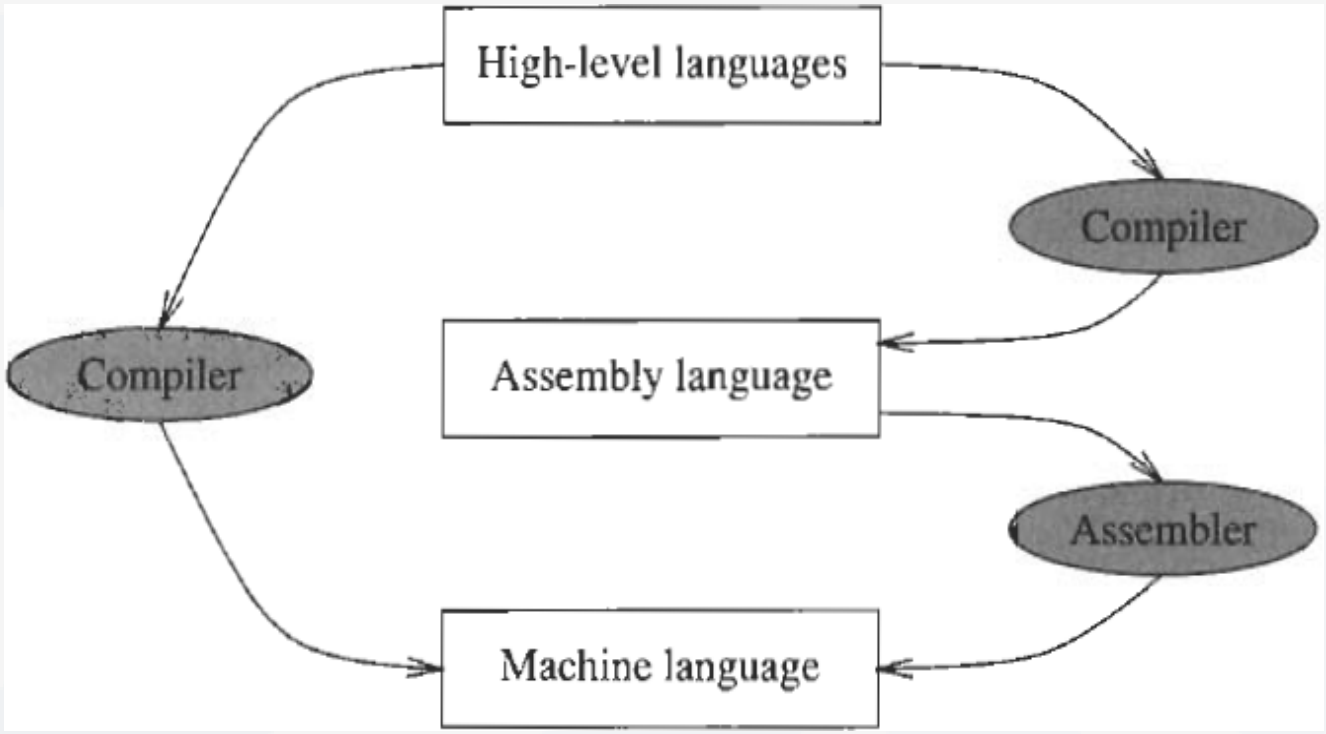
Programmer's View

Programmer's View



Programmer's View

```
Mov   AX,   count1
Add   AX,   count2
Add   AX,   count3
Add   AX,   count4
Mov  result , AX
```

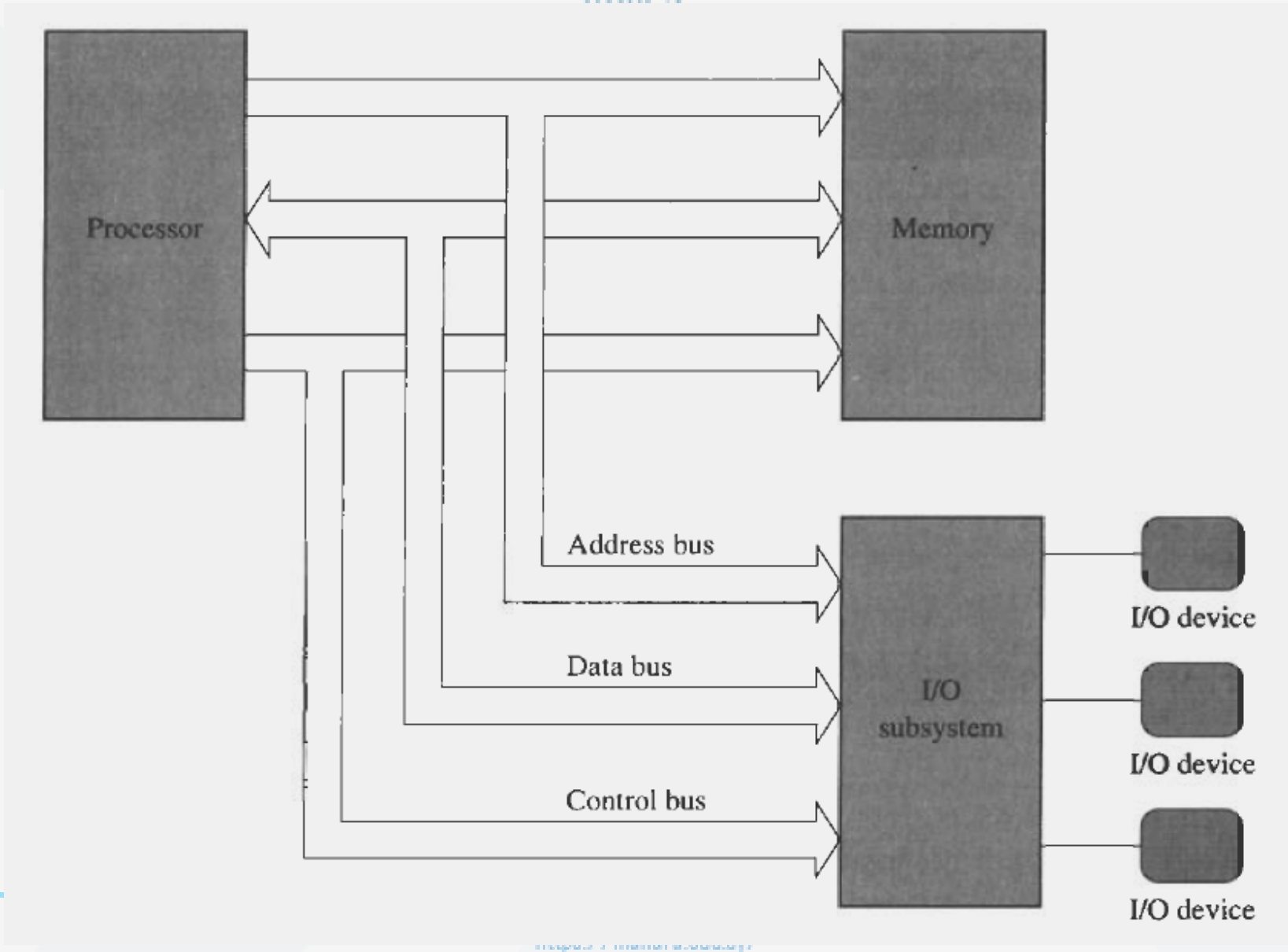


Translation of higher-level languages into machine language is done by assemblers and compilers.
A compiler can translate a high-level language program directly into the machine language, or it can produce the equivalent assembly language.

Programmer's View

- **Machine language**
  - Native to a processor: executed directly by hardware
  - Instructions consist of binary code: 1s and 0s
- **Assembly language**
  - Slightly higher-level language
  - Readability of instructions is better than machine language
  - One-to-one correspondence with machine language instructions
- **Assemblers** translate assembly to machine code
- **Compilers** translate high-level programs to machine code
  - Either directly, or
  - Indirectly via an assembler

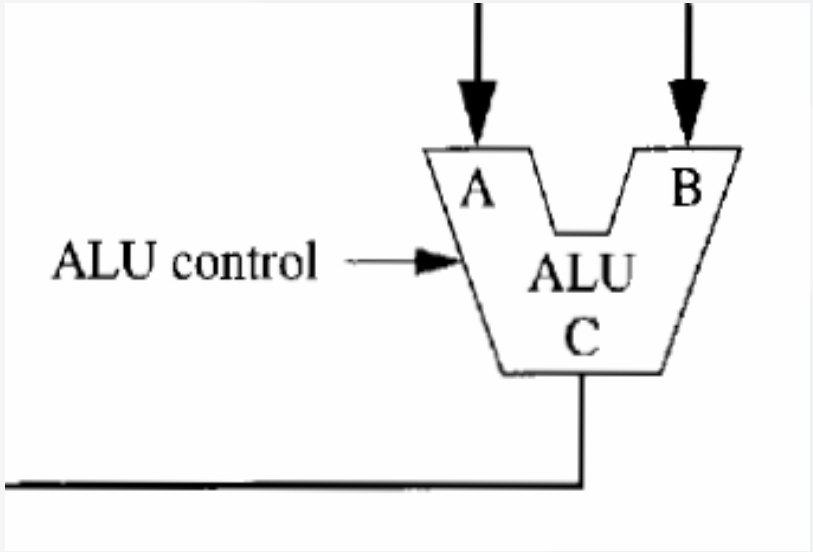
Architect's View

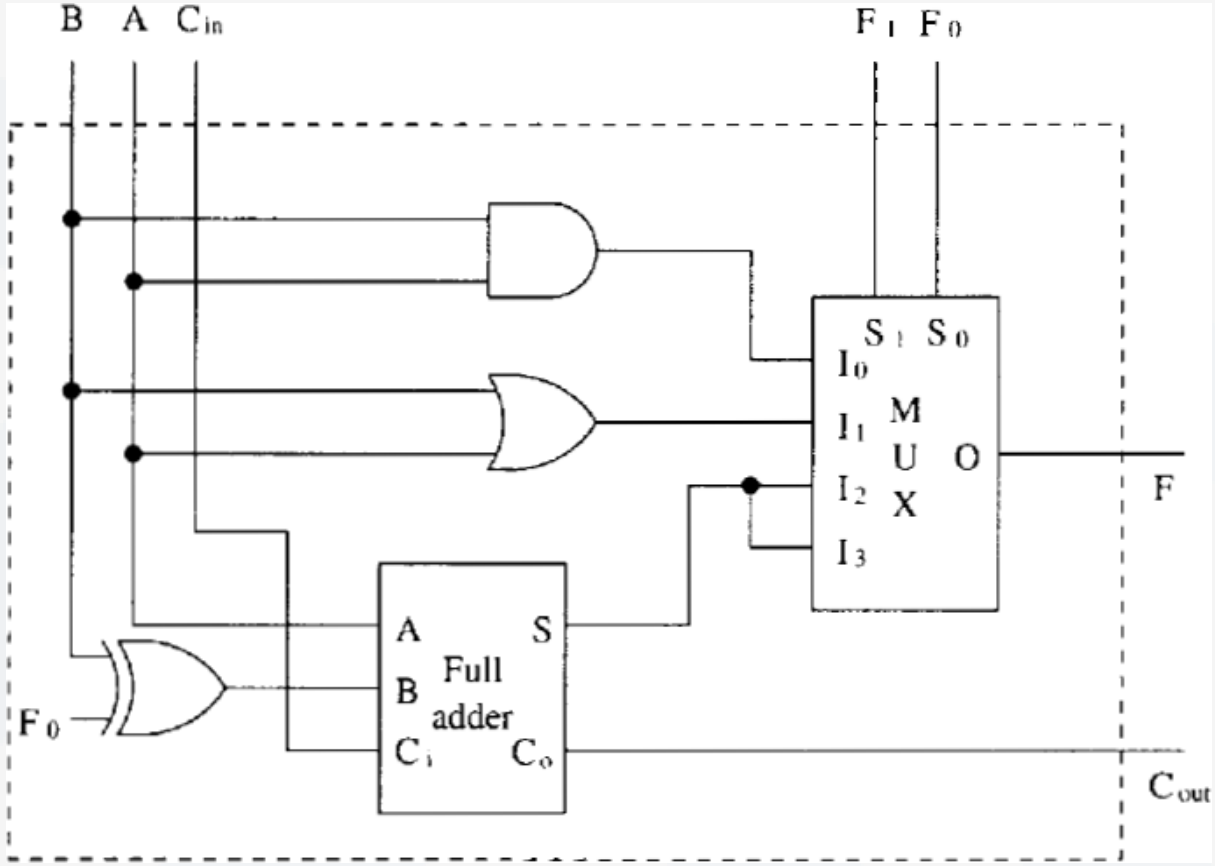


Architect's View

- **Memory**
  - Stores the **instructions** and **data** comprising the program to be executed
- **CPU**
  - **Interprets** and **executes** program instructions in sequence
- **I/O devices**
  - **Communicates** the CPU with the real world
- **System buses**
  - A collection of wires that allow access to the circuitry around the CPU

Implementer's View



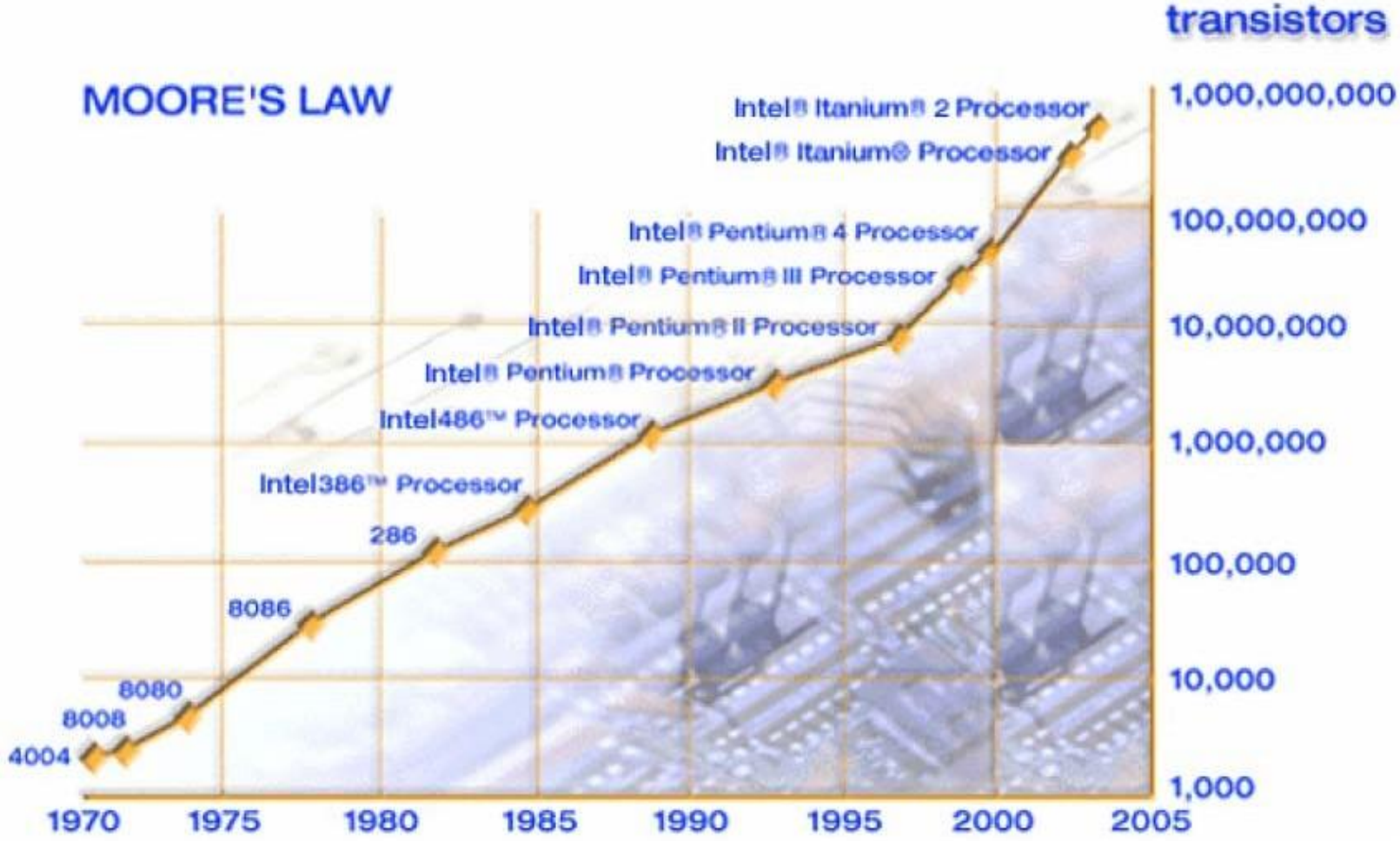


An example I-bit ALU design. It can perform one of four functions, selected by F1,F0 inputs. Implementer's view also includes transistors which constructs the logic gates.

Moore's Law

- The number of transistors on a microchip doubles about every 18-24 months, assuming the price of the chip stays the same.
- The speed of a microprocessor doubles about every 18-24 months, assuming the price of the processor stays the same.
- The price of a microchip drops about 48% every 18-24 months, assuming the same processor speed or on-chip memory capacity.

Moore's Law

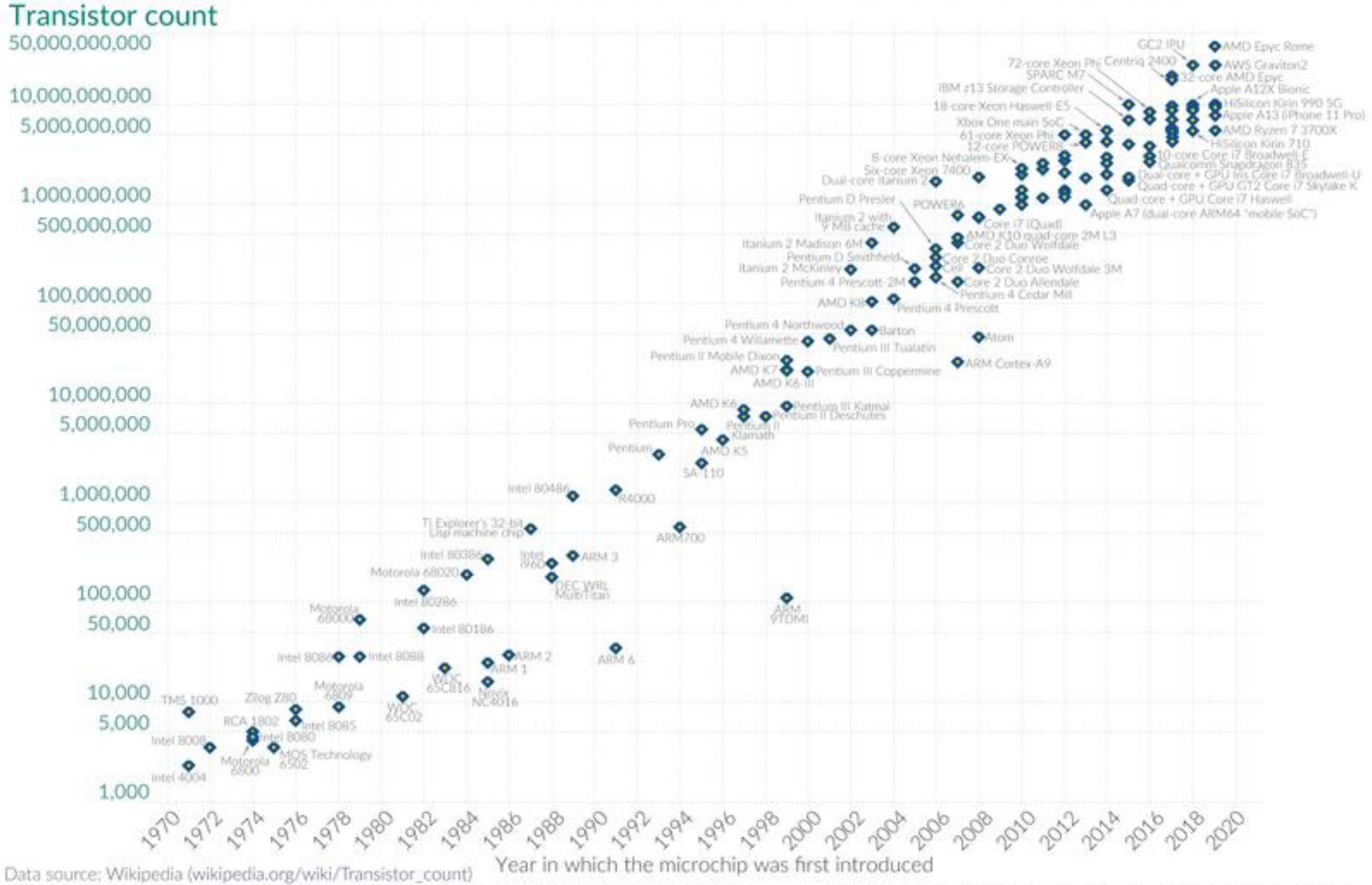


	Year	Transistors
4004	1971	2 250
8008	1972	2 500
8080	1974	5 000
8086	1978	29 000
286	1982	120 000
386™	1985	275 000
486™	1989	1 180 000
Pentium®	1993	3 100 000
Pentium® II	1997	7 500 000
Pentium® III	1999	24 000 000
Pentium® 4	2000	42 000 000
Itanium®	2002	220 000 000
Itanium® 2	2003	410 000 000

Moore's Law: The number of transistors on microchips doubles every two years

Our World in Data

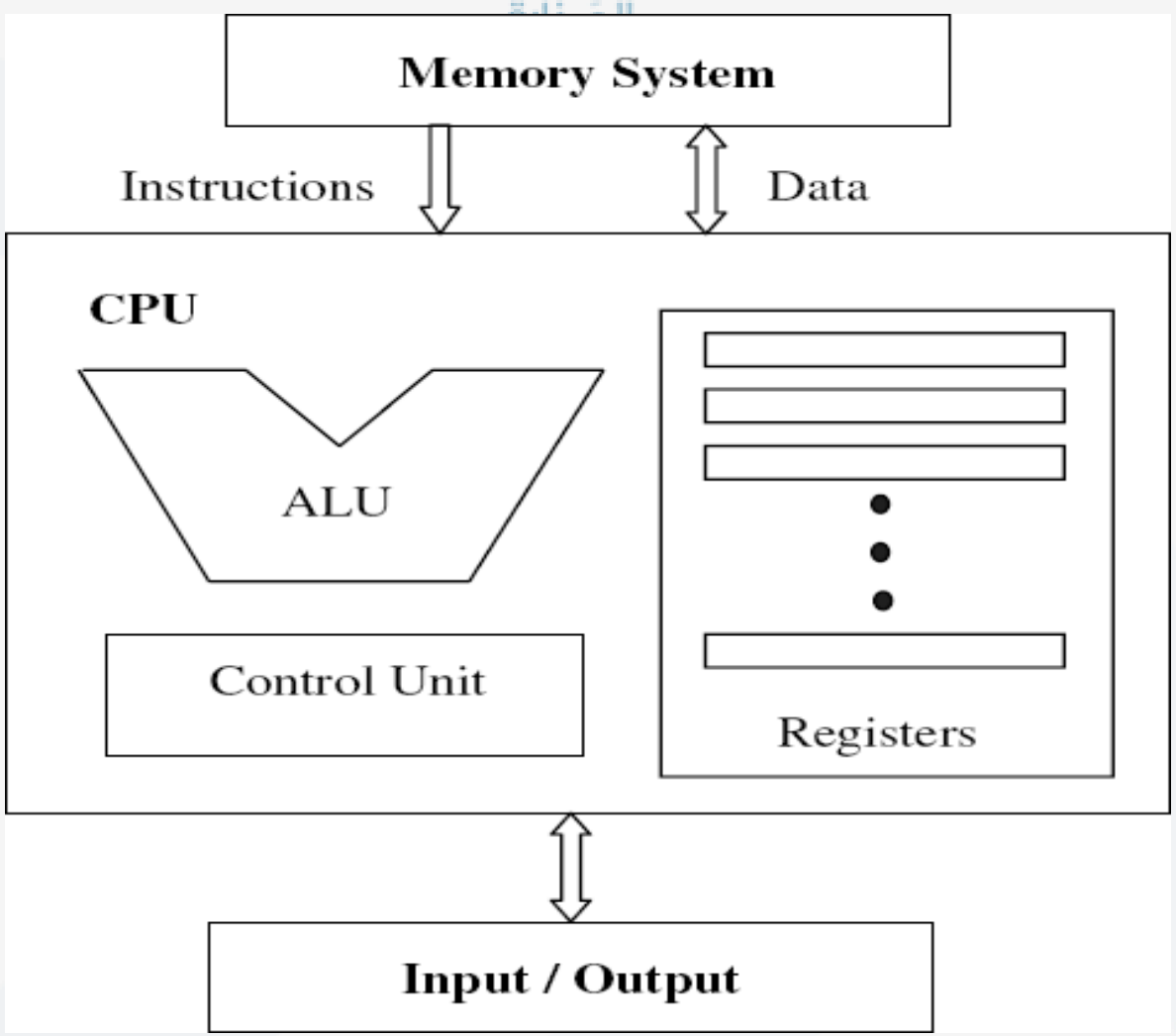
Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.



Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)

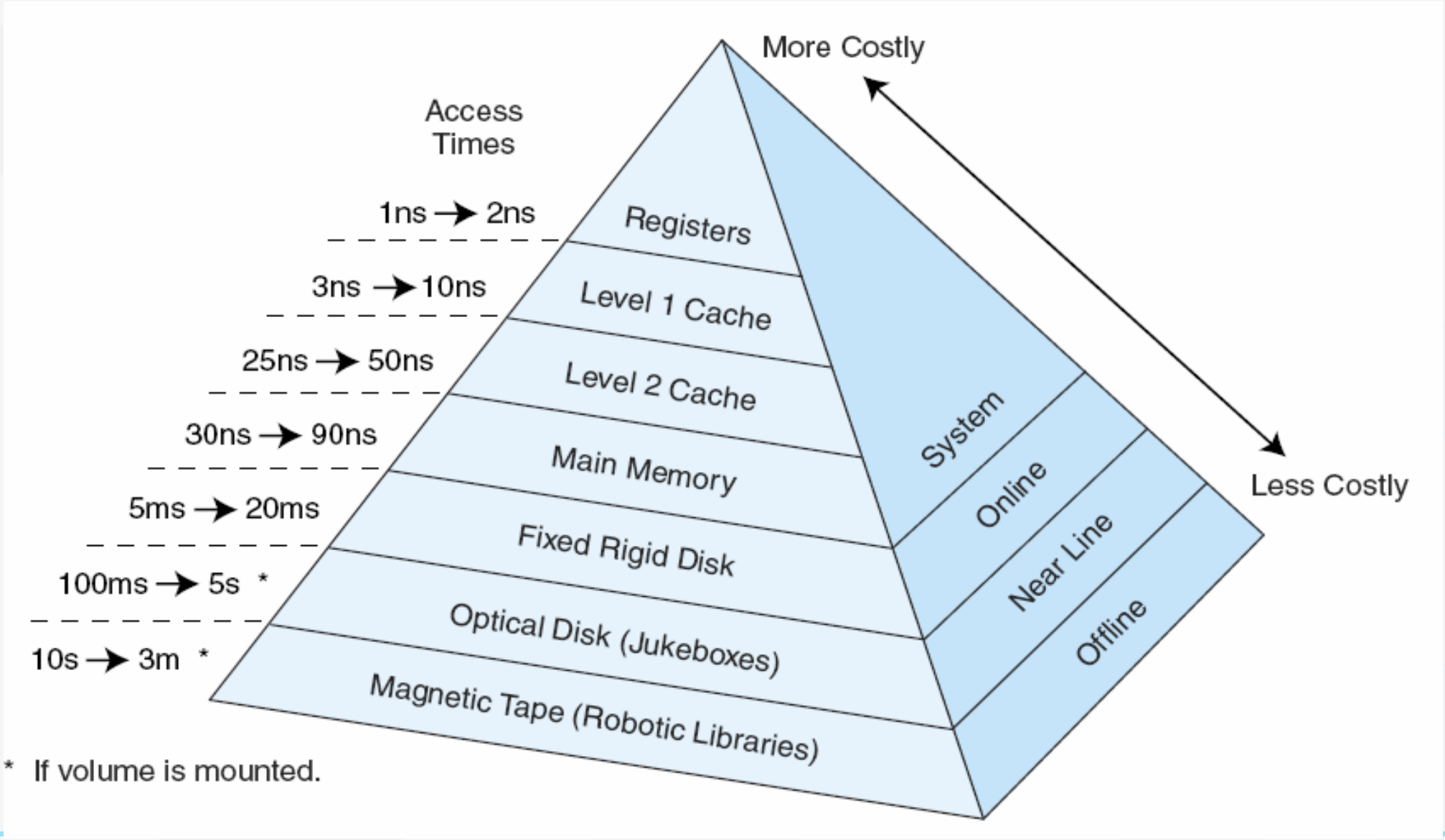
OurWorldinData.org – Research and data to make progress against the world's largest problems.

Computer Architecture



Central processing unit main components and interactions with the memory and I/O

Memory Hierarchy



Micro-processor Vs. Micro-controller

- **Micro-processor (MPU, µP)**
  - CPU alone
  - may contain some memory
  - classified by data path width 4, 8, 16, 32 or 64 bits
- **Micro-controller (MCU)**
  - microprocessor plus peripherals on a single chip
  - a one chip computer system

Assembly vs. Machine Code

Instruction Address	Machine Code	Assembly Instruction
0005	B8 0001	MOV AX, 1
0008	B8 0002	MOV AX, 2
000B	B8 0003	MOV AX, 3
000E	B8 0004	MOV AX, 4
0011	BB 0001	MOV BX, 1
0014	B9 0001	MOV CX, 1
0017	BA 0001	MOV DX, 1
001A	8B C3	MOV AX, BX
001C	8B C1	MOV AX, CX
001E	8B C2	MOV AX, DX
0020	83 C0 01	ADD AX, 1
0023	83 C0 02	ADD AX, 2
0026	03 C3	ADD AX, BX
0028	03 C1	ADD AX, CX
002A	03 06 0000	ADD AX, i
002E	83 E8 01	SUB AX, 1
0031	2B C3	SUB AX, BX
0033	05 1234	ADD AX, 1234h

Translating Languages

English: D is assigned the sum of A times B plus 10.

High-Level Language: D = A * B + 10

A statement in a high-level language is translated typically into several machine-level instructions

Intel Assembly Language:

```
mov   eax, A
mul   B
add   eax, 10
mov   D, eax
```

Intel Machine Language:

```
A1 00404000
F7 25 00404004
83 C0 0A
A3 00404008
```

Advantages of High-Level Languages

- **Program development is faster**
  - **High-level statements: fewer instructions to code**
- **Program maintenance is easier**
  - **For the same above reasons**
- **Programs are portable**
  - **Contain few machine-dependent details**
    - **Can be used with little or no modifications on different machines**
  - **Compiler translates to the target machine language**
  - **However, Assembly language programs are not portable**

Why assembly language?

- Assembly language teaches **how a computer works** at the machine level (i.e. registers).
- Assembly language is not used just to illustrate algorithms, but to demonstrate what is actually happening inside the computer!

Instructions and Machine Language

- Each command of a program is called an instruction (it instructs the computer what to do).
- Computers only deal with binary data, hence the instructions must be in binary format (0s and 1s) .
- The set of all instructions (in binary form) makes up the computer's machine language. This is also referred to as the instruction set.

Instruction Fields

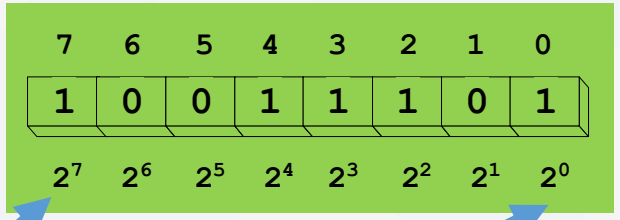
- **Machine language instructions usually are made up of several fields. Each field specifies different information for the computer. The major two fields are:**
- **Opcode field which stands for operation code and it specifies the particular operation that is to be performed.**
  - **Each operation has its unique opcode.**
- **Operands fields which specify where to get the source and destination operands for the operation specified by the opcode.**
  - **The source/destination of operands can be a constant, the memory or one of the general-purpose registers.**

Positional Number Systems

- **Different Representations of Natural Numbers**
- **XXVII Roman numerals (not positional)**
- **27 Radix-10 or decimal number (positional)**
- **11011_2 Radix-2 or binary number (also positional)**
- **Fixed-radix positional representation with *k* digits**
  - **Number *N* in radix *r* = $(d_{k-1}d_{k-2}\ldots d_1d_0)_r$**
  - **Value = $d_{k-1}\times r^{k-1} + d_{k-2}\times r^{k-2} + \ldots + d_1\times r + d_0$**
  - **Examples:**
    - **$(278)_{10} = 2x10^2 + 7x10^1 + 8x10^0$**
    - **$(11011)_2 = 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2 + 1 = 27$**
    - **$(2103)_4 = 2\times4^3 + 1\times4^2 + 0\times4 + 3 = 147$**

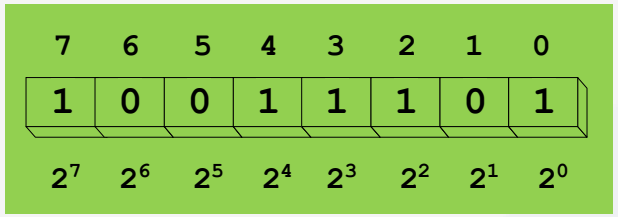
Binary Numbers

- Each binary digit (called bit) is either 1 or 0
- Bits have no inherent meaning, can represent
  - Unsigned and signed integers
  - Characters
  - Floating-point numbers
  - Images, sound, etc.
- Bit Numbering
  - Most significant bit (MSB) is leftmost (bit 7 in an 8-bit number)
  - Least significant bit (LSB) is rightmost (bit 0)



Converting Binary to Decimal

- Each bit represents a power of 2
- Every binary number is a sum of powers of 2
- Decimal Value = $(d_{n-1} \times 2^{n-1}) + \ldots + (d_1 \times 2^1) + (d_0 \times 2^0)$
- Binary $(10011101)_2 = 2^7 + 2^4 + 2^3 + 2^2 + 1 = 157$



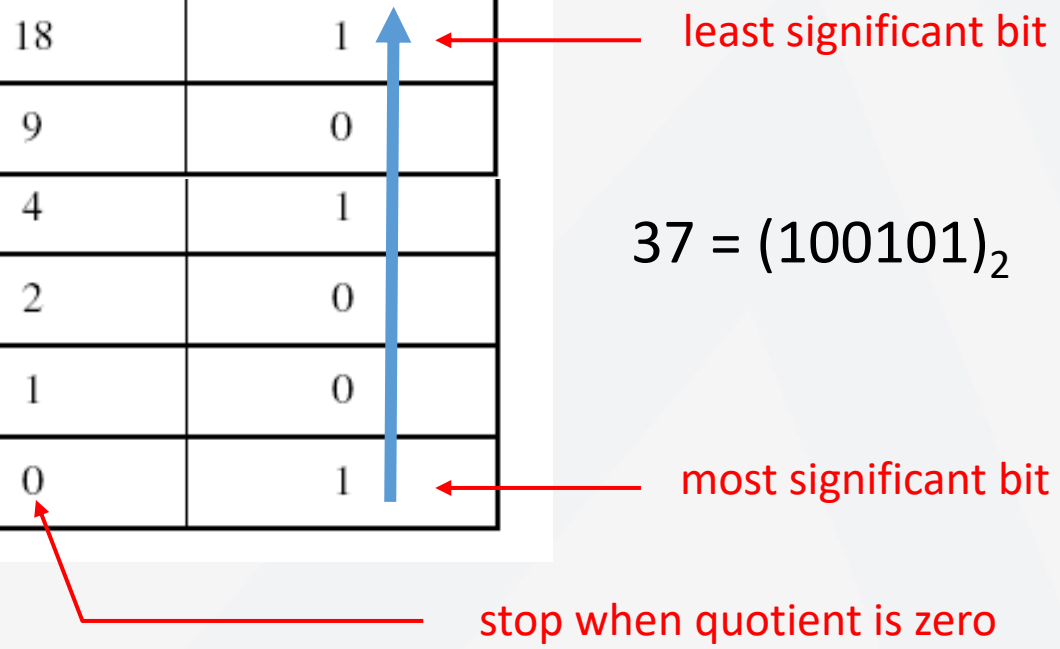
Some common powers of 2

2^n	Decimal Value	2^n	Decimal Value
2^0	1	2^8	256
2^1	2	2^9	512
2^2	4	2^{10}	1024
2^3	8	2^{11}	2048
2^4	16	2^{12}	4096
2^5	32	2^{13}	8192
2^6	64	2^{14}	16384
2^7	128	2^{15}	32768

Convert Unsigned Decimal to Binary

- Repeatedly divide the decimal integer by 2
- Each remainder is a binary digit in the translated value

Division	Quotient	Remainder
37 / 2	18	1
18 / 2	9	0
9 / 2	4	1
4 / 2	2	0
2 / 2	1	0
1 / 2	0	1



$37 = (100101)_2$

Hexadecimal Numbers

- 16 Hexadecimal Digits: 0 – 9, A – F
- More convenient to use than binary numbers

Binary, Decimal, and Hexadecimal Equivalents

Binary	Decimal	Hexadecimal	Binary	Decimal	Hexadecimal
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	10	A
0011	3	3	1011	11	B
0100	4	4	1100	12	C
0101	5	5	1101	13	D
0110	6	6	1110	14	E
0111	7	7	1111	15	F

Converting Binary to Hexadecimal

- Each hexadecimal digit corresponds to 4 binary bits

- Example1:

  Convert the 32-bit binary number to hexadecimal

  1110 1011 0001 0110 1010 0111 1001 0100

- Solution:

E	B	1	6	A	7	9	4
1110	1011	0001	0110	1010	0111	1001	0100

- Example2:

  Convert the hexadecimal number EB16A794 to binary

Converting Hexadecimal to Decimal

Multiply each digit by its corresponding power of 16

$$\text{Value} = (d_{n-1} \times 16^{n-1}) + (d_{n-2} \times 16^{n-2}) + \ldots + (d_1 \times 16) + d_0$$

Examples:

$$(1234)_{16} = (1 \times 16^3) + (2 \times 16^2) + (3 \times 16^1) + (4 \times 16^0) =$$

Decimal Value 4660

$$(3BA4)_{16} = (3 \times 16^3) + (11 \times 16^2) + (10 \times 16^1) + (4 \times 16^0) =$$

Decimal Value 15268

Converting Decimal to Hexadecimal

- Repeatedly divide the decimal integer by 16
- Each remainder is a hex digit in the translated value

Division	Quotient	Remainder
422 / 16	26	6
26 / 16	1	A
1 / 16	0	1

least significant digit

most significant digit

stop when quotient is zero

Decimal 422 = 1A6 hexadecimal

Fractional Numbers

$0.1253)_{10}=0.00100000000)_2$

$11.101)_2=(1\times2^1)+(1\times2^0)+(1\times2^{-1})+(0\times2^{-2})+(1\times2^{-3})$

$11.101)_2=2+1+0.5+0+0.125$

$11.101)_2=3+0.625$

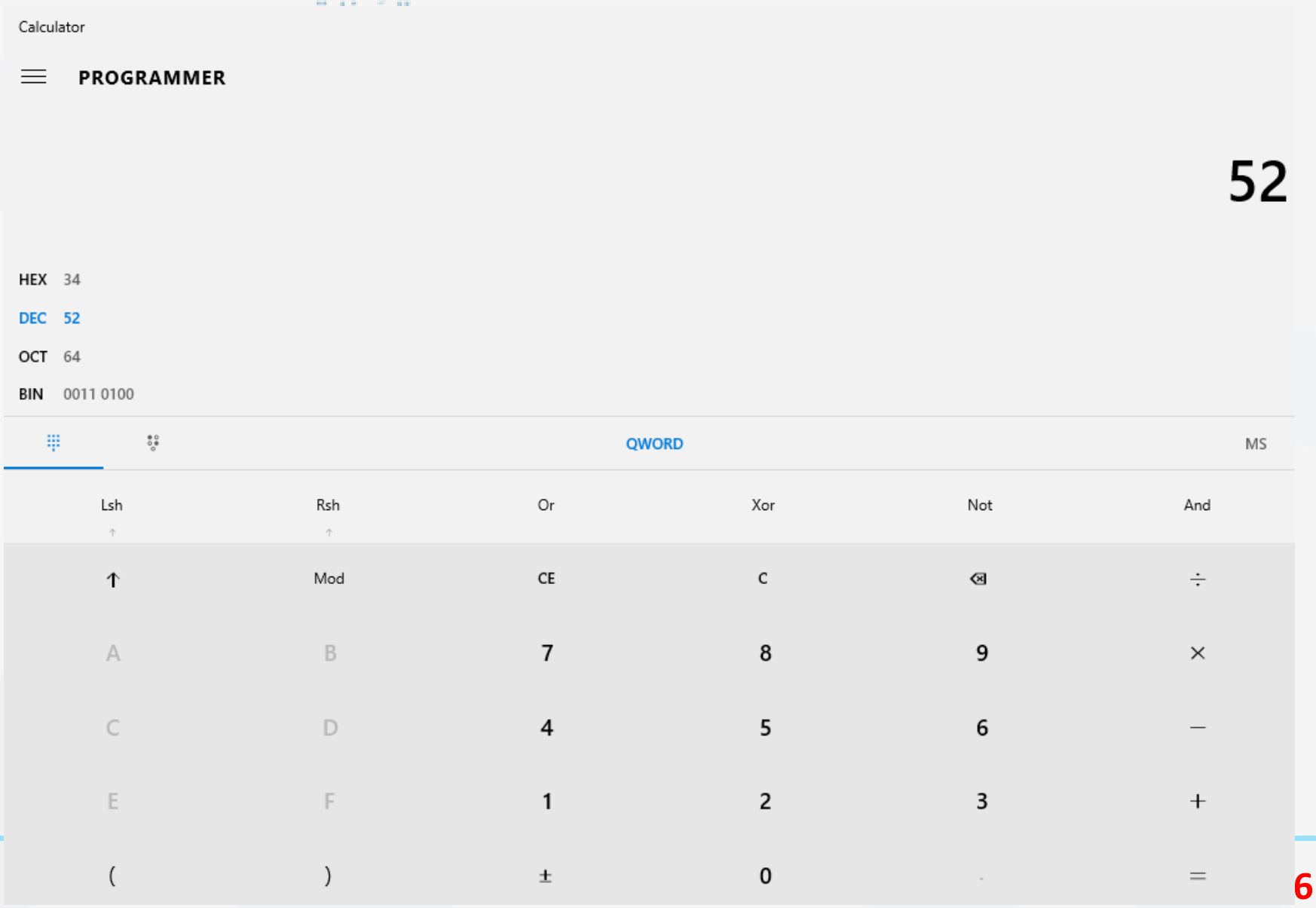
$11.101)_2=3.625_{10}$

$10101011.11001101)_2=AB.CD)_{16}$

Decimal Number				Answer	Ineger Part
0.1253	x	2	=	0.2506	0
0.2506	x	2	=	0.5012	0
0.5012	x	2	=	1.0024	1
0.0024	x	2	=	0.0048	0
0.0048	x	2	=	0.0096	0
0.0096	x	2	=	0.0192	0
0.0192	x	2	=	0.0384	0
0.0384	x	2	=	0.0768	0
0.0768	x	2	=	0.1536	0
0.1536	x	2	=	0.3072	0
0.3072	x	2	=	0.6144	0

Convert Numbers tools

- Microsoft Windows calculator



Convert Numbers tools

https://ncalculators.com/digital-computation/binary-decimal-converter.htm

$19)_{10}=10011)_2$

$2587)_{10}=101000011011)_2$

$1110010100)_2=916)_{10}$

$111001101)_2=461)_{10}$

Fractional Numbers

$128.85)_{10}=10000000.11011001100)_2$

$43.27)_{10}=101011.01000101)_2$

https://ncalculators.com/digital-computation/binary-hex-converter.htm

$001111110010)_2=3F2)_{16}$

$101011001001)_2=AC9)_{16}$

$3AB)_{16}=001110101011)_2$

$ABC6)_{16}=1010101111000110)_2$

$A.B)_{16}=1010.1011)_2$

https://ncalculators.com/digital-computation/hex-decimal-converter.htm

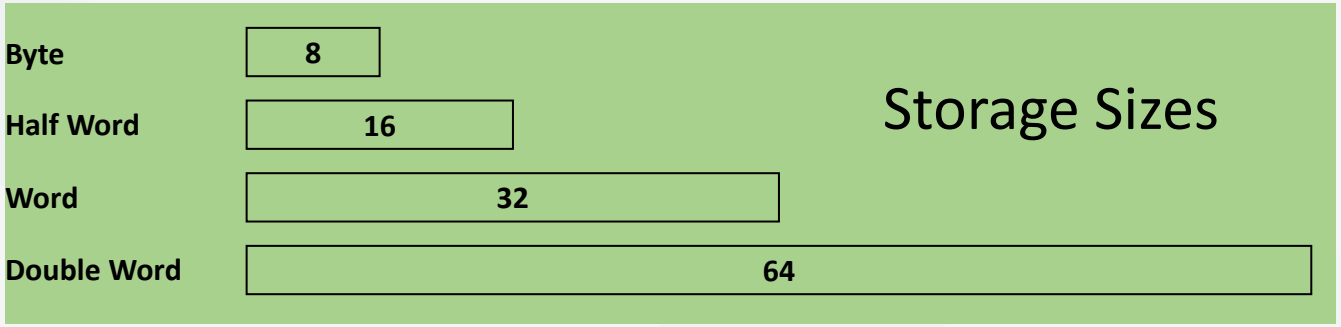
$1523)_{10}=5F3)_{16}$

$253249)_{10}=3DD41)_{16}$

$8C1)_{16}=2241)_{10}$

$FE3A_{16}=65082_{10}$

Integer Storage Sizes



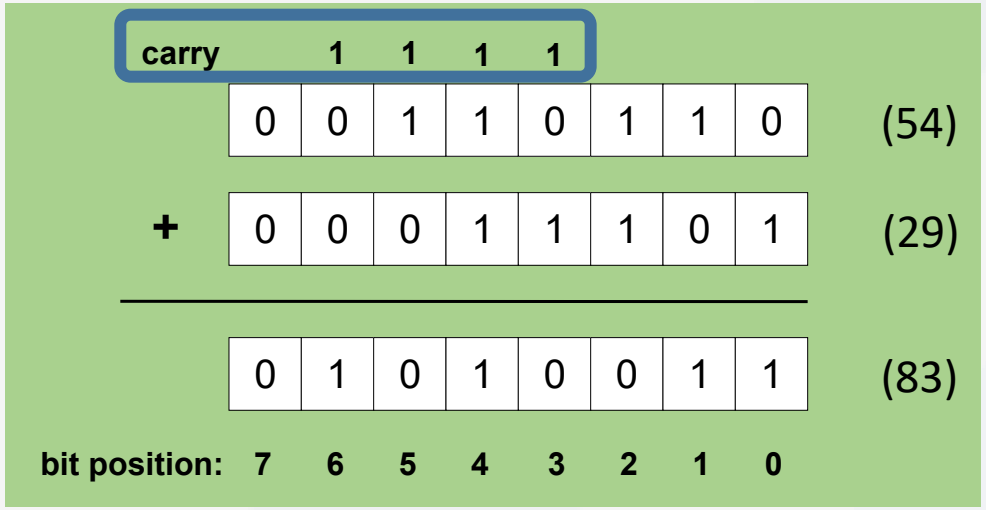
Storage Type	Unsigned Range	Powers of 2
Byte	0 to 255	0 to $(2^8 - 1)$
Half Word	0 to 65,535	0 to $(2^{16} - 1)$
Word	0 to 4,294,967,295	0 to $(2^{32} - 1)$
Double Word	0 to 18,446,744,073,709,551,615	0 to $(2^{64} - 1)$

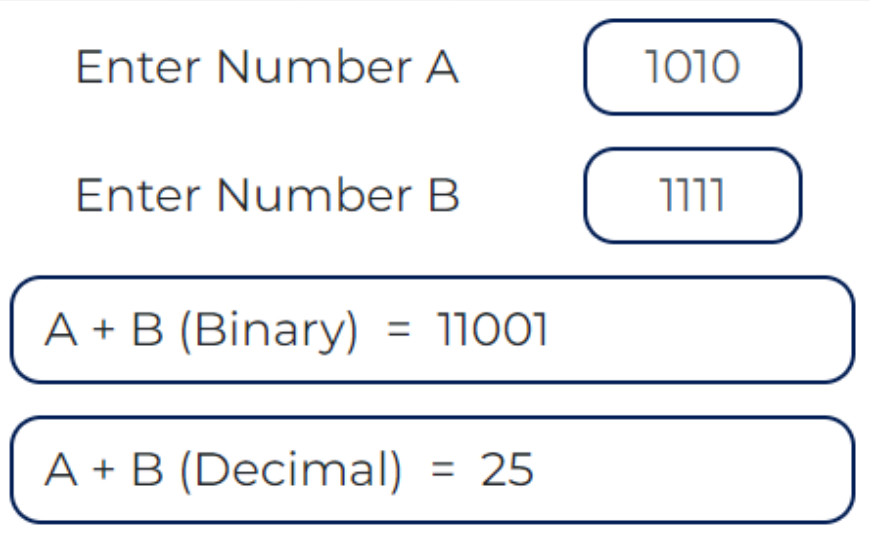
What is the largest 20-bit unsigned integer?

Answer: $2^{20} - 1 = 1{,}048{,}575$

Binary Addition

- Start with the least significant bit (rightmost bit)
- Add each pair of bits
- Include the carry in the addition, if present





Hexadecimal Addition

- Start with the least significant hexadecimal digits
- Let Sum = summation of two hex digits
- If Sum is greater than or equal to 16
  - Sum = Sum – 16 and Carry = 1
- Example:

```
carry:          1  1     1
      1 C 3 7 2 8 6 A
    + 9 3 9 5 E 8 4 B
    -----------------
      A F C D 1 0 B 5
```

A + B = 10 + 11 = 21
Since 21 ≥ 16
Sum = 21 – 16 = 5
Carry = 1

Signed Integers

- Several ways to represent a signed number
- Divide the range of values into 2 equal parts
  - First part corresponds to the positive numbers (≥ 0)
  - Second part correspond to the negative numbers (< 0)
- Focus will be on the 2's complement representation
  - Has many advantages over other representations
  - Used widely in processors to represent signed integers

Forming the Two's Complement

starting value	`00100100 = +36`
step1: reverse the bits (1's complement)	`11011011`
step 2: add 1 to the value from step 1	`+ 1`
sum = 2's complement representation	`11011100 = -36`

Sum of an integer and its 2's complement must be zero:

00100100 + 11011100 = 00000000 (8-bit sum) ⇒ Ignore Carry

Another way to obtain the 2's complement:

Start at the least significant 1

Leave all the 0s to its right unchanged

Complement all the bits to its left

```
Binary Value
= 00100100
2's Complement
= 11011100
```

least significant 1

Two's Complement Representation

- Positive numbers
  - Signed value = Unsigned value

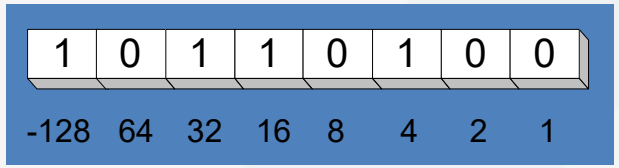
    $00001111)_2=15)_{10}$ (Unsigned)

    $00001111)_2=15)_{10}$ (signed)
- Negative numbers
  - Signed value = Unsigned value – 2^n

    n = number of bits

    $11111000)_2=248$ (Unsigned)

    $11111000)_2=248-256=-8$ (Signed)



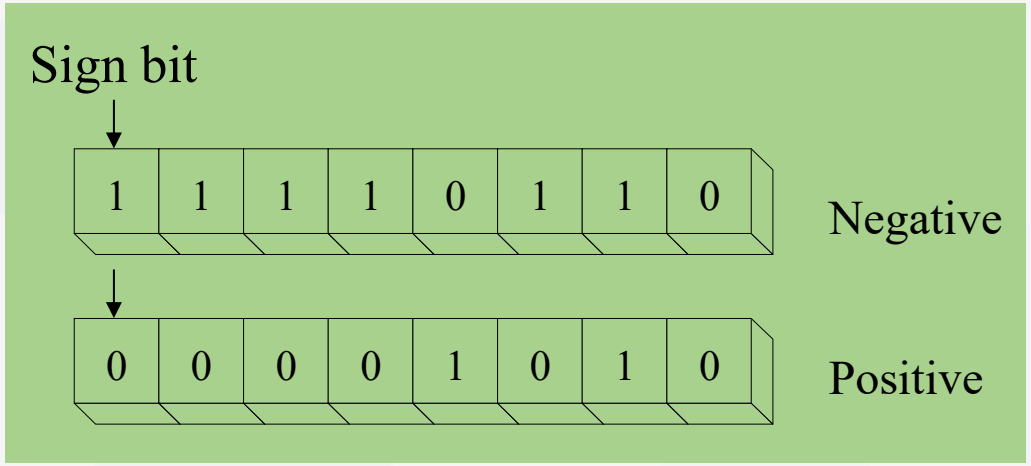
- Negative weight for MSB

  = -128 + 32 + 16 + 4 = -76
  - Another way to obtain the signed value is to assign a negative weight to most-significant bit

8-bit Binary value	Unsigned value	Signed value
00000000	0	0
00000001	1	+1
00000010	2	+2
. . .	. . .	. . .
01111110	126	+126
01111111	127	+127
10000000	128	-128
10000001	129	-127
. . .	. . .	. . .
11111110	254	-2
11111111	255	-1

Sign Bit

- Highest bit indicates the sign
- 1 = negative
- 0 = positive



For Hexadecimal Numbers, check most significant digit

If highest digit is > 7, then value is negative

Examples: 8A and C5 are negative bytes

B1C42A00 is a negative word (32-bit signed integer)

Sign Extension

Step 1: Move the number into the lower-significant bits

Step 2: Fill all the remaining higher bits with the sign bit

- This will ensure that both magnitude and sign are correct
- Examples
  - Sign-Extend 10110011 to 16 bits

    10110011 = -77 → **11111111 10110011 = -77**
  - Sign-Extend 01100010 to 16 bits

    01100010 = +98 → **00000000 01100010 = +98**
- Infinite 0s can be added to the left of a positive number
- Infinite 1s can be added to the left of a negative number

Binary Subtraction

- When subtracting A – B, convert B to its 2's complement
- Add A to (–B)

borrow: 1 1 1

0 1 0 0 1 1 0 1

− 0 0 1 1 1 0 1 0

0 0 0 1 0 0 1 1

carry: 1 1 1 1

0 1 0 0 1 1 0 1

+ 1 1 0 0 0 1 1 0 (2's complement)

0 0 0 1 0 0 1 1 (same result)

- Final carry is ignored.

Ranges of Signed Integers

For n-bit signed integers: Range is -2^{n-1} to $(2^{n-1} - 1)$

Positive range: 0 to $2^{n-1} - 1$

Negative range: -2^{n-1} to -1

Storage Type	Unsigned Range	Powers of 2
Byte	–128 to +127	-2^7 to $(2^7 - 1)$
Half Word	–32,768 to +32,767	-2^{15} to $(2^{15} - 1)$
Word	–2,147,483,648 to +2,147,483,647	-2^{31} to $(2^{31} - 1)$
Double Word	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	-2^{63} to $(2^{63} - 1)$

Practice: What is the range of signed values that may be stored in 20 bits?

Carry and Overflow

- Overflow and carry out are philosophically the same thing. Both indicate that the answer does not fit in the space available. Carry out is generated, when there is a carry out of the most-significant bit. The overflow happens when there is a carry into the most significant bit.
- Carry Out: It can be observed in the case of unsigned number arithmetic. However, with the generation of carry out, the sum does not get corrupted. The carry flag is set. The content of the register holding the result along with the carry provides the correct sum.
- Overflow: It can be observed in the case of signed number arithmetic. However, with the generation of overflow, the sum gets corrupted. The overflow flag is set, which indicates the result is incorrect.

- *https://www.vlsifacts.com/what-is-overflow-in-case-of-binary-arithmetic/#:~:text=Overflow%20and%20carry%20out%20are,into%20the%20most%20significant%20bit.*

Carry and Overflow

- Carry is important when …
  - Adding or subtracting unsigned integers
  - Indicates that the unsigned sum is out of range
- Overflow is important when …
  - Adding or subtracting signed integers
  - Indicates that the signed sum is out of range
- Overflow occurs when
  - Adding two positive numbers and the sum is negative
  - Adding two negative numbers and the sum is positive
  - Can happen because of the fixed number of sum bits

Carry and Overflow Examples

- We can have carry without overflow and vice-versa
- Four cases are possible (Examples are 8-bit numbers)

				1					
+	0	0	0	0	1	1	1	1	15
	0	0	0	0	1	0	0	0	8
	0	0	0	1	0	1	1	1	23

Carry = 0 Overflow = 0

	1	1	1	1	1				
+	0	0	0	0	1	1	1	1	15
	1	1	1	1	1	0	0	0	248 (-128+120=-8)
	0	0	0	0	0	1	1	1	7

Carry = 1 Overflow = 0

	1								
+	0	1	0	0	1	1	1	1	79
	0	1	0	0	0	0	0	0	64
	1	0	0	0	1	1	1	1	79+64=143 (-128+17= -113)

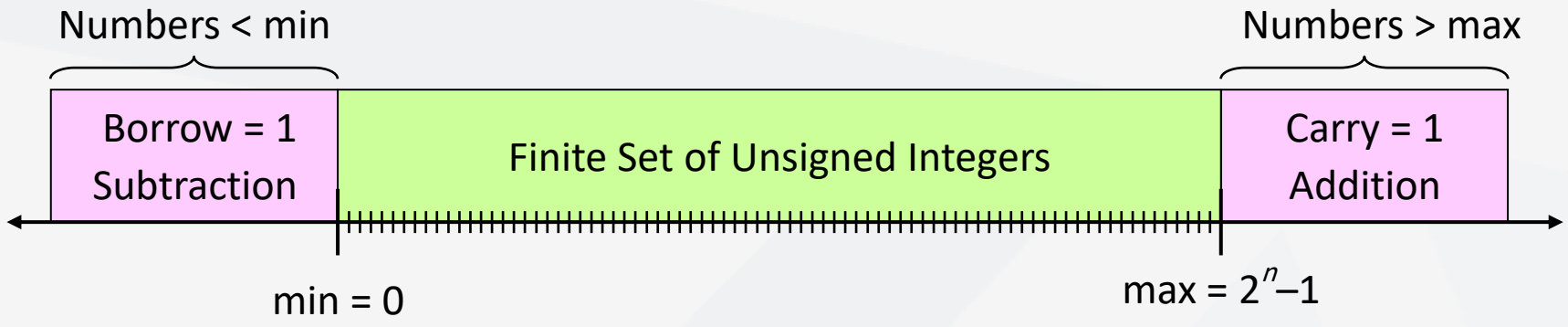
Carry = 0 Overflow = 1

	1				1	1			
+	1	1	0	1	1	0	1	0	218 (-128+90=-38)
	1	0	0	1	1	1	0	1	157 (-128+29=-99)
	0	1	1	1	0	1	1	1	-38+-99=119

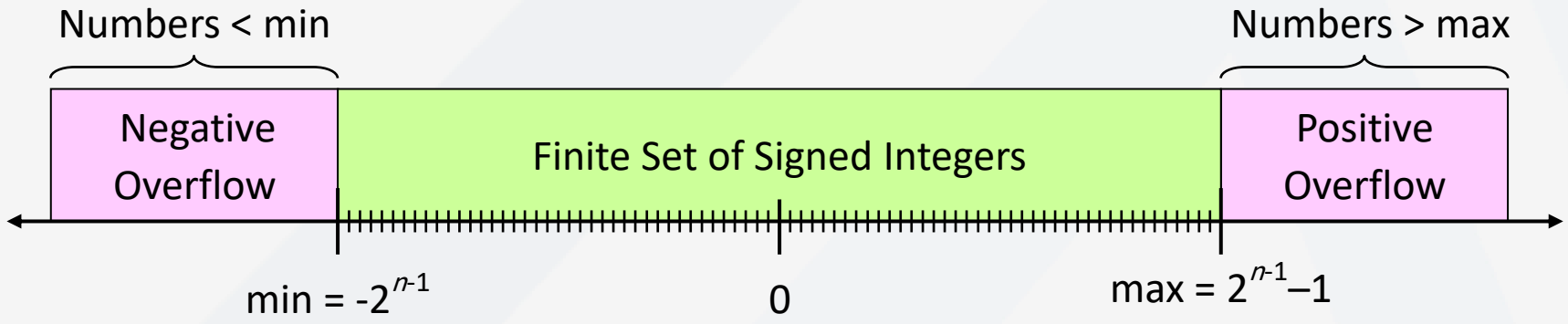
Carry = 1 Overflow = 1

Range, Carry, Borrow, and Overflow

- Unsigned Integers: *n*-bit representation



- Signed Integers: *n*-bit 2's complement representation



Character Storage

- Character sets
  - Standard ASCII: 7-bit character codes (0 – 127)
  - Extended ASCII: 8-bit character codes (0 – 255)
  - Unicode: 16-bit character codes (0 – 65,535)
  - Unicode standard represents a universal character set
    - Defines codes for characters used in all major languages
    - Used in Windows-XP: each character is encoded as 16 bits
  - UTF-8: variable-length encoding used in HTML
    - Encodes all Unicode characters
    - Uses 1 byte for ASCII, but multiple bytes for other characters
- Null-terminated String
  - Array of characters followed by a NULL character

Printable ASCII Codes

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	space	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

- Examples:
  - ASCII code for space character = 20 (hex) = 32 (decimal)
  - ASCII code for 'L' = 4C (hex) = 76 (decimal)
  - ASCII code for 'a' = 61 (hex) = 97 (decimal)

Control Characters

- The first 32 characters of ASCII table are used for control
- Control character codes = 00 to 1F (hexadecimal)
  - Not shown in previous slide
- Examples of Control Characters
  - Character 0 is the NULL character ⇒ used to terminate a string
  - Character 9 is the Horizontal Tab (HT) character
  - Character 0A (hex) = 10 (decimal) is the Line Feed (LF)
  - Character 0D (hex) = 13 (decimal) is the Carriage Return (CR)
  - The LF and CR characters are used together
    - They advance the cursor to the beginning of next line
- One control character appears at end of ASCII table
  - Character 7F (hex) is the Delete (DEL) character

Lecture References:

- **Advanced Computer Architecture and Parallel Processing, H. El-Rewini, M. Abo-El-Barr, Wiley, 2005**
- **Introduction to Parallel Computing, 2nd, A. Grama, A.Gupta, G. Karypis, V. Kumar, Addison Wesley, 2003**
- **COE 308, Computer Architecture Course,Prof. Muhamed Mudawar, College of Computer Sciences and Engineering, King Fahd University of Petroleum and Minerals, 2012.**