COMPUTER SYSTEM ARCHITECTURE

Lecture Slides

Dr. Nisha Chaurasia

WHAT IS COMPUTER SYSTEM ARCHITECTURE

- Computer architecture is concerned with the structure and behavior of the various functional modules of the computer and how they interact to provide the processing needs of the user.
- Computer organization is concerned with the way the hardware components are connected together to form a computer system.
- Computer design is concerned with the development of the hardware for the computer taking into consideration a given set of specifications.

SOURCE: MORRIS MANO, THIRD EDITION

CHAPTER-1
DIGITAL LOGIC CIRCUITS

- Introduces the fundamental knowledge needed for the design of digital systems constructed with individual gates and flip-flops.
- It covers Boolean algebra, combinational circuits, and sequential circuits, providing necessary background for understanding the digital circuits.

SOURCE: MORRIS MANO, THIRD EDITION

SOURCE: MORRIS MANO, THIRD EDITION

- The digital computer is a digital system that performs various computational tasks.
- The word digital implies that the information in the computer is represented by variables that take a limited number of discrete values.
- For e.g., the decimal digits 0, 1, 2, ... , 9, for example, provide 10 discrete values.
- In practice, digital computers function more reliably if only two states are used. Because of the physical restriction of components, and because human logic tends to be binary (i.e., true/false, yes/no statements), digital components that are constrained to take discrete values are further constrained to take only two values and are said to be binary.

SOURCE: MORRIS MANO, THIRD EDITION

- Digital computers use the binary number system, which has two digits: 0 and 1. A binary digit is called a bit.
- Information is represented in digital computers in groups of bits.
- By using various coding techniques, groups of bits can be made to represent not only binary numbers but also other discrete symbols, such as decimal digits or letters of the alphabet.

SOURCE: MORRIS MANO, THIRD EDITION

- In contrast to the common decimal numbers that employ the base 10 system, binary numbers use a base 2 system with two digits: 0 and 1.
- For example, the binary number 1001011 represents a quantity that can be converted to a decimal number by multiplying each bit by the base 2 raised to an integer power as follows:

$$1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 75$$

- Hence, $(1001011)_2 = 75_{10}$

SOURCE: MORRIS MANO, THIRD EDITION

PROGRAM

- A computer system is sometimes subdivided into two functional entities: hardware and software.
- The hardware of the computer consists of all the electronic components and electromechanical devices that comprise the physical entity of the device.
- Computer software consists of the instructions and data that the computer manipulates to perform various data-processing tasks.
- A sequence of instructions for the computer is called a program.
- The data that are manipulated by the program constitute the data base.

OPERATING SYSTEM

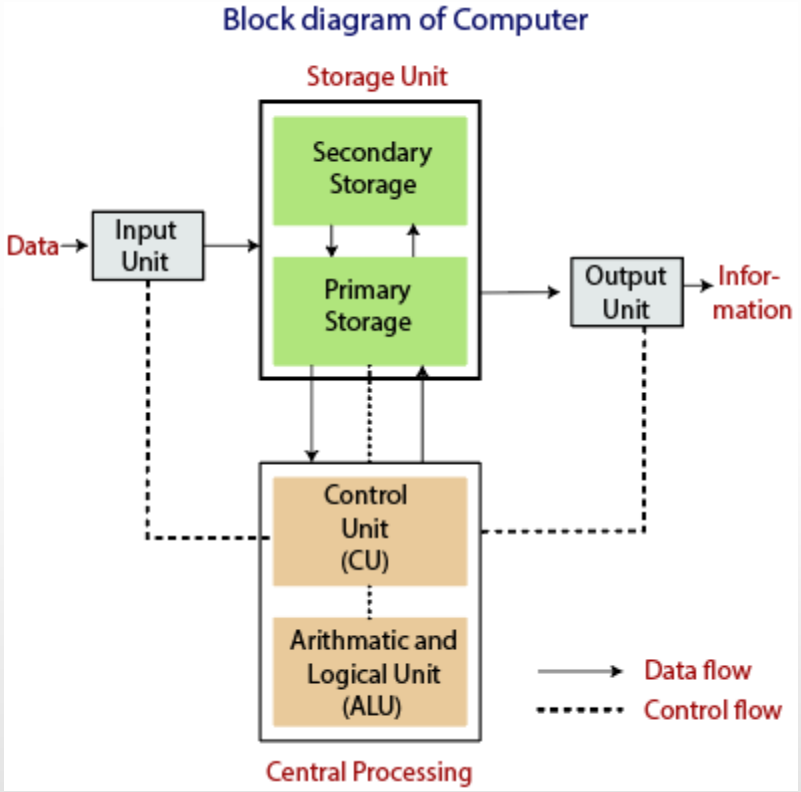
- The programs included in a systems software package are referred to as the operating system.
- They are distinguished from application programs written by the user for the purpose of solving particular problems.
- For example, a high-level language program written by a user to solve particular data-processing needs is an application program, but the compiler that translates the high-level language program to machine language is a system program.

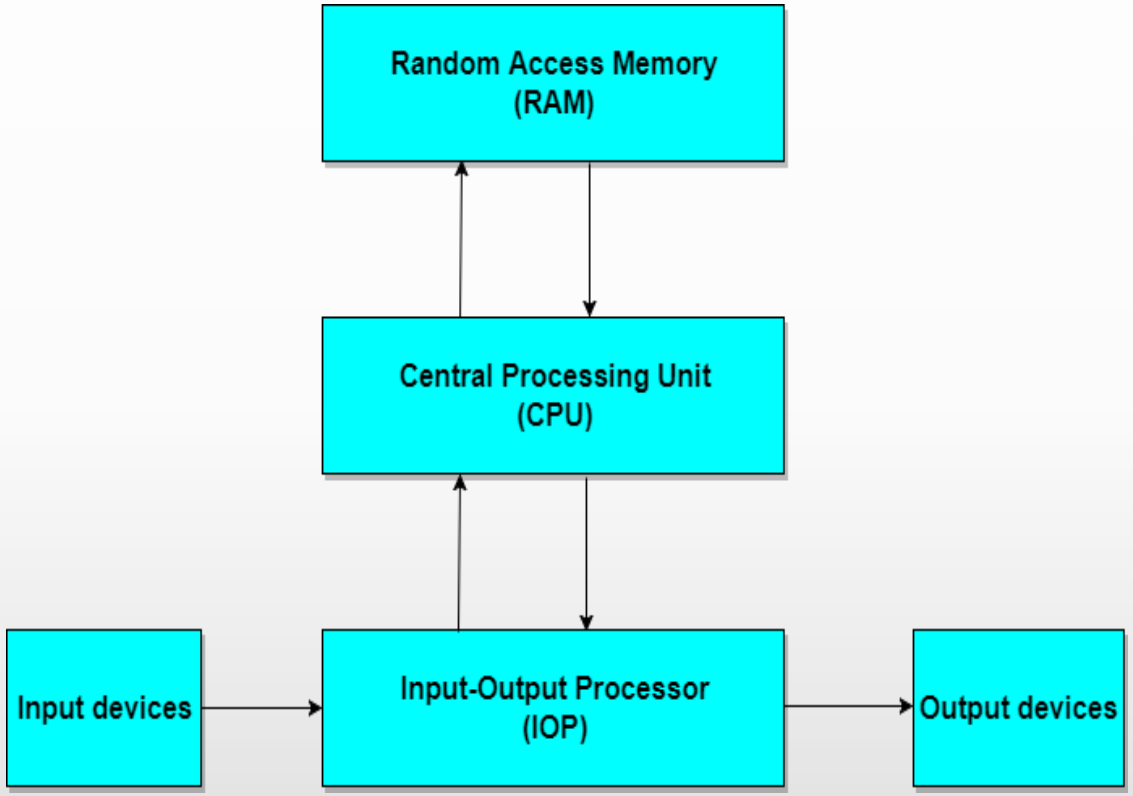
SOURCE: MORRIS MANO, THIRD EDITION

COMPUTER HARDWARE

- The hardware of the computer is usually divided into three major parts:
- The **central processing unit (CPU)** contains an arithmetic and logic unit for manipulating data, a number of registers for storing data, and control circuits for fetching and executing instructions.
- The **memory** of a computer contains storage for instructions and data. It is called a random access memory (RAM) because the CPU can access any location in memory at random and retrieve the binary information within a fixed interval of time.
- The **input and output processor (IOP)** contains electronic circuits for communicating and controlling the transfer of information between the computer and the outside world. The input and output devices connected to the computer include keyboards, printers, terminals, magnetic disk drives, and other communication devices.

SOURCE: MORRIS MANO, THIRD EDITION





Source: Google Search

COMPUTER ORGANIZATION

- Computer organization is concerned with the way the hardware components operate and the way they are connected together to form the computer system.
- The various components are assumed to be in place and the task is to investigate the organizational structure to verify that the computer parts operate as intended.

SOURCE: MORRIS MANO, THIRD EDITION

COMPUTER DESIGN

- Compute design is concerned with the hardware design of the computer.
- Once the computer specifications are formulated, it is the task of the designer to develop hardware for the system.
- Computer design is concerned with the determination of what hardware should be used and how the parts should be connected. This aspect of computer hardware is sometimes referred to as computer implementation.

SOURCE: MORRIS MANO, THIRD EDITION

COMPUTER ARCHITECTURE

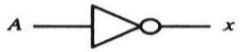
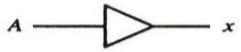
- Computer architecture is concerned with the structure and behavior of the computer as seen by the user.
- It includes the information formats, the instruction set, and techniques for addressing memory.
- The architectural design of a computer system is concerned with the specifications of the various functional modules, such as processors and memories, and structuring them together into a computer system.

SOURCE: MORRIS MANO, THIRD EDITION

SOURCE: MORRIS MANO, THIRD EDITION

LOGIC GATES

- Binary logic deals with binary variables and with operations that assume a logical meaning.
- It is used to describe, in algebraic or tabular form, the manipulation and processing of binary information.
- The manipulation of binary information is done by logic circuits called **gates** .
- Gates are blocks of hardware that produce signals of binary 1 or 0 when input logic requirements are satisfied.
- Each gate has a distinct graphic symbol and its operation can be described by means of an algebraic expression.
- The input-output relationship of the binary variables for each gate can be represented in tabular form by a **truth table**.

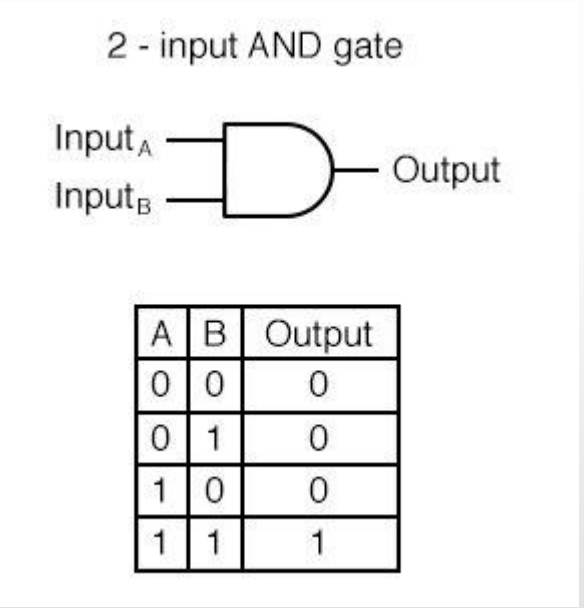
Name	Graphic symbol	Algebraic function	Truth table
AND		$x = A \cdot B$ or $x = AB$	A B | x 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$x = A + B$	A B | x 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$x = A'$	A | x 0 | 1 1 | 0
Buffer		$x = A$	A | x 0 | 0 1 | 1
NAND		$x = (AB)'$	A B | x 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$x = (A + B)'$	A B | x 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$x = A \oplus B$ or $x = A'B + AB'$	A B | x 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR or equivalence		$x = (A \oplus B)'$ or $x = A'B' + AB$	A B | x 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

SOURCE: MORRIS MANO, THIRD EDITION

SOURCE: MORRIS MANO, THIRD EDITION

AND Gate

- The AND gate produces the AND logic function: that is, the output is 1 if input A and input B are both equal to 1; otherwise, the output is 0.
- These conditions are also specified in the truth table for the AND gate. The table shows that output x is 1 only when both input A and input Bare 1.
- The algebraic operation symbol of the AND function is the same as the multiplication symbol (.) of ordinary arithmetic.
- AND gates may have more than two inputs, and by definition, the output is 1 if and only if all inputs are 1.

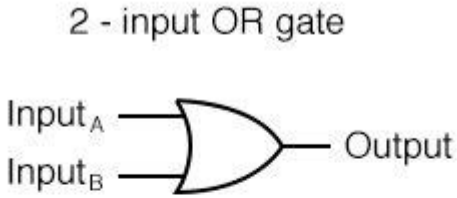


A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

SOURCE: MORRIS MANO, THIRD EDITION

OR Gate

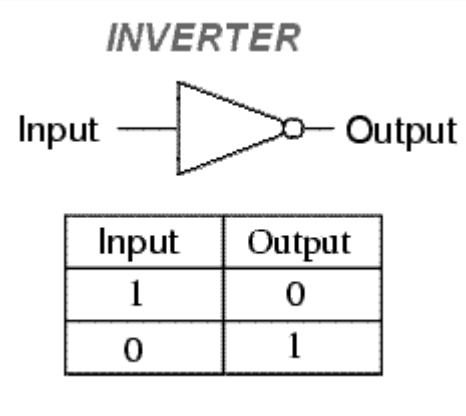
- The OR gate produces the inclusive-OR function; that is, the output is 1 if input A or input B or both inputs are 1; otherwise, the output is 0.
- The algebraic symbol of the OR function is +, similar to arithmetic addition.
- OR gates may have more than two inputs, and by definition, the output is 1 if any input is 1.



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Inverter (NOT)

- The inverter circuit inverts the logic sense of a binary signal.
- It produces the NOT, or complement function.
- The algebraic symbol used for the logic complement is either a prime or a bar over the variable symbol.

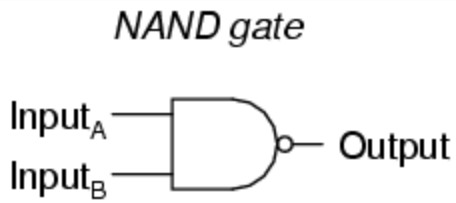


Input	Output
1	0
0	1

SOURCE: MORRIS MANO, THIRD EDITION

NAND Gate

- This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate.
- The outputs of all NAND gates are high if any of the inputs are low.
- The NAND function is the complement of the AND function, as indicated by the graphic symbol, which consists of an AND graphic symbol followed by a small circle.

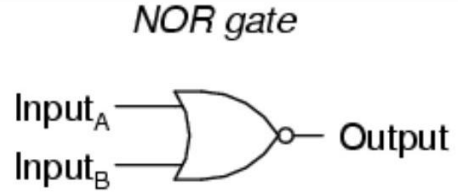


A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

SOURCE: MORRIS MANO, THIRD EDITION

NOR Gate

- This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate.
- The outputs of all NOR gates are low if any of the inputs are high.
- The NOR gate is the complement of the OR gate and uses an OR graphic symbol followed by a small circle.



A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

SOURCE: MORRIS MANO, THIRD EDITION

Exclusive-OR (XOR)

- It produces a high output if either, but not both, of its two inputs are high.
- The exclusive-OR gate has a graphic symbol similar to the OR gate except for the additional curved line on the input side.
- The output of this gate is 1 if any input is 1 but excludes the combination when both inputs are 1.

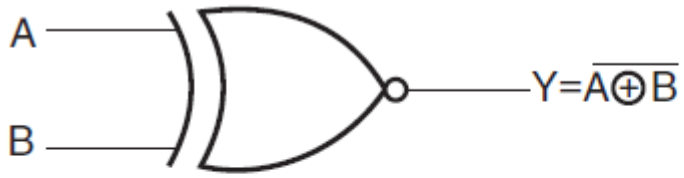
Exclusive-OR gate

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

SOURCE: MORRIS MANO, THIRD EDITION

Exclusive-NOR (XNOR)

- The 'Exclusive-NOR' gate circuit does the opposite to the XOR gate.
- It will give a low output if either, but not both, of its two inputs are high.



$$Y = (\overline{A \oplus B}) = (A.B + \overline{A}.\overline{B})$$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

SOURCE: MORRIS MANO, THIRD EDITION

BOOLEAN ALGEBRA

- Boolean algebra is an algebra that deals with binary variables and logic operations.
- This forms the algebraic expression showing the operation of the logic circuit for each input variable either True or False that results in a logic “1” output.
- The three basic logic operations are AND, OR, and complement (NOT).
- The purpose of Boolean algebra is to facilitate the analysis and design of digital circuits. It provides a convenient tool to:
  - Express in algebraic form a truth table relationship between binary variables.
  - Express in algebraic form the input-output relationship of logic diagrams.
  - Find simpler circuits for the same function.

SOURCE: MORRIS MANO, THIRD EDITION

BOOLEAN FUNCTION

- Boolean function can be expressed algebraically with binary variables, the logic operation symbols, parentheses, and equal sign.
- For a given value of the variables, the Boolean function can be either 1 or 0.

E.g.,

$$F = x + y' z$$

The function F is equal to 1 if x is 1 or if both y' and z are equal to 1; F is equal to 0 otherwise. But saying that y' = 1 is equivalent to saying that y = 0 since y' is the complement of y. Therefore, we may say that F is equal to 1 if x = 1 or if yz = 01.

SOURCE: MORRIS MANO, THIRD EDITION

TRUTH TABLE

- The relationship between a function and its binary variables can be represented in a **truth table.**
- A truth table shows how the truth or falsity of a compound statement depends on the truth or falsity of the simple statements from which it's constructed.
- It defines the function of a logic gate by providing a concise list that shows all the output states in tabular form for each possible combination of input variable that the gate could encounter.
- To represent a function in a truth table we need a list of the 2^n combinations of the **n** binary variables.

SOURCE: MORRIS MANO, THIRD EDITION

LOGIC DIAGRAM

- Boolean function can be transformed from an algebraic expression into a logic diagram composed of AND, OR, and inverter/NOT gates.
- The logic diagram consists of gates and symbols that can directly replace an expression in Boolean arithmetic.
- This is a graphical representation of a logic circuit that shows the wiring and connections of each individual logic gate, represented by a specific graphical symbol that implements the logic circuit.

SOURCE: MORRIS MANO, THIRD EDITION

BOOLEAN ALGEBRA (REVISITED)

- A Boolean function specified by a truth table can be expressed algebraically in many different ways.
- **Boolean algebra** is the branch of algebra in which the values of the variables are the truth values true and false, usually denoted 1 and 0 respectively.
- By manipulating a Boolean expression according to Boolean algebra rules, one may obtain a simpler expression that will require fewer gates.
- It is used to analyze and simplify the digital (logic) circuits.

SOURCE: MORRIS MANO, THIRD EDITION

	AND Form	OR Form
Commutative Law	$A \cdot B = B \cdot A$	$A + B = B + A$
Associate Law	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$	$(A + B) + C = A + (B + C)$
Distributive Law	$(A+B)+ C = (A+C).(B+C)$	$(A + B) \cdot C = (A \cdot C) + (B \cdot C)$
Identity Law	$A \cdot 1 = A$	$A + 0 = A$
Zero and One Law	$A \cdot 0 = 0$	$A + 1 = 1$
Inverse Law	$A \cdot A' = 0$	$A + A' = 1$
Idempotent Law	$A \cdot A = A$	$A + A = A$
Absorption Law	$A(A+B) = A$	$A + A.B = A$ $A + A'B = A+B$
DeMorgan's Law	$(A \cdot B)' = (A)' + (B)'$	$(A+ B)' = (A)' \cdot (B)'$
Double Complement Law	$\overline{\overline{x}} = x$	

SOURCE: MORRIS MANO, THIRD EDITION

Source: Google Search

NUMBER SYSTEM

- The number system is a way to represent or express numbers.
- Based on the different symbol used to represent numbers, there are various types of number systems:
  - The decimal number system
  - The binary number system
  - The octal number system and
  - The hexadecimal number system
  - Binary Coded Decimal or BCD Numbering System

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

CONVERTING FROM BINARY TO DECIMAL

- decimal = $d_0 \times 2^0 + d_1 \times 2^1 + d_2 \times 2^2 + \ldots$
- Example

binary number:	1	1	1	0	0	1
power of 2:	2^5	2^4	2^3	2^2	2^1	2^0

$$111001_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 57_{10}$$

SOURCE: INTERNET

CONVERTING FROM DECIMAL INTEGER TO BINARY

- Conversion steps:
  - Divide the number by 2.
  - Get the integer quotient for the next iteration.
  - Get the remainder for the binary digit.
  - Repeat the steps until the quotient is equal to 0.
- Example

Convert 13_{10} to binary :

Division by 2	Quotient	Remainder	Bit #
13/2	6	1	0
6/2	3	0	1
3/2	1	1	2
1/2	0	1	3

$13_{10} = 1101_2$

SOURCE: INTERNET

CONVERTING DECIMAL FRACTION TO BINARY

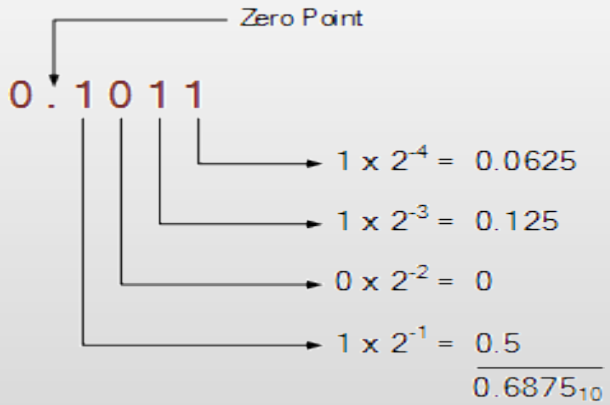
- Binary Fractions use the same weighting principle as decimal numbers except that each binary digit uses the base-2 numbering system.
- A decimal number representation of $(0.XY)_{10}$ can be converted into base of 2 and represented by $(0.\ a_1\ ,\ a_2\ ,\ a_3\ ,\ \text{etc.})_2$.
- The fraction number is multiplied by 2, the result of integer part is a_1 and fraction part multiply by 2, and then separate integer part from fraction, the integer part represents a_1 ; this process continues until the fraction becomes 0.

Example: $(0.625)_{10}$

		Integer		Fraction	Coefficient
0.625	* 2 =	1	.	25	$a_{-1} = 1$
0.25	* 2 =	0	.	5	$a_{-2} = 0$
0.5	* 2 =	1	.	0	$a_{-3} = 1$

Answer: $(0.625)_{10} = (0.a_{-1}\ a_{-2}\ a_{-3})_2 = (0.101)_2$

MSB LSB



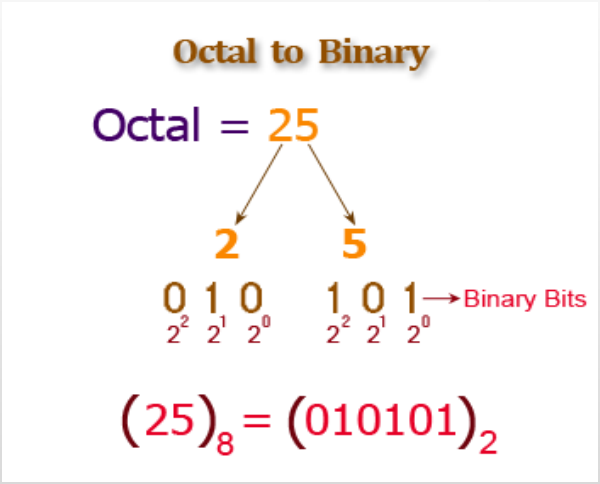
SOURCE: INTERNET

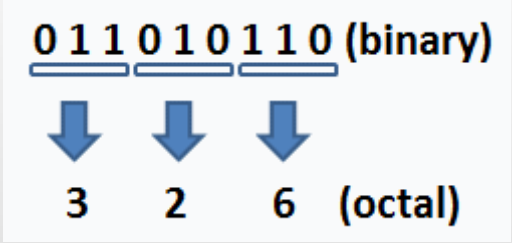
CONVERTING FROM OCTAL TO BINARY

- The octal numeral system, or oct for short, is the base-8 number system, and uses the digits 0 to 7.
- Octal numerals can be made from binary numerals by grouping consecutive binary digits into groups of three (starting from the right).

Decimal	Octal	Binary
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111

SOURCE: INTERNET

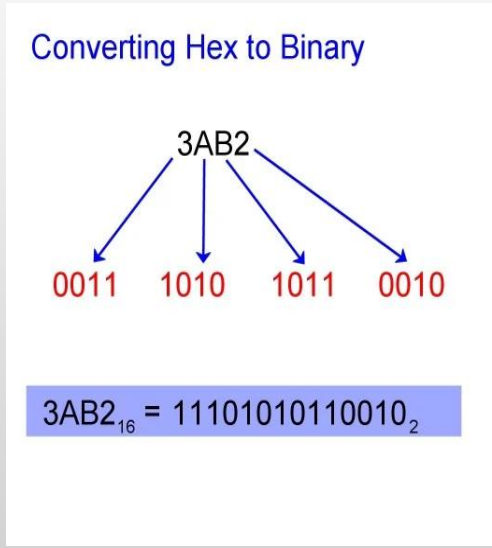


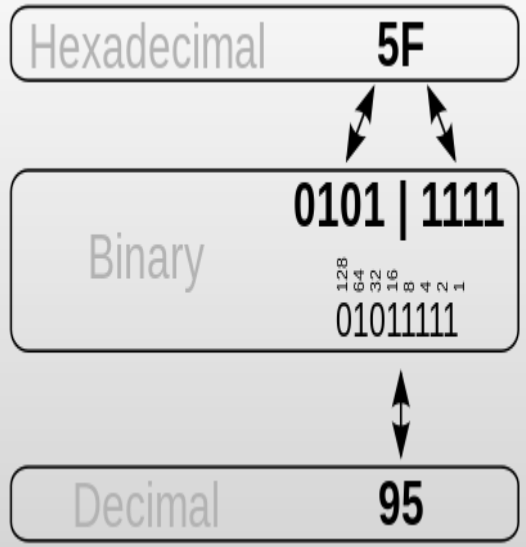


CONVERTING FROM HEX TO BINARY

- Hexadecimal (also base 16, or hex) is a positional numeral system with a radix, or base, of 16. It uses sixteen distinct symbols, most often the symbols 0–9 to represent values zero to nine, and A, B, C, D, E, F (or alternatively a, b, c, d, e, f) to represent values ten to fifteen.

Decimal	Hexadecimal	Binary
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111





SOURCE: INTERNET

BINARY CODED DECIMAL (BCD)

- **BCD** or **Binary Coded Decimal** is that number system or code which has the binary numbers or digits to represent a decimal number where each digit is represented by a fixed number of binary bits, usually between four and eight.
- A decimal number contains 10 digits (0-9). Now the equivalent binary numbers can be found out of these 10 decimal numbers. In case of BCD the binary number formed by four binary digits, will be the equivalent code for the given decimal digits. In BCD we can use the binary number from 0000-1001 only, which are the decimal equivalent from 0-9 respectively.
- The BCD_{8421} code is so called because each of the four bits is given a 'weighting' according to its column value in the binary system. The least significant bit (lsb) has the weight or value 1, the next bit, going left, the value 2. The next bit has the value 4, and the most significant bit (msb) the value 8. E.g.,

  24_{10} in 8 bit binary would be 00011000 but in BCD_{8421} is 0010 0100.

  992_{10} in 16 bit binary would be 0000001111100000 but in BCD_{8421} is 1001 1001 0010.

Decimal	BCD	
0	0 0 0 0	
1	0 0 0 1	
2	0 0 1 0	
3	0 0 1 1	
4	0 1 0 0	
5	0 1 0 1	
6	0 1 1 0	
7	0 1 1 1	
8	1 0 0 0	
9	1 0 0 1	
-	1 0 1 0	Unused
-	1 0 1 1	
-	1 1 0 0	
-	1 1 0 1	
-	1 1 1 0	
-	1 1 1 1	

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Binary-Coded Decimal vs. Binary to Decimal Conversion

Decimal Number	BCD	Binary
0	0000 0000 0000 0000	0000 0000 0000 0000
1	0000 0000 0000 0001	0000 0000 0000 0001
2	0000 0000 0000 0010	0000 0000 0000 0010
3	0000 0000 0000 0011	0000 0000 0000 0011
4	0000 0000 0000 0100	0000 0000 0000 0100
5	0000 0000 0000 0101	0000 0000 0000 0101
6	0000 0000 0000 0110	0000 0000 0000 0110
7	0000 0000 0000 0111	0000 0000 0000 0111
8	0000 0000 0000 1000	0000 0000 0000 1000
9	0000 0000 0000 1001	0000 0000 0000 1001
⋮		
9620	1001 0110 0010 0000	0010 0101 1001 0100
120	0001 0010 0000	0000 0000 0111 1000
→ 4568	0100 0101 0110 1000	0001 0001 1101 1000

REALPARS

SOURCE: HTTPS://REALPARS.COM/BCD/

Binary Base-2	Decimal Base-10	Hexa-Decimal Base-16	Octal Base-8	BCD Code
0000	0	0	0	0
0001	1	1	1	1
0010	2	2	2	2
0011	3	3	3	3
0100	4	4	4	4
0101	5	5	5	5
0110	6	6	6	6
0111	7	7	7	7
1000	8	8	10	8
1001	9	9	11	9
1010	10	A	12	---
1011	11	B	13	---
1100	12	C	14	---
1101	13	D	15	---
1110	14	E	16	---
1111	15	F	17	---

SOURCE: INTERNET

FLOATING POINT REPRESENTATION

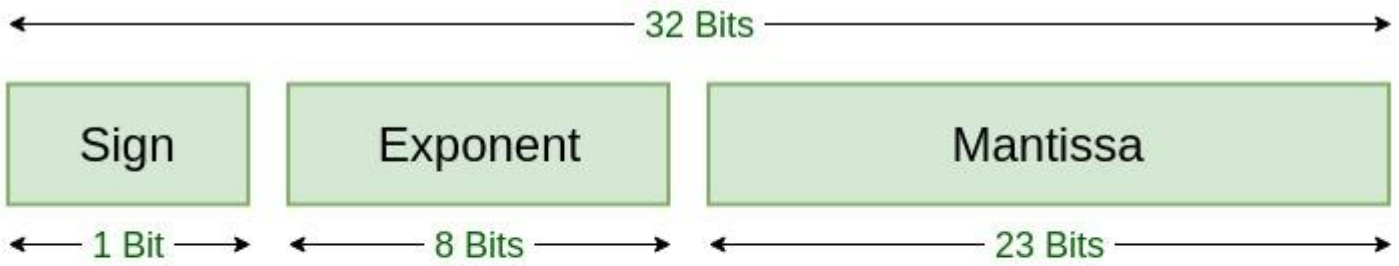
- Integers are whole numbers without fractional components. 1, 2, and 3 are integers, while 0.1, 2.2, and 3.0001 all have fractional components are called floating point numbers.
- The floating point unit performs floating point operations. Floating point numbers have a sign, a mantissa, and an exponent.
- The central processing unit (CPU) typically consists of an arithmetic logic unit (ALU), floating point unit (FLU/FPU), registers, control unit, and the cache memory.
- The arithmetic logic unit performs integer arithmetic operations such as addition, subtraction, and logic operations such as AND, OR, XOR, etc.

SOURCE: INTERNET

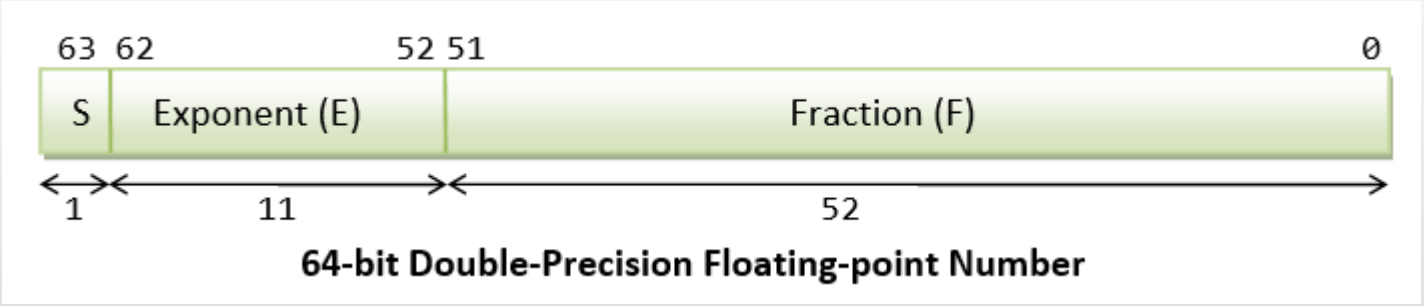
IEEE STANDARD FOR FLOATING POINT REPRESENTATION

- The Institute of Electrical and Electronics Engineers (IEEE) developed a standard to represent floating point numbers, referred to as IEEE 754.
- This standard defines a format for both single (32-bit) and double (64-bit) precision floating point numbers.
- Floating point numbers in single precision are represented by 32 bits while in order to increase the accuracy of a floating point number, IEEE 745 offers double precision represented by 64 bits.
- Decimal floating points are represented by $M \times 10^{E}$, where M is the signed mantissa (normalized mantissa) and E is the exponent (biased exponent).

SOURCE: MORRIS MANO, THIRD EDITION



Single Precision
IEEE 754 Floating-Point Standard



64-bit Double-Precision Floating-point Number

SOURCE: INTERNET

SOURCE: MORRIS MANO, THIRD EDITION

- **Biased Exponent** is the exponent + 127 $(01111111)_2$; therefore, the exponent is represented by a positive number.
- **Normalized Mantissa/mantissa** is represented by 1.M, where M is called normalized mantissa; if M = 00101, then mantissa is 1.00101.
- Example:

  Find normalized mantissa and biased exponent of $(111.0000111)_2$.

  111.0000111 can be written in the form of $1.110000111 * 2^{10}$

  Where

  M = 110000111

  Biased exponent = 10 + 01111111 = 10000001

  The representation of 111.0000111 in single precision is

1bit	8 bits	23 bits
0	10000001	11000011100000000000000

SOURCE: MORRIS MANO, THIRD EDITION

- Convert the following single precision floating point to decimal number.

  101111101 11001000000000000000000

  S = 1 means mantissa is negative.

  Biased exponent = 01111101.

  Exponent = 01111101–01111111 = 00000010.

  Normalized mantissa = 11001000000000000000000.

  Mantissa =1. 11001000000000000000000.

  Decimal number = $1.11001000000000000000000 * 2^{-10} = 0.01110011$.

SOURCE: MORRIS MANO, THIRD EDITION

- Represent 5.75 in IEEE 745 single precision.

  –15. 625 = $(1111.101)_2$

  –1111.101 = –1.11101101 * 2^{11}

  S = 1

  Normalized mantissa = 0.11101101.

  Biased exponent = 11 + 01111111 = 10000010.

IEEE745 single precision is

1 10000010 11101101000000000000000.

As examples, our 12-bit floating-point number with a binary representation of:

Sign Bit	Exponent				Mantissa						
1	0	0	0	0	1	0	0	0	0	0	1

converts to its decimal via $(-1)^1 \times 2^{1-7} \times 0.1000001 = -1 \times 2^{-6} \times 0.1000001 = -0.000001000001_2 = -(2^{-6} + 2^{-12})_{10} = -(1/64 + 1/4096) = -65/4096 = -0.015869140625$. While the floating-point number:

Sign Bit	Exponent				Mantissa						
0	0	1	1	0	1	0	1	0	1	0	1

converts to its decimal via $(-1)^0 \times 2^{6-7} \times 1.1010101 = (1 \times 2^{-1} \times 1.1010101)_2 = 0.11010101_2 = (2^{-1} + 2^{-2} + 2^{-4} + 2^{-6} + 2^{-8})_{10} = 1/2 + 1/4 + 1/16 + 1/64 + 1/256 = 213/256 = 0.83203125$.

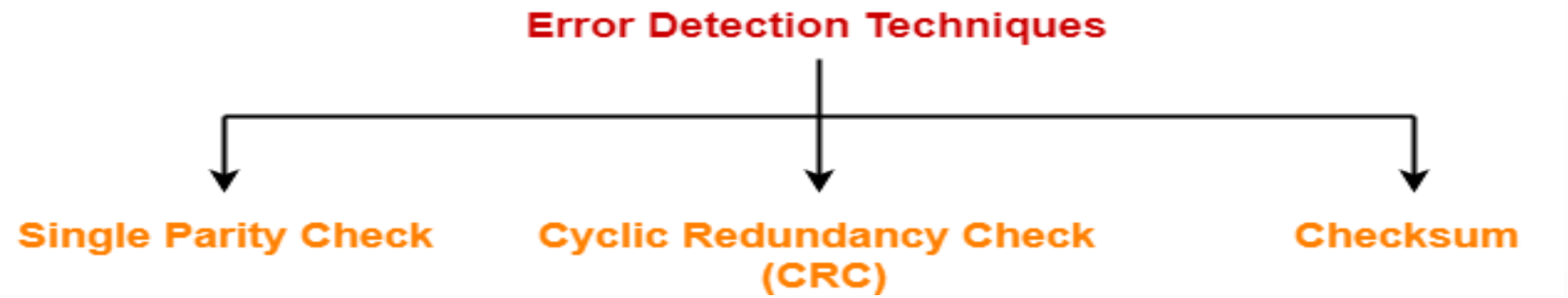
SOURCE: INTERNET

ERROR DETECTION & CORRECTION CODES

ERROR

- The sequence of bits is called as “Data stream”.
- The change in position of single bit also leads to catastrophic (major) error in data output.
- The error detection and correction techniques are used to get the exact or approximate output.
- In a data sequence, if 1 is changed to zero or 0 is changed to 1, it is called “Bit error”.
- There are generally 3 types of errors occur in data transmission from transmitter to receiver. They are
  - **Single bit errors** (The change in one bit in the whole data sequence; occurs in parallel communication system)
  - **Multiple bit errors** (If there is change in two or more bits of data sequence of transmitter to receiver; occurs in both serial type and parallel type data communication networks)
  - **Burst errors** (The change of set of bits in data sequence; calculated in from the first bit change to last bit change; occurs in serial communication and they are difficult to solve)

SOURCE: HTTPS://WWW.ELECTRONICSHUB.ORG/



PARITY (Vertical Redundancy Check (VRC))

A parity bit is used for error detection of information, since a bit or bits may be changed during the transmission of information from source to destination, a parity bit is an extra bit appended to the information. It represents whether the number of ones or zeroes is either even or odd in the original transmission and can alert the destination to a loss of information.

- Even Parity

  The extra bit (0 or 1) is chosen such that the number of ones becomes even.

- Odd Parity

  The extra bit (0 or 1) is chosen such that the number of ones becomes odd.

SOURCE: HTTPS://WWW.ELECTRONICSHUB.ORG/

3 bit data			Message with even parity		Message with odd parity	
A	B	C	Message	Parity	Message	Parity
0	0	0	000	0	000	1
0	0	1	001	1	001	0
0	1	0	010	1	010	0
0	1	1	011	0	011	1
1	0	0	100	1	100	0
1	0	1	101	0	101	1
1	1	0	110	0	110	1
1	1	1	111	1	111	0

Original Data	Even Parity	Odd Parity
00000000	0	1
01011011	1	0
01010101	0	1
11111111	0	1
10000000	1	0
01001001	1	0

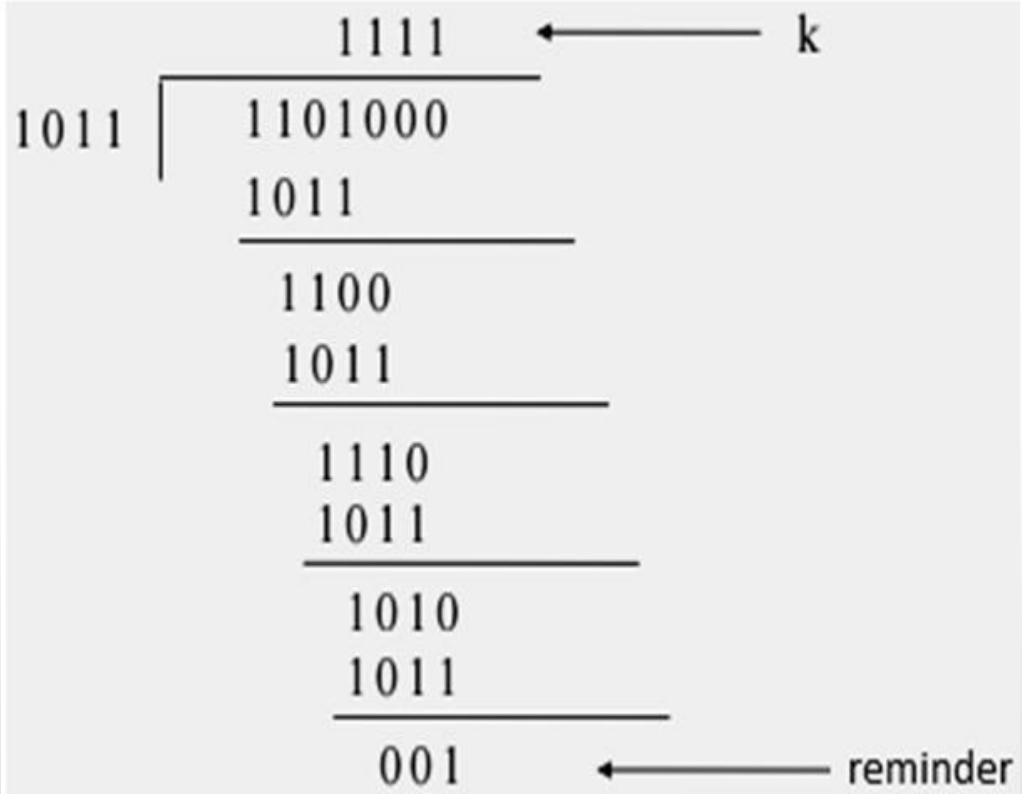
SOURCE: HTTPS://WWW.ELECTRONICSHUB.ORG/

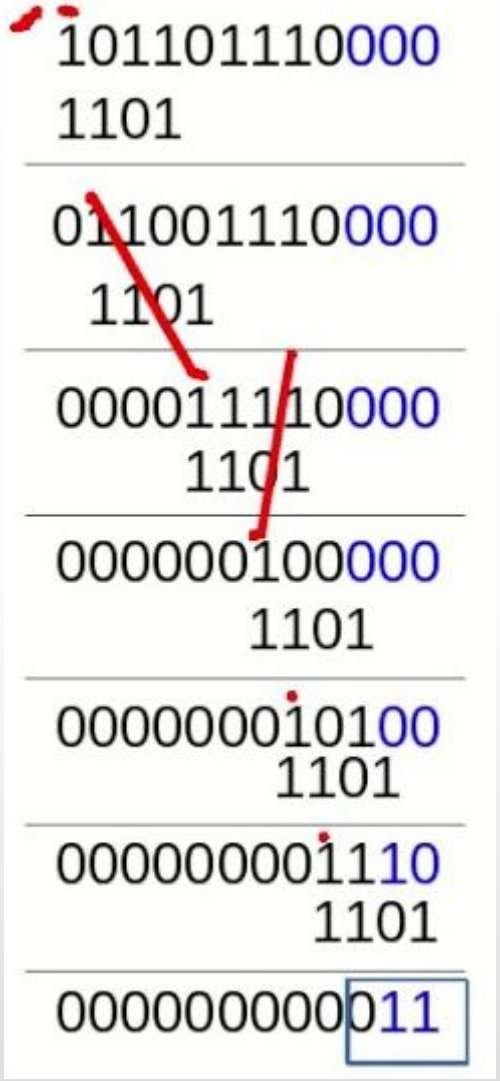
SOURCE: MORRIS MANO, THIRD EDITION

Cyclic Redundancy Check (CRC)

- CRC is commonly used to detect accidental changes to data transmitted via telecommunications networks and storage devices.
- A cyclic code is a linear (n, k) block code with the property that every cyclic shift of a codeword results in another code word. Here k indicates the length of the message at transmitter (the number of information bits). n is the total length of the message after adding check bits. (actual data and the check bits). n, k is the number of check bits. The codes used for cyclic redundancy check there by error detection are known as CRC codes (Cyclic redundancy check codes).Cyclic redundancy-check codes are shortened cyclic codes.
- CRC involves binary division of the data bits being sent by a predetermined divisor agreed upon by the communicating system. The divisor is generated using polynomials. So, CRC is also called polynomial code checksum.

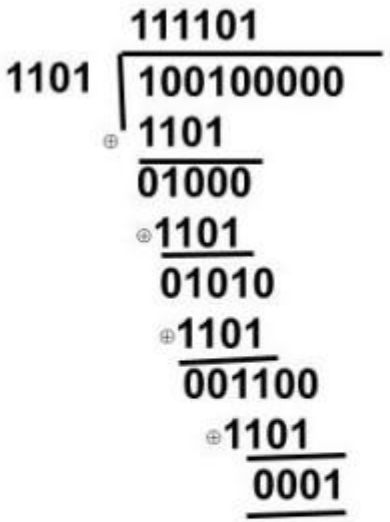
SOURCE: MORRIS MANO, THIRD EDITION





SOURCE: INTERNET

Senders Side

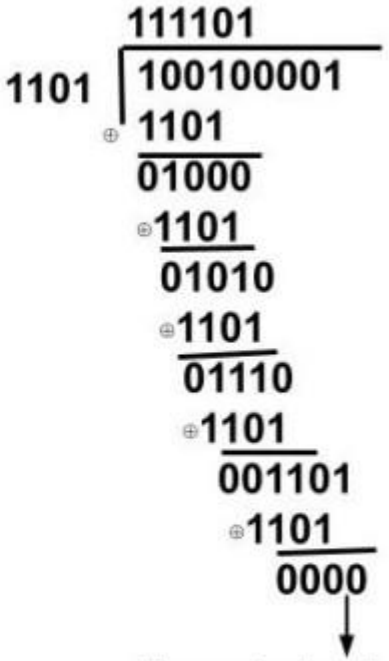


CRC bits → 001

Transmitted bits = Original Message +CRC bits
= 100100000+001
= 100100001

⊕ represents bitwise XOR

Receivers Side

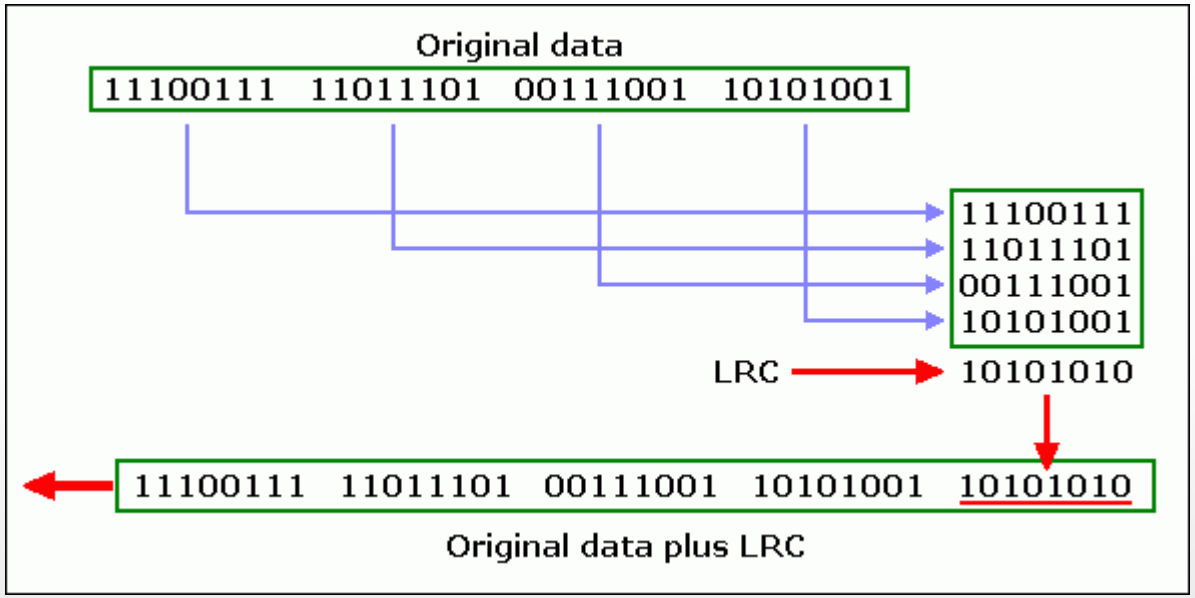


Remainder is zero, So data is accepted

SOURCE: MORRIS MANO, THIRD EDITION

Longitudinal Redundancy Check (LRC)

- In longitudinal redundancy method, a BLOCK of bits are arranged in a table format (in rows and columns) and we will calculate the parity bit for each column separately.
- The set of these parity bits are also sent along with our original data bits.
- Longitudinal redundancy check is a bit by bit parity computation, as we calculate the parity of each column individually.
- LRC increases the likelihood of detecting burst error.
- However, if two bits in one data unit are damaged and two bits in exactly the same positions in another data unit are also damaged, the LRC checker will not detect an error.



1010*0*/1 0011*0*/1 11011101 11100111
10101010 (LRC)

Calculate the LRC for Data Received

1010*0*/1
0011*0*/1
11011101
11100111

→ LRC Calculated by Receiver 10101010

→ Compare with LRC Received 10101010

SOURCE: MORRIS MANO, THIRD EDITION

Checksum

- A checksum number is appended to the packet sequence so that the sum of data plus checksum is zero.
- When received, the packet sequence may be added, along with the checksum, by a local microprocessor. If the sum is nonzero, an error has occurred.
- The checksum method includes parity bits, check digits and longitudinal redundancy check (LRC).

SOURCE: MORRIS MANO, THIRD EDITION

example: add two 16-bit integers

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

If k = 4, and n = 8 then

k=4, n=8
10110011
10101011
01011110
1
01011111
01011010
10111001
11010101
10001110
1
Sum : 10001111
Checksum 01110000

At sender side

10110011
10101011
01011110
1
01011111
01011010
10111001
11010101
10001110
1
10001111
01110000
Sum: 11111111
Complement = 00000000
Conclusion = Accept data

At receiver side

SOURCE: MORRIS MANO, THIRD EDITION

Hamming Codes

- Hamming code is a set of error-correction codes that can be used to **detect and correct the errors** that can occur when the data is moved or stored from the sender to the receiver.
- Redundant bits: Redundant bits are extra binary bits that are generated and added to the information-carrying bits of data transfer to ensure that no bits were lost during the data transfer.
  The number of redundant bits can be calculated using the following formula:

  $2^r \geq m + r + 1$ where, r = redundant bit, m = data bit
- Parity Bits: A parity bit is a bit appended to a data of binary bits to ensure that the total number of 1's in the data is even or odd.

SOURCE: INTERNET

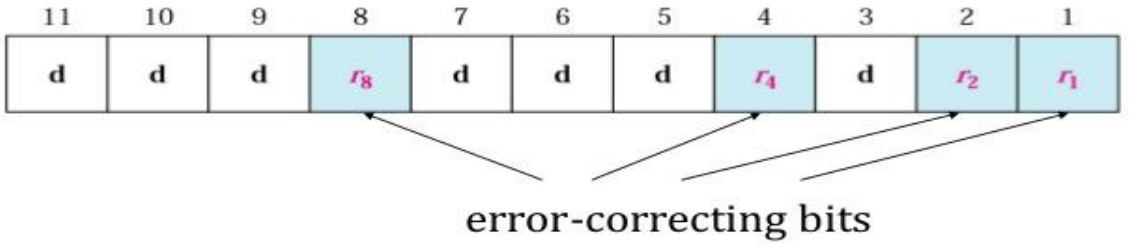
The key to the Hamming Code is the use of extra parity bits to allow the identification of a single error. Create the code word as follows:

1. Mark all bit positions that are powers of two as parity bits. (positions 1, 2, 4, 8, 16, 32, 64, etc.)
2. All other bit positions are for the data to be encoded. (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, etc.)
3. Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips.

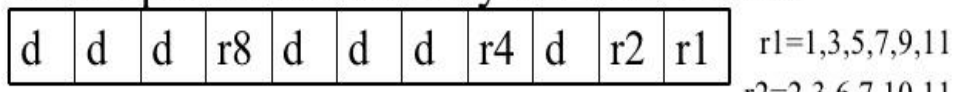
   Position 1: check 1 bit, skip 1 bit, check 1 bit, skip 1 bit, etc. (1,3,5,7,9,11,13,15,...)
   Position 2: check 2 bits, skip 2 bits, check 2 bits, skip 2 bits, etc. (2,3,6,7,10,11,14,15,...)
   Position 4: check 4 bits, skip 4 bits, check 4 bits, skip 4 bits, etc. (4,5,6,7,12,13,14,15,20,21,22,23,...)
   Position 8: check 8 bits, skip 8 bits, check 8 bits, skip 8 bits, etc. (8-15,24-31,40-47,...)
   Position 16: check 16 bits, skip 16 bits, check 16 bits, skip 16 bits, etc. (16-31,48-63,80-95,...)
   Position 32: check 32 bits, skip 32 bits, check 32 bits, skip 32 bits, etc. (32-63,96-127,160-191,...)
   etc.

4. Set a parity bit to 1 if the total number of ones in the positions it checks is odd. Set a parity bit to 0 if the total number of ones in the positions it checks is even.

SOURCE: INTERNET



Example of redundancy bit calculation



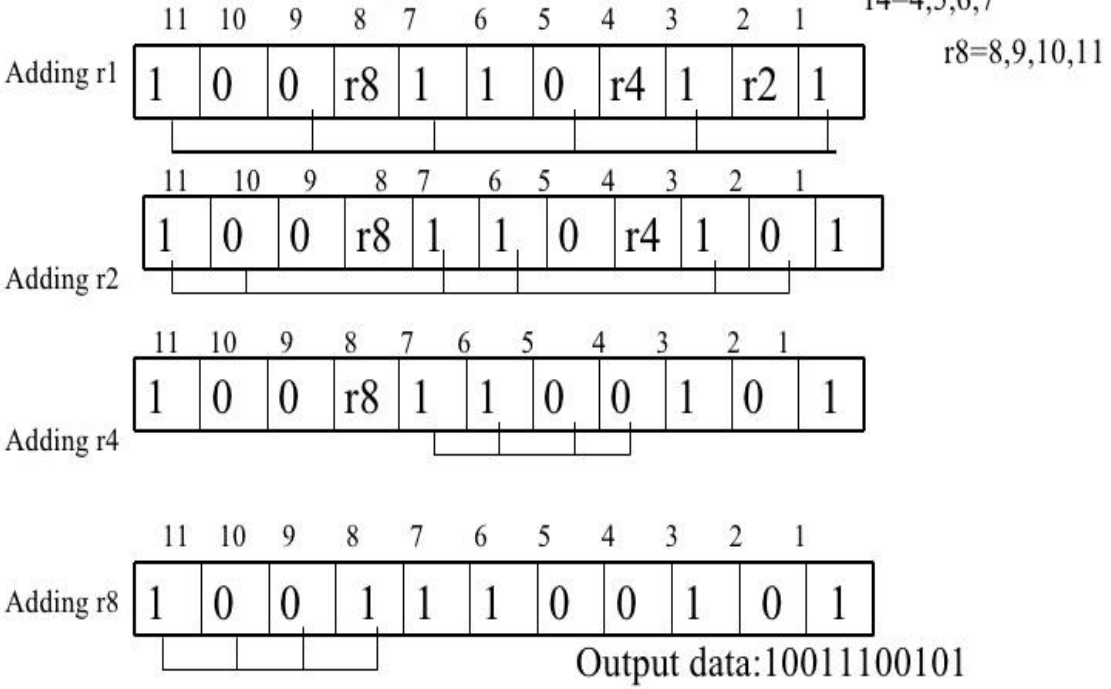
r1=1,3,5,7,9,11

r2=2,3,6,7,10,11

r4=4,5,6,7

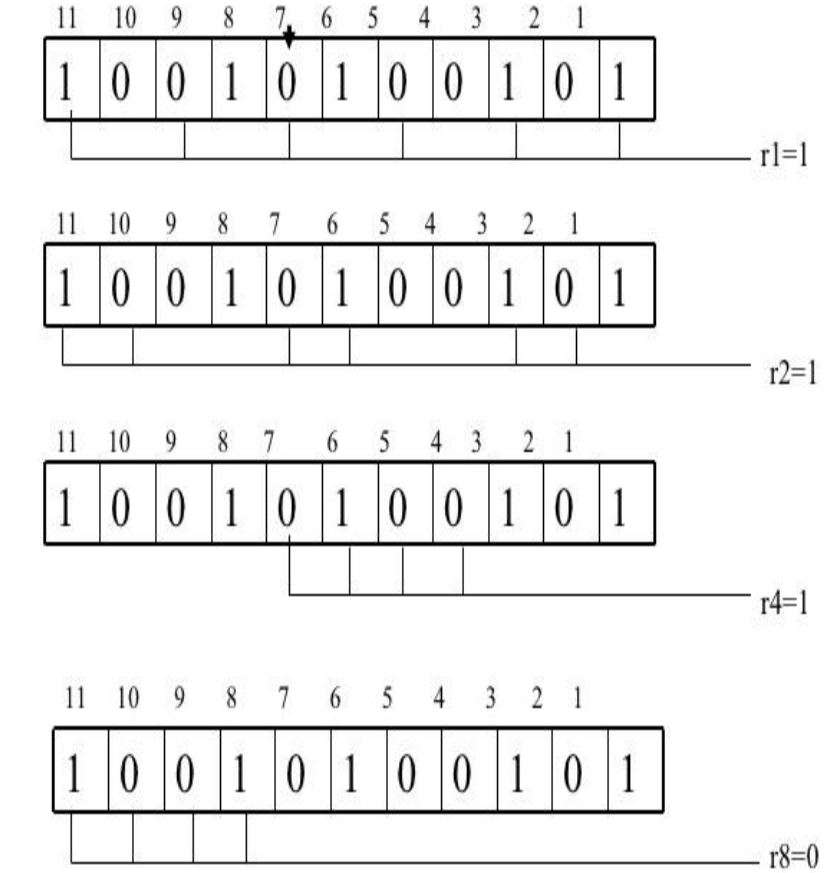
r8=8,9,10,11

Input Data 1001101



example Error detecting using hamming code

error



If no.1,s is even 0

If no.1,s is odd 1

8 4 2 1

0 1 1 1

7

It mean the 7 bit is corrupted

SOURCE: INTERNET

DUALITY PRINCIPLE

- Dual:

The dual of a Boolean expression is the expression one obtains by interchanging addition and multiplication and interchanging 0's and 1's. The dual of the function F is denoted F^d.

- Duality Principle:

This principle states that any algebraic equality derived from these axioms will still be valid whenever the OR and AND operators, and identity elements 0 and 1, have been interchanged. i.e. changing every OR into AND and vice versa, and every 0 into 1 and vice versa, i.e.,

If F and G are Boolean functions such that F = G, then $F^d = G^d$.

Example:

The dual of xy' + x'z = (x + y') · (x' + z).

SOURCE: MORRIS MANO, THIRD EDITION

SOURCE: WILLIAM STALLINGS, NINTH EDITION

CHAPTER-2
A BRIEF HISTORY OF COMPUTERS

- The chapter gives a overview of the evolution of computer technology from early digital computers to the latest microprocessors.
- The generation of computers is classified as:
  - The First Generation: Vacuum Tubes
  - The Second Generation: Transistors
  - The Third Generation: Integrated Circuits
  - Later Generations
- Each new generation is characterized by greater processing performance, larger memory capacity, and smaller size than the previous one.

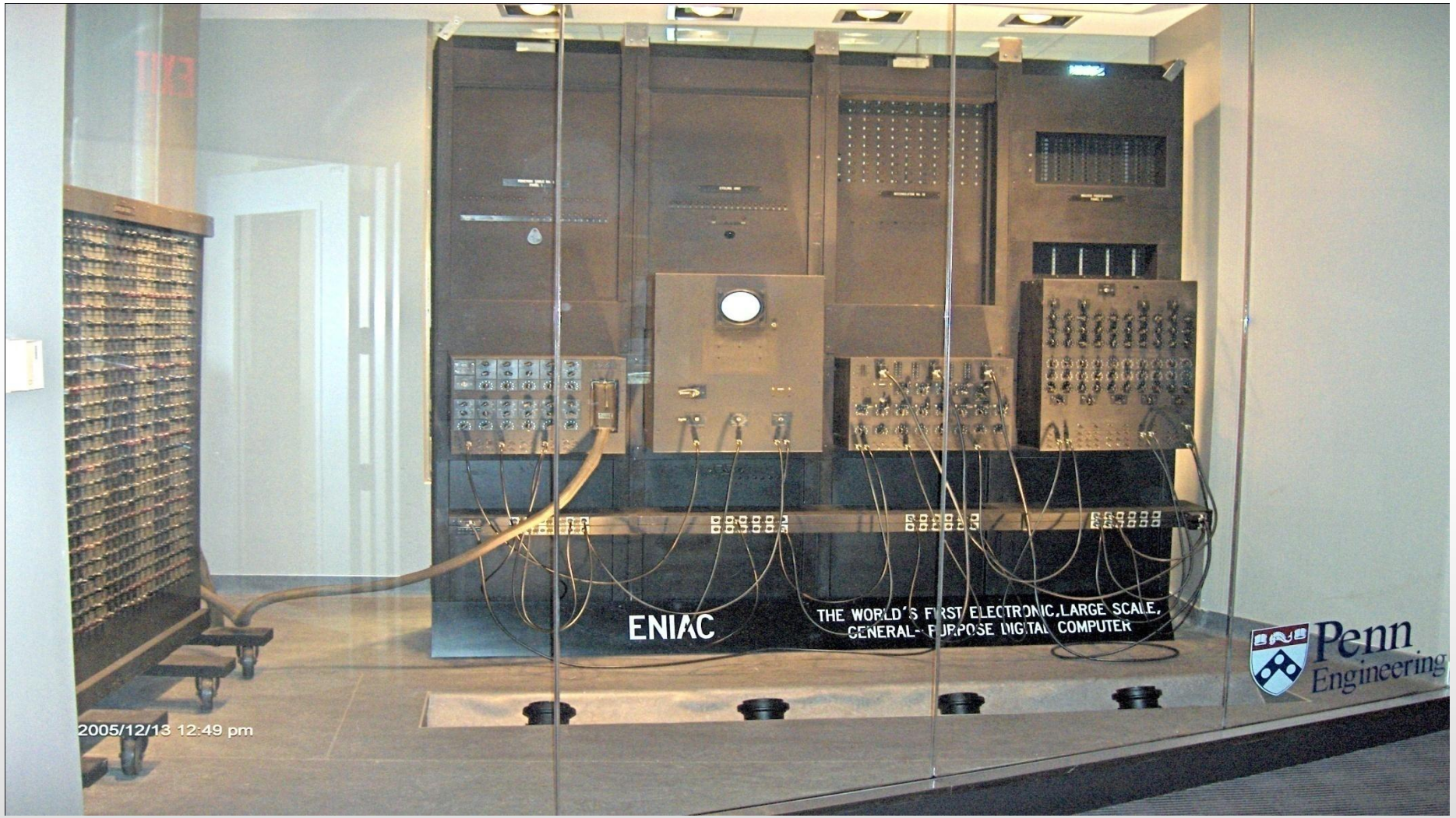
SOURCE: WILLIAM STALLINGS, NINTH EDITION

Generation	Approximate Dates	Technology	Typical Speed (operations per second)
1	1946–1957	Vacuum tube	40,000
2	1958–1964	Transistor	200,000
3	1965–1971	Small- and medium-scale integration	1,000,000
4	1972–1977	Large-scale integration	10,000,000
5	1978–1991	Very-large-scale integration	100,000,000
6	1991–	Ultra-large-scale integration	1,000,000,000

THE FIRST GENERATION: VACUUM TUBES

- ENIAC

The ENIAC (Electronic Numerical Integrator And Computer), designed and constructed at the University of Pennsylvania, was the world's first general purpose electronic digital computer. The ENIAC was a decimal rather than a binary machine. That is, numbers were represented in decimal form, and arithmetic was performed in the decimal system. Its memory consisted of 20 accumulators, each capable of holding a 10-digit decimal number. A ring of 10 vacuum tubes represented each digit. At any time, only one vacuum tube was in the ON state, representing one of the 10 digits. The major drawback of the ENIAC was that it had to be programmed manually by setting switches and plugging and unplugging cables.

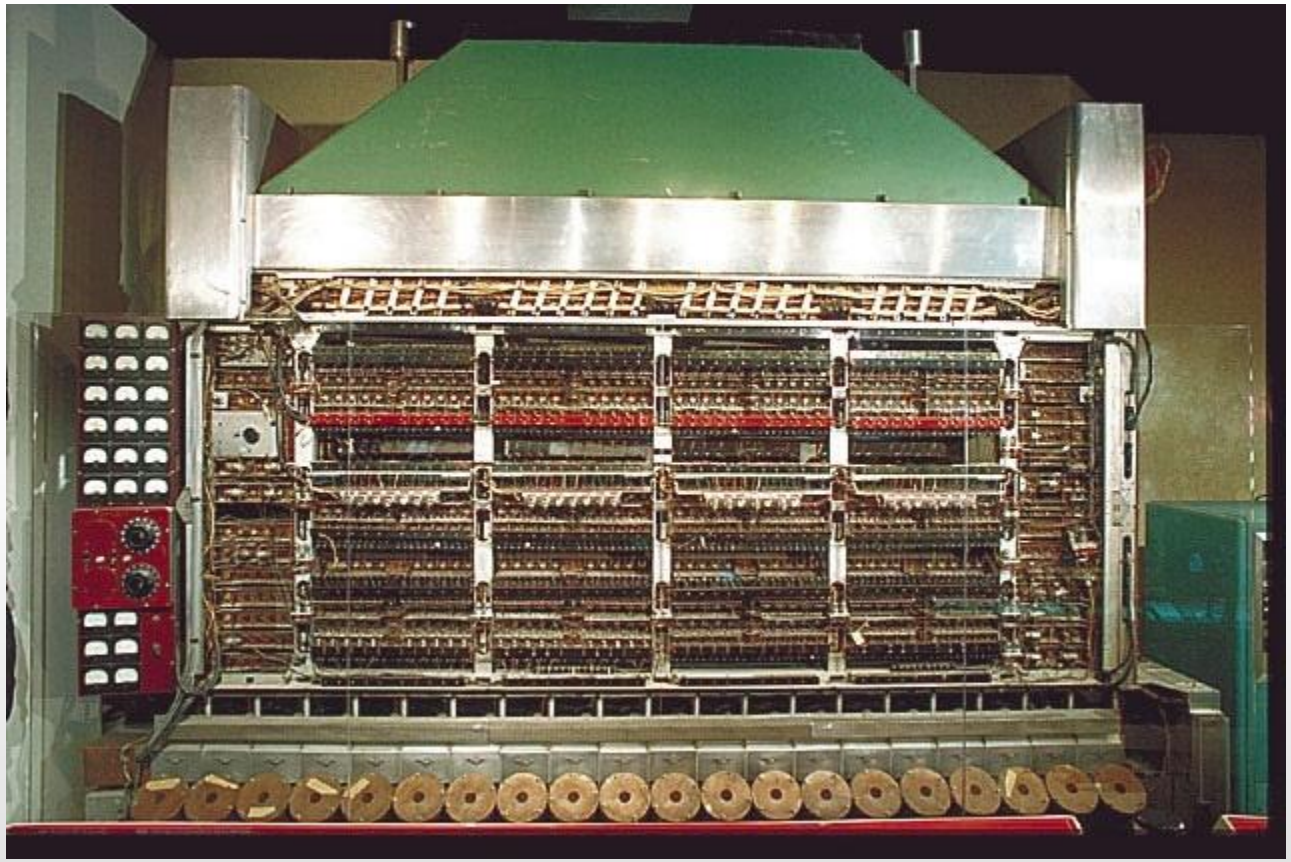


SOURCE: INTERNET

- THE VON NEUMANN MACHINE

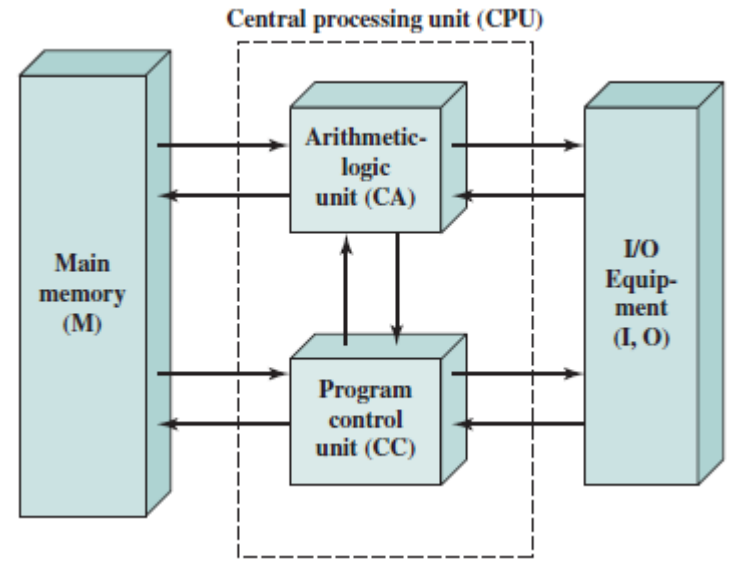
The task of entering and altering programs for the ENIAC was extremely tedious. But suppose a program could be represented in a form suitable for storing in memory alongside the data. Then, a computer could get its instructions by reading them from memory, and a program could be set or altered by setting the values of a portion of memory. This idea, known as the stored-program concept, is usually attributed to the ENIAC designers, most notably the mathematician John von Neumann, who was a consultant on the ENIAC project.

In 1946, von Neumann and his colleagues began the design of a new stored program computer, referred to as the IAS computer, at the Princeton Institute for Advanced Studies.



SOURCE: INTERNET

SOURCE: WILLIAM STALLINGS, NINTH EDITION



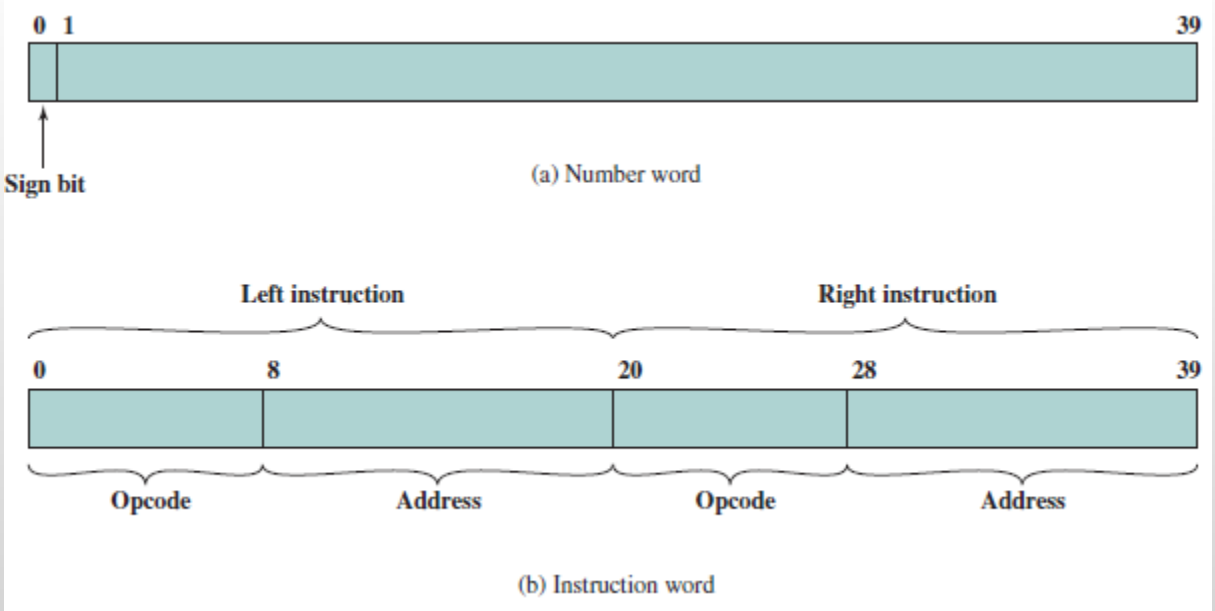
Structure of the IAS Computer

- A **main memory**, which stores both data and instructions.
- An **arithmetic and logic unit (ALU)** capable of operating on binary data.
- A **control unit**, which interprets the instructions in memory and causes them to be executed.
- **Input/output (I/O)** equipment operated by the control unit.

With rare exceptions, all of today's computers have this same general structure and function and are thus referred to as **von Neumann machines**.

SOURCE: WILLIAM STALLINGS, NINTH EDITION

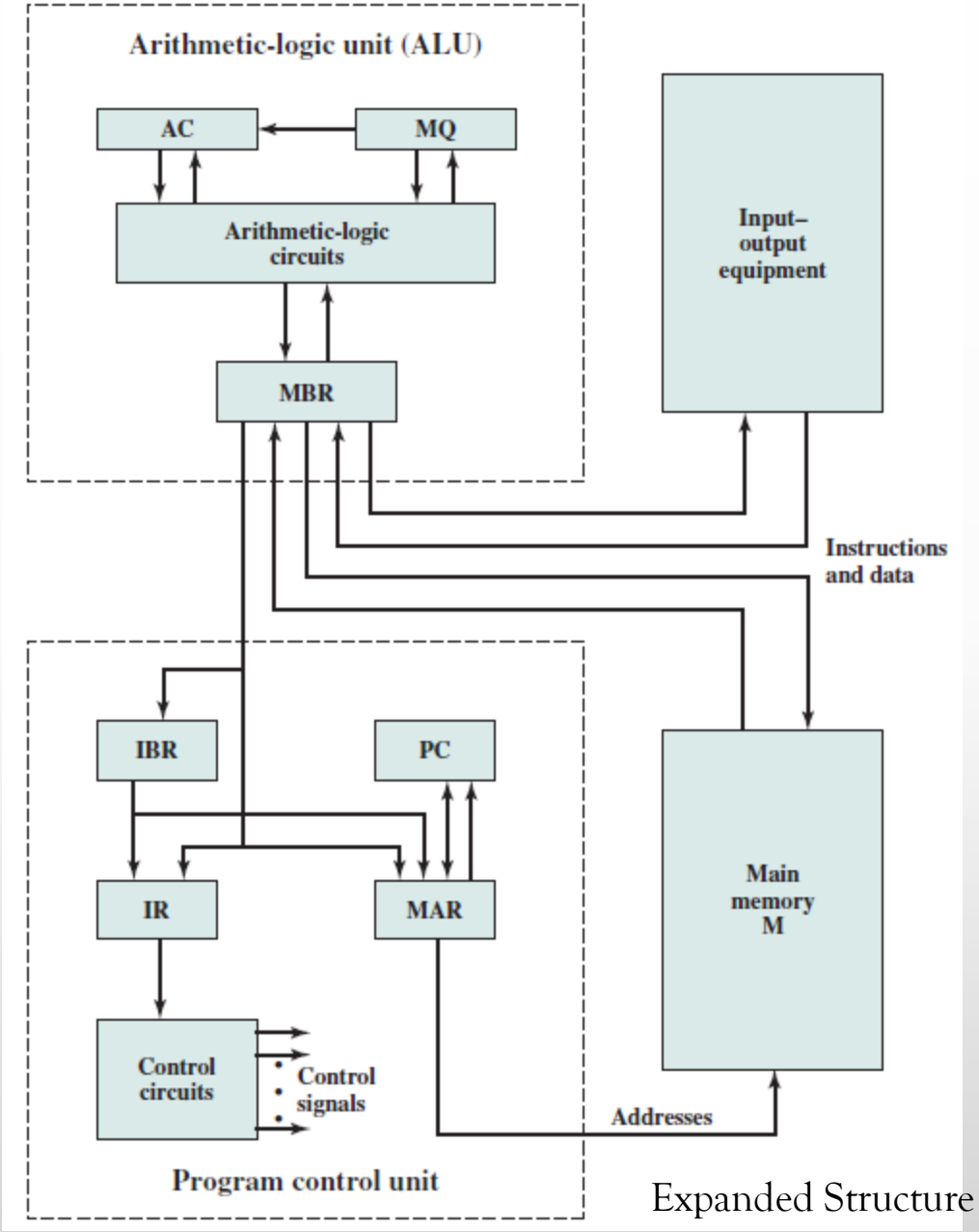
- The **memory** of the IAS consists of 1000 storage locations, called words, of 40 binary digits (bits) each. Both data and instructions are stored there. Numbers are represented in binary form, and each instruction is a binary code. **Each number is represented by a sign bit and a 39-bit value.** A word may also contain **two 20-bit instructions**, with each instruction consisting of an **8-bit operation code (opcode)** specifying the operation to be performed and a **12-bit address** designating one of the words in memory (numbered from 0 to 999).



SOURCE: WILLIAM STALLINGS, NINTH EDITION

- The **control unit** operates the IAS by fetching instructions from memory and executing them one at a time.
- The control unit and the ALU contain storage locations, called registers, defined as follows:
  - **Memory buffer register (MBR):** Contains a word to be stored in memory or sent o the I/O unit, or is used to receive a word from memory or from the I/O unit.
  - **Memory address register (MAR):** Specifies the address in memory of the word to be written from or read into the MBR.
  - **Instruction register (IR):** Contains the 8-bit opcode instruction being executed.
  - **Instruction buffer register (IBR):** Employed to hold temporarily the righthand instruction from a word in memory.
  - **Program counter (PC):** Contains the address of the next instruction pair to be fetched from memory.
  - **Accumulator (AC) and multiplier quotient (MQ):** Employed to hold temporarily operands and results of ALU operations. For example, the result of multiplying two 40-bit numbers is an 80-bit number; the most significant 40 bits are stored in the AC and the least significant in the MQ.

SOURCE: WILLIAM STALLINGS, NINTH EDITION



Expanded Structure of IAS Computer

- The IAS operates by repetitively performing an **instruction cycle**. Each instruction cycle consists of two sub cycles.
  - During the **fetch cycle**, the opcode of the next instruction is loaded into the IR and the address portion is loaded into the MAR. This instruction may be taken from the IBR, or it can be obtained from memory by loading a word into the MBR, and then down to the IBR, IR, and MAR.
  - Once the opcode is in the IR, the **execute cycle** is performed. Control circuitry interprets the opcode and executes the instruction by sending out the appropriate control signals to cause data to be moved or an operation to be performed by the ALU.

SOURCE: WILLIAM STALLINGS, NINTH EDITION

- The IAS computer had a total of **21 instructions** which can be grouped as follows:
  - **Data transfer:** Move data between memory and ALU registers or between two ALU registers.
  - **Unconditional branch:** Normally, the control unit executes instructions in sequence from memory. This sequence can be changed by a branch instruction, which facilitates repetitive operations.
  - **Conditional branch:** The branch can be made dependent on a condition, thus allowing decision points.
  - **Arithmetic:** Operations performed by the ALU.
  - **Address modify:** Permits addresses to be computed in the ALU and then inserted into instructions stored in memory. This allows a program considerable addressing flexibility.

SOURCE: WILLIAM STALLINGS, NINTH EDITION

Instruction Type	Opcode	Symbolic Representation	Description
Data transfer	00001010	LOAD MQ	Transfer contents of register MQ to the accumulator AC
	00001001	LOAD MQ,M(X)	Transfer contents of memory location X to MQ
	00100001	STOR M(X)	Transfer contents of accumulator to memory location X
	00000001	LOAD M(X)	Transfer M(X) to the accumulator
	00000010	LOAD – M(X)	Transfer –M(X) to the accumulator
	00000011	LOAD |M(X)|	Transfer absolute value of M(X) to the accumulator
	00000100	LOAD – |M(X)|	Transfer –|M(X)| to the accumulator
Unconditional branch	00001101	JUMP M(X,0:19)	Take next instruction from left half of M(X)
	00001110	JUMP M(X,20:39)	Take next instruction from right half of M(X)
Conditional branch	00001111	JUMP + M(X,0:19)	If number in the accumulator is nonnegative, take next instruction from left half of M(X)
	00010000	JUMP + M(X,20:39)	If number in the accumulator is nonnegative, take next instruction from right half of M(X)
Arithmetic	00000101	ADD M(X)	Add M(X) to AC; put the result in AC
	00000111	ADD |M(X)|	Add |M(X)| to AC; put the result in AC
	00000110	SUB M(X)	Subtract M(X) from AC; put the result in AC
	00001000	SUB |M(X)|	Subtract |M(X)| from AC; put the remainder in AC
	00001011	MUL M(X)	Multiply M(X) by MQ; put most significant bits of result in AC, put least significant bits in MQ
	00001100	DIV M(X)	Divide AC by M(X); put the quotient in MQ and the remainder in AC
	00010100	LSH	Multiply accumulator by 2; that is, shift left one bit position
	00010101	RSH	Divide accumulator by 2; that is, shift right one position
Address modify	00010010	STOR M(X,8:19)	Replace left address field at M(X) by 12 rightmost bits of AC
	00010011	STOR M(X,28:39)	Replace right address field at M(X) by 12 rightmost bits of AC

The IAS Instruction Set

- COMMERCIAL COMPUTERS

The UNIVAC I was the first successful commercial computer. It was intended for both scientific and commercial applications.

The UNIVAC II, which had greater memory capacity and higher performance than the UNIVAC I, was delivered in the late 1950s and illustrates several trends that have remained characteristic of the computer industry.

First, advances in technology allow companies to continue to build larger, more powerful computers. Second, each company tries to make its new machines backward compatible with the older machines.

SOURCE: MORRIS MANO, THIRD EDITION

The UNIVAC division also began development of the 1100 series of computers, which was to be its major source of revenue. This series illustrates a distinction that existed at one time. The first model, the UNIVAC 1103, and its successors for many years were primarily intended for scientific applications, involving long and complex calculations.

IBM, then the major manufacturer of punched-card processing equipment, delivered its first electronic stored-program computer, the 701, in 1953. The 701 was intended primarily for scientific applications. In 1955, IBM introduced the companion 702 product, which had a number of hardware features that suited it to business applications. These were the first of a long series of 700/7000 computers that established IBM as the overwhelmingly dominant computer manufacturer.

UNIVAC I

UNIVAC II

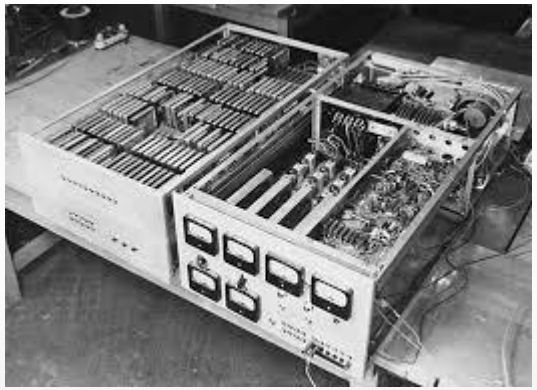
UNIVAC 1103

IBM 701

SOURCE: INTERNET

THE SECOND GENERATION: TRANSISTORS

- The first major change in the electronic computer came with the replacement of the vacuum tube by the transistor. The transistor is smaller, cheaper, and dissipates less heat than a vacuum tube but can be used in the same way as a vacuum tube to construct computers.
- Unlike the vacuum tube, which requires wires, metal plates, a glass capsule, and a vacuum, **the transistor is a solid-state device**, made from silicon.
- *The use of the transistor defines the second generation of computers.* It has become widely accepted to classify computers into generations based on the fundamental hardware technology employed.



SOURCE: INTERNET

- The second generation saw the introduction of more complex arithmetic and logic units and control units, the use of high-level programming languages, and the provision of *system software* with the computer.
- In broad terms, system software provided the ability to load programs, move data to peripherals, and libraries to perform common computations, similar to what modern OSes like Windows and Linux do.
- The second generation is noteworthy also for the appearance of the Digital Equipment Corporation (DEC), who delivered its first computer, the (Programmed Data Processor) PDP-1. This computer and this company began the minicomputer phenomenon that became prominent in the third generation.
- From the introduction of the 700 series in 1952 to the introduction of the last member of the 7000 series in 1964, IBM 7094 which underwent an evolution that is typical of computer products. Successive members of the product line show increased performance, increased capacity, and/or lower cost.

SOURCE: INTERNET

DEC PDP-1

IBM 7094

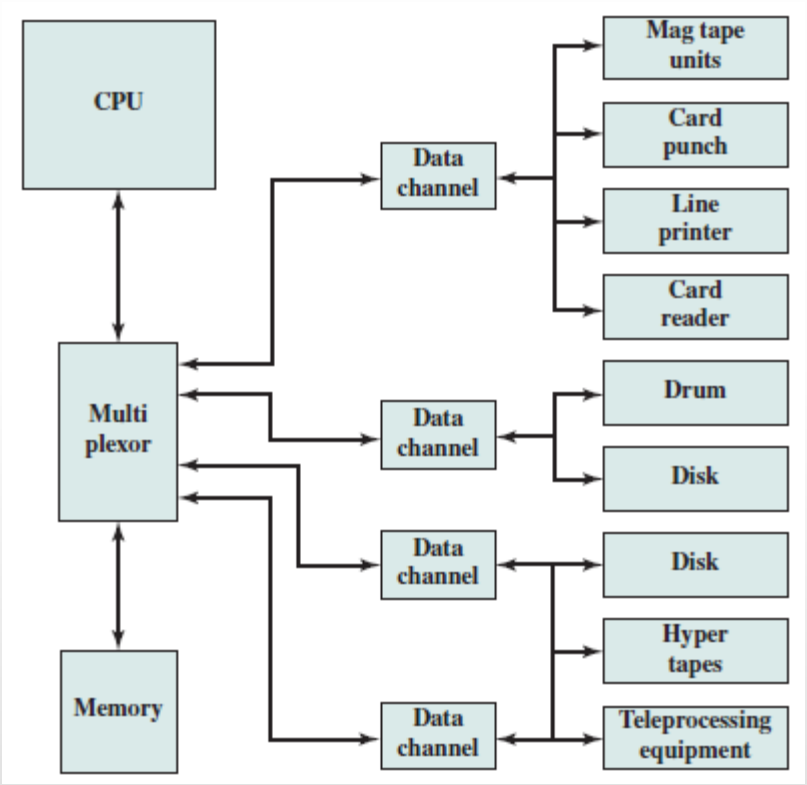
SOURCE: WILLIAM STALLINGS, NINTH EDITION

Model Number	First Delivery	CPU Tech-nology	Memory Tech-nology	Cycle Time (μs)	Memory Size (K)	Number of Opcodes	Number of Index Registers	Hardwired Floating-Point	I/O Overlap (Channels)	Instruc-tion Fetch Overlap	Speed (relative to 701)
701	1952	Vacuum tubes	Electrostatic tubes	30	2–4	24	0	no	no	no	1
704	1955	Vacuum tubes	Core	12	4–32	80	3	yes	no	no	2.5
709	1958	Vacuum tubes	Core	12	32	140	3	yes	yes	no	4
7090	1960	Transistor	Core	2.18	32	169	3	yes	yes	no	25
7094 I	1962	Transistor	Core	2	32	185	7	yes (double precision)	yes	yes	30
7094 II	1964	Transistor	Core	1.4	32	185	7	yes (double precision)	yes	yes	50

Example members of the IBM 700/7000 Series

SOURCE: WILLIAM STALLINGS, NINTH EDITION

- The most important of these is the use of **data channels**. A data channel is an independent I/O module with its own processor and instruction set. In a computer system with such devices, the CPU does not execute detailed I/O instructions. Such instructions are stored in a main memory to be executed by a special-purpose processor in the data channel itself.
- The CPU initiates an I/O transfer by sending a control signal to the data channel, instructing it to execute a sequence of instructions in memory. The data channel performs its task independently of the CPU and signals the CPU when the operation is complete. This arrangement relieves the CPU of a considerable processing burden.
- Another new feature is the **multiplexor**, which is the central termination point for data channels, the CPU, and memory. The multiplexor schedules access to the memory from the CPU and data channels, allowing these devices to act independently



An IBM 7094 Configuration

THE THIRD GENERATION: INTEGRATED CIRCUITS

- A single, self-contained transistor is called a **discrete component**. Throughout the 1950s and early 1960s, electronic equipment was composed largely of discrete components—transistors, resistors, capacitors, and so on.
- In 1958 came the achievement that revolutionized electronics and started the era of microelectronics: the invention of the **integrated circuit**.
- It is the integrated circuit that defines the third generation of computers.

- MICROELECTRONICS

Microelectronics means, literally, "small electronics." The basic elements of a digital computer, as we know, must perform storage, movement, processing, and control functions. Only two fundamental types of components are required: **gates** and **memory cells**.

A **gate** is a device that implements a simple Boolean or logical function, such as IF A AND B ARE TRUE THEN C IS TRUE (AND gate). Such devices are called gates because they control data flow in much the same way that canal gates control the flow of water.

The **memory cell** is a device that can store one bit of data; that is, the device can be in one of two stable states at any time. By interconnecting large numbers of these fundamental devices, we can construct a computer.

SOURCE: WILLIAM STALLINGS, NINTH EDITION

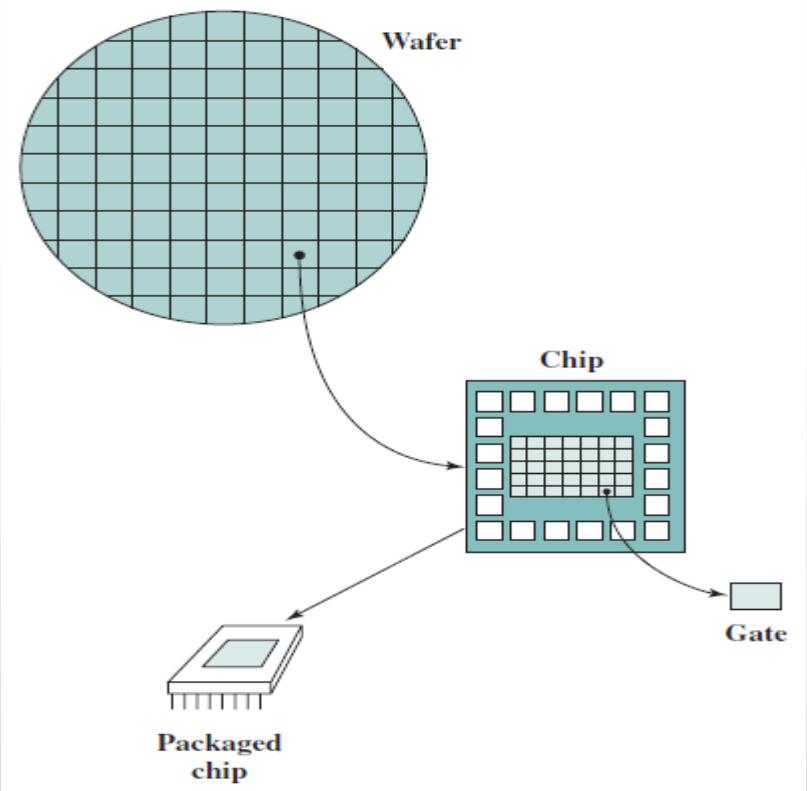
- Four basic functions of a computer:
  - **Data storage**: Provided by memory cells.
  - **Data processing**: Provided by gates.
  - **Data movement**: The paths among components are used to move data from memory to memory and from memory through gates to memory.
  - **Control**: The paths among components can carry control signals. For example, a gate will have one or two data inputs plus a control signal input that activates the gate. When the control signal is ON, the gate performs its function on the data inputs and produces a data output. Similarly, the memory cell will store the bit that is on its input lead when the WRITE control signal is ON and will place the bit that is in the cell on its output lead when the READ control signal is ON.
- Thus, a computer consists of gates, memory cells, and interconnections among these elements. The gates and memory cells are, in turn, constructed of simple digital electronic components.

SOURCE: WILLIAM STALLINGS, NINTH EDITION

- The integrated circuit exploits the fact that such components as transistors, resistors, and conductors can be fabricated from a semiconductor such as silicon.
- It is merely an extension of the solid-state art to fabricate an entire circuit in a tiny piece of silicon rather than assemble discrete components made from separate pieces of silicon into the same circuit.

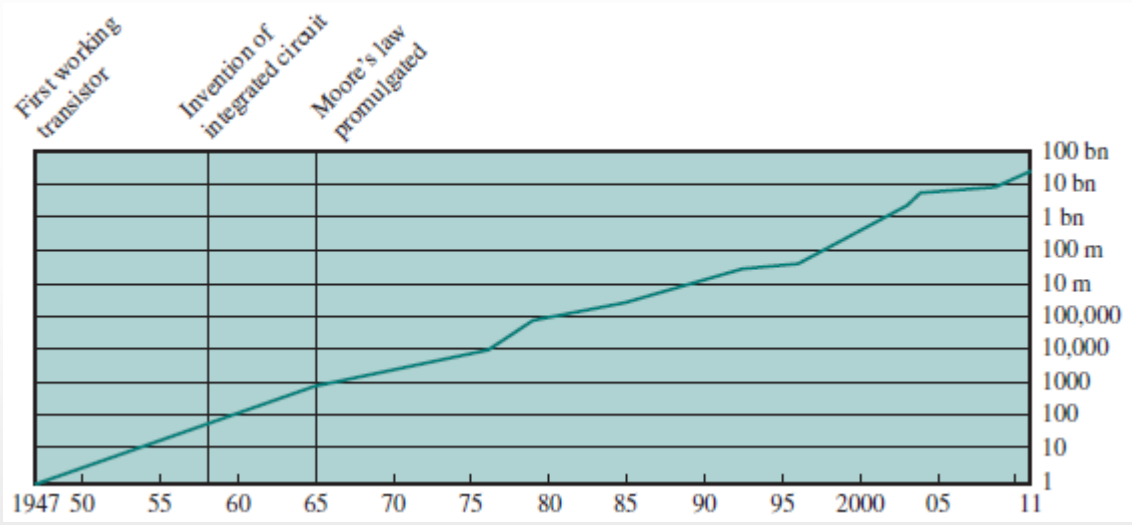
SOURCE: WILLIAM STALLINGS, NINTH EDITION

- A thin wafer of silicon is divided into a matrix of small areas, each a few millimeters square.
- The identical circuit pattern is fabricated in each area, and the wafer is broken up into chips.
- Each chip consists of many gates and/or memory cells plus a number of input and output attachment points.
- This chip is then packaged in housing that protects it and provides pins for attachment to devices beyond the chip.
- A number of these packages can then be interconnected on a printed circuit board to produce larger and more complex circuits.



Relationship among Wafer, Chip, and Gate

- Initially, only a few gates or memory cells could be reliably manufactured and packaged together. These early integrated circuits are referred to as **small scale integration (SSI).**
- As time went on, it became possible to pack more and more components on the same chip.
- Gordon Moore, cofounder of Intel, observed that the number of transistors that could be put on a single chip was doubling every year and correctly predicted that this pace would continue into the near future.



Growth in Transistor Count on Integrated Circuits

- The consequences of Moore's law are profound:
  a. The cost of a chip has remained virtually unchanged during this period of rapid growth in density. This means that the cost of computer logic and memory circuitry has fallen at a dramatic rate.
  b. Because logic and memory elements are placed closer together on more densely packed chips, the electrical path length is shortened, increasing operating speed.
  c. The computer becomes smaller, making it more convenient to place in a variety of environments.
  d. There is a reduction in power and cooling requirements.
  e. The interconnections on the integrated circuit are much more reliable than solder connections. With more circuitry on each chip, there are fewer interchip connections.

- IBM SYSTEM/360

In 1964, IBM announced the System/360, a new family of computer products. The 360 product line was incompatible with older IBM machines. Thus, the transition to the 360 would be difficult for the current customer base. Hence, it break out of some of the constraints of the 7000 architecture and targeted to produce a system capable of evolving with the new integrated circuit technology.

With some modifications and extensions, the architecture of the 360 remains to this day the architecture of IBM's mainframe computers.

The System/360 was the industry's first planned family of computers. The concept of a family of compatible computers was both novel and extremely successful.

The System/360 not only dictated the future course of IBM but also had a profound impact on the entire industry. Many of its features have become standard on other large computers.

SOURCE: WILLIAM STALLINGS, NINTH EDITION

- The characteristics of a family are as follows:
  - **Similar or identical instruction set**: In many cases, the exact same set of machine instructions is supported on all members of the family. Thus, a program that executes on one machine will also execute on any other. In some cases, the lower end of the family has an instruction set that is a subset of that of the top end of the family. This means that programs can move up but not down.
  - **Similar or identical operating system**: The same basic operating system is available for all family members. In some cases, additional features are added to the higher-end members.
  - **Increasing speed**: The rate of instruction execution increases in going from lower to higher family members.
  - **Increasing number of I/O ports**: The number of I/O ports increases in going from lower to higher family members.
  - **Increasing memory size**: The size of main memory increases in going from lower to higher family members.
  - **Increasing cost**: At a given point in time, the cost of a system increases in going from lower to higher family members.

SOURCE: WILLIAM STALLINGS, NINTH EDITION

Characteristic	Model 30	Model 40	Model 50	Model 65	Model 75
Maximum memory size (bytes)	64K	256K	256K	512K	512K
Data rate from memory (Mbytes/s)	0.5	0.8	2.0	8.0	16.0
Processor cycle time (μs)	1.0	0.625	0.5	0.25	0.2
Relative speed	1	3.5	10	21	50
Maximum number of data channels	3	3	4	6	6
Maximum data rate on one channel (Kbytes/s)	250	400	800	1250	1250

Key Characteristics of the System/360 Family

SOURCE: WILLIAM STALLINGS, NINTH EDITION

- DEC PDP-8

In the same year that IBM shipped its first System/360, another momentous first shipment occurred: PDP-8 from Digital Equipment Corporation (DEC). At a time when the average computer required an airconditioned room, the PDP-8 (dubbed a minicomputer by the industry, after the miniskirt of the day) was small enough that it could be placed on top of a lab bench or be built into other equipment. It could not do everything the mainframe could, and was cheap enough for each lab technician to have one. In contrast, the System/360 series of mainframe computers introduced just a few months before cost hundreds of thousands of dollars.

The low cost and small size of the PDP-8 enabled another manufacturer to purchase a PDP-8 and integrate it into a total system for resale. These other manufacturers came to be known as original equipment manufacturers (OEMs), and the OEM market became and remains a major segment of the computer marketplace.

SOURCE: WILLIAM STALLINGS, NINTH EDITION

Model	First Shipped	Cost of Processor + 4K 12-bit Words of Memory ($1000s)	Data Rate from Memory (words/μs)	Volume (cubic feet)	Innovations and Improvements
PDP-8	4/65	16.2	1.26	8.0	Automatic wire-wrapping production
PDP-8/5	9/66	8.79	0.08	3.2	Serial instruction implementation
PDP-8/1	4/68	11.6	1.34	8.0	Medium-scale integrated circuits
PDP-8/L	11/68	7.0	1.26	2.0	Smaller cabinet
PDP-8/E	3/71	4.99	1.52	2.2	Omnibus
PDP-8/M	6/72	3.69	1.52	1.8	Half-size cabinet with fewer slots than 8/E
PDP-8/A	1/75	2.6	1.34	1.2	Semiconductor memory; floating-point processor

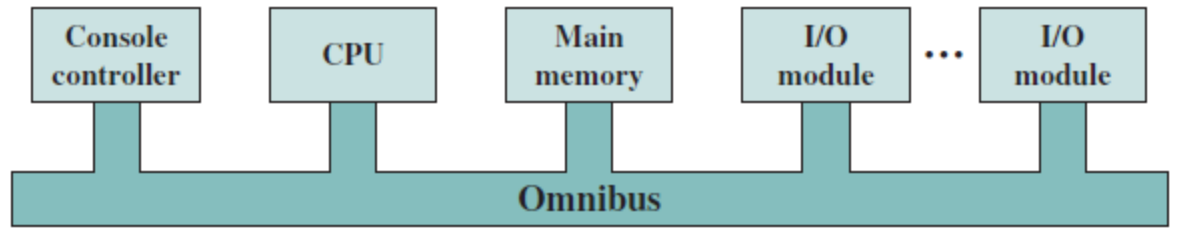
Evolution of the PDP-8

SOURCE: WILLIAM STALLINGS, NINTH EDITION

In contrast to the central-switched architecture used by IBM on its 700/7000 and 360 systems, later models of the PDP-8 used a structure that is now virtually universal for microcomputers: the bus structure.

The PDP-8 bus, called the Omnibus, consists of 96 separate signal paths, used to carry control, address, and data signals. Because all system components share a common set of signal paths, their use can be controlled by the CPU. This architecture is highly flexible, allowing modules to be plugged into the bus to create various configurations.

SOURCE: WILLIAM STALLINGS, NINTH EDITION



PDP-8 Bus Structure

LATER GENERATIONS

- Beyond the third generation there is less general agreement on defining generations of computers.
- There have been a number of later generations, based on advances in integrated circuit technology.
- With the introduction of large-scale integration (LSI), more than 1000 components can be placed on a single integrated circuit chip. Very-large-scale integration (VLSI) achieved more than 10,000 components per chip, while current ultra-large-scale integration (ULSI) chips can contain more than one billion components.

- SEMICONDUCTOR MEMORY

The first application of integrated circuit technology to computers was construction of the processor (the control unit and the arithmetic and logic unit) out of integrated circuit chips. But it was also found that this same technology could be used to construct memories.

In the 1950s and 1960s, most computer memory was constructed from tiny rings of ferromagnetic material, each about a sixteenth of an inch in diameter.

These rings were strung up on grids of fine wires suspended on small screens inside the computer. Magnetized one way, a ring (called a core) represented a one; magnetized the other way, it stood for a zero. Magnetic-core memory was rather fast; it took as little as a millionth of a second to read a bit stored in memory. But it was expensive, bulky, and used destructive readout: The simple act of reading a core erased the data stored in it. It was therefore necessary to install circuits to restore the data as soon as it had been extracted.

Then, in 1970, Fairchild produced the first relatively capacious semiconductor memory. This chip, about the size of a single core, could hold 256 bits of memory. It was nondestructive and much faster than core. It took only 70 billionths of a second to read a bit. However, the cost per bit was higher than for that of core.

In 1974, a seminal event occurred: The price per bit of semiconductor memory dropped below the price per bit of core memory. Following this, there has been a continuing and rapid decline in memory cost accompanied by a corresponding increase in physical memory density. This has led the way to smaller, faster machines with memory sizes of larger and more expensive machines from just a few years earlier.

Since 1970, semiconductor memory has been through 13 generations: 1K, 4K, 16K, 64K, 256K, 1M, 4M, 16M, 64M, 256M, 1G, 4G, and, as of this writing, 16 Gbits on a single chip (1K = 2^{10}, 1M = 2^{20}, 1G = 2^{30}). Each generation has provided four times the storage density of the previous generation, accompanied by declining cost per bit and declining access time.

- MICROPROCESSORS

As time went on, more and more elements were placed on each chip, so that fewer and fewer chips were needed to construct a single computer processor.

A breakthrough was achieved in 1971, when Intel developed its 4004. The 4004 was the first chip to contain all of the components of a CPU on a single chip: The microprocessor was born.

The 4004 can add two 4-bit numbers and can multiply only by repeated addition. By today's standards, the 4004 is hopelessly primitive, but it marked the beginning of a continuing evolution of microprocessor capability and power.

This evolution can be seen most easily in the number of bits that the processor deals with at a time.

- **Data bus width**: the number of bits of data that can be brought into or sent out of the processor at a time.
- Number of bits in the accumulator or in the set of general-purpose registers.

The next major step in the evolution of the microprocessor was the introduction in 1972 of the Intel 8008. This was the first 8-bit microprocessor and was almost twice as complex as the 4004. This was the first general-purpose microprocessor, designed to be the CPU of a general-purpose microcomputer. The 8080, however, is faster, has a richer instruction set, and has a large addressing capability.

About the same time, 16-bit microprocessors began to be developed. One of these was the 8086. The next step in this trend occurred in 1981, when both Bell Labs and Hewlett-Packard developed 32-bit, single-chip microprocessors. Intel introduced its own 32-bit microprocessor, the 80386, in 1985.

SOURCE: WILLIAM STALLINGS, NINTH EDITION

(a) 1970s Processors

	4004	8008	8080	8086	8088
Introduced	1971	1972	1974	1978	1979
Clock speeds	108 kHz	108 kHz	2 MHz	5 MHz, 8 MHz, 10 MHz	5 MHz, 8 MHz
Bus width	4 bits	8 bits	8 bits	16 bits	8 bits
Number of transistors	2300	3500	6000	29,000	29,000
Feature size (μm)	10		6	3	6
Addressable memory	640 Bytes	16 kB	64 kB	1 MB	1 MB

(b) 1980s Processors

	80286	386TM DX	386TM SX	486TM DX CPU
Introduced	1982	1985	1988	1989
Clock speeds	6 MHz–12.5 MHz	16 MHz–33 MHz	16 MHz–33 MHz	25 MHz–50 MHz
Bus width	16 bits	32 bits	16 bits	32 bits
Number of transistors	134,000	275,000	275,000	1.2 million
Feature size (μm)	1.5	1	1	0.8–1
Addressable memory	16 MB	4 GB	16 MB	4 GB
Virtual memory	1 GB	64 TB	64 TB	64 TB
Cache	–	–	–	8 kB

(c) 1990s Processors

	486TM SX	Pentium	Pentium Pro	Pentium II
Introduced	1991	1993	1995	1997
Clock speeds	16 MHz–33 MHz	60 MHz–166 MHz,	150 MHz–200 MHz	200 MHz–300 MHz
Bus width	32 bits	32 bits	64 bits	64 bits
Number of transistors	1.185 million	3.1 million	5.5 million	7.5 million
Feature size (μm)	1	0.8	0.6	0.35
Addressable memory	4 GB	4 GB	64 GB	64 GB
Virtual memory	64 TB	64 TB	64 TB	64 TB
Cache	8 kB	8 kB	512 kB L1 and 1 MB L2	512 kB L2

(d) Recent Processors

	Pentium III	Pentium 4	Core 2 Duo	Core i7 EE 990
Introduced	1999	2000	2006	2011
Clock speeds	450–660 MHz	1.3–1.8 GHz	1.06–1.2 GHz	3.5 GHz
Bus width	64 bits	64 bits	64 bits	64 bits
Number of transistors	9.5 million	42 million	167 million	1170 million
Feature size (nm)	250	180	65	32
Addressable memory	64 GB	64 GB	64 GB	64 GB
Virtual memory	64 TB	64 TB	64 TB	64 TB
Cache	512 kB L2	256 kB L2	2 MB L2	1.5 MB L2/12 MB L3

Evolution of Intel Microprocessors

COMPUTER TYPES

The different types of computers are:

- **Personal computers**: This is the most common type found in homes, schools, business offices, etc. It is the most common type of desktop computers with processing and storage units along with various in put and out put devices.
- Notebook computers: These are compact and portable versions of PC.
- **Work stations**: These have high resolution input/output (I/O) graphics capability, but with same dimensions as that of desktop computer. These are used in engineering applications of interactive design work.
- **Enterprise systems**: These are used for business data processing in medium to large corporations that require much more computing power and storage capacity than work stations. Internet associated with servers have become a dominant worldwide source of all types of information.
- **Super computers**: These are used for large scale numerical calculations required in the applications like weather forecasting etc.

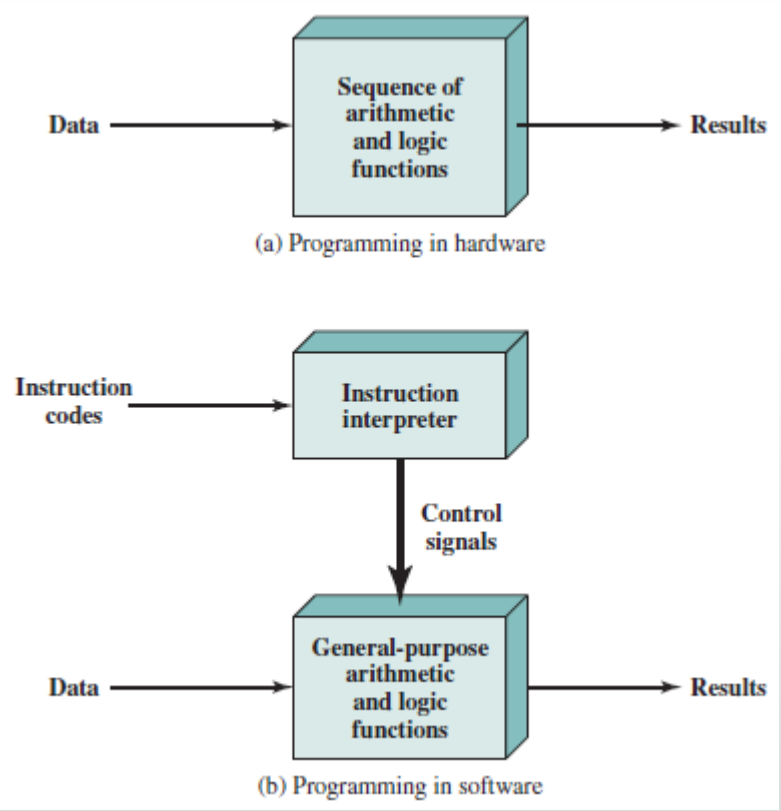
SOURCE:
HTTPS://WWW.EIT.EDU.AU/CMS/RESOURCES/TECHNICAL-RESOURSES/TYPES-AND-FUNCTIONAL-UNIT-OF-COMPUTERS

SOURCE: WILLIAM STALLINGS, NINTH EDITION

- A computer consists of CPU (central processing unit), memory, and I/O components, with one or more modules of each type.
- These components are interconnected in some fashion to achieve the basic function of the computer, which is to execute programs.
- Thus, at a top level, we can characterize a computer system by describing
  (1) the external behavior of each component, that is, the data and control signals that it exchanges with other components and
  (2) the interconnection structure and the controls required to manage the use of the interconnection structure.

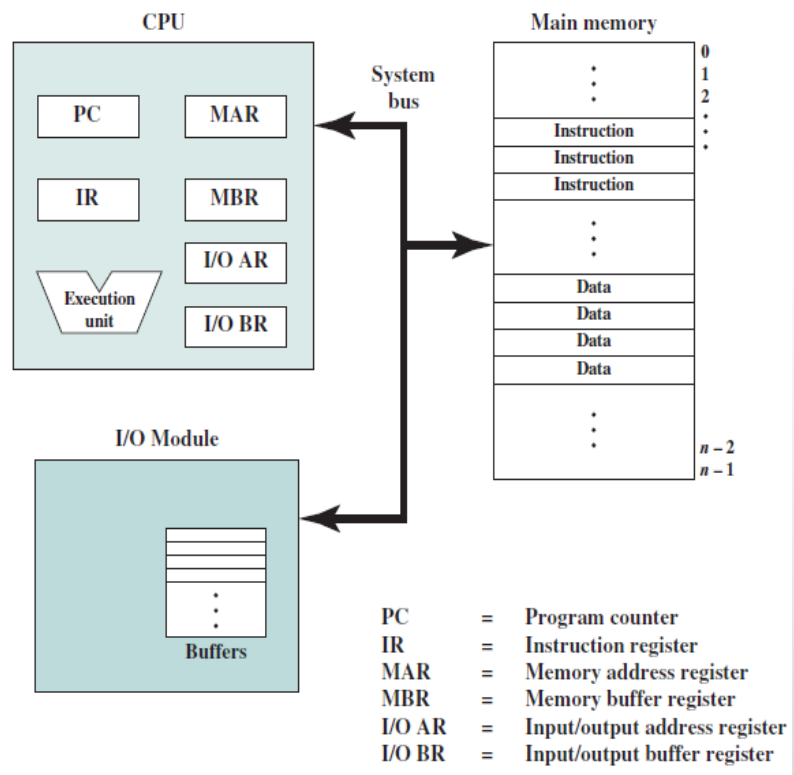
SOURCE: WILLIAM STALLINGS, NINTH EDITION

- A computer design referred to as the von Neumann architecture is based on three key concepts:
  - Data and instructions are stored in a single read–write memory.
  - The contents of this memory are addressable by location, without regard to the type of data contained there.
  - Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next.
- There is a small set of basic logic components that can be combined in various ways to store binary data and perform arithmetic and logical operations on that data. The process of connecting the various components in the desired configuration as a form of programming is termed a *hardwired program* where the resulting "program" is in the form of hardware.
- Instead of rewiring the hardware for each new program, all we need to do is provide a new sequence of codes. Each code is, in effect, an instruction, and part of the hardware interprets each instruction and generates control signals. To distinguish this new method of programming, a sequence of codes or instructions is called *software*.



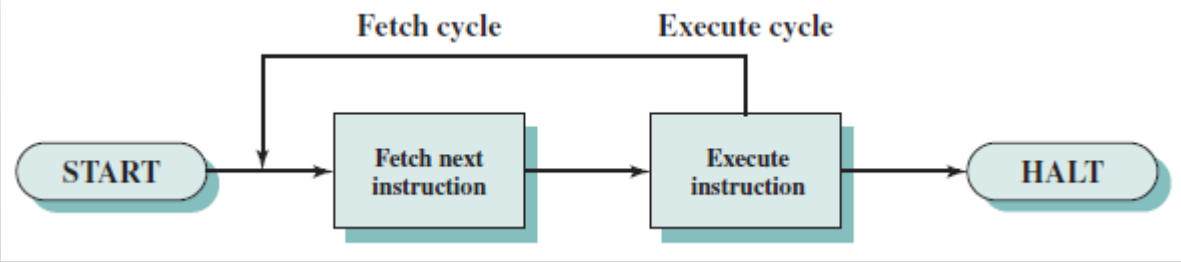
SOURCE: WILLIAM STALLINGS, NINTH EDITION

- Data and instructions must be put into the system. For this we need some sort of input module. This module contains basic components for accepting data and instructions in some form and converting them into an internal form of signals usable by the system. A means of reporting results is needed, and this is in the form of an output module. Taken together, these are referred to as ***I/O components***.
- There must be a place to store temporarily both instructions and data. That module is called ***memory, or main memory***, to distinguish it from external storage or peripheral devices.
- The CPU exchanges data with memory. For this purpose, it typically makes use of ***two internal (to the CPU) registers***: *a memory address register (MAR)*, which specifies the address in memory for the next read or write, *and a memory buffer register (MBR)*, which contains the data to be written into memory or receives the data read from memory. Similarly, an *I/O address register (I/OAR)* specifies a particular I/O device. An *I/O buffer (I/OBR) register* is used for the exchange of data between an I/O module and the CPU.



COMPUTER FUNCTION

- The basic function performed by a computer is execution of a program, which consists of a set of instructions stored in memory.
- The processor does the actual work by executing instructions specified in the program.
- In its simplest form, instruction processing consists of two steps:
  - The processor reads (fetches) instructions from memory one at a time and executes each instruction.
  - Program execution consists of repeating the process of instruction fetch and instruction execution.
- The processing required for a single instruction is called an **instruction cycle** consisting of two steps are referred to as **the fetch cycle** and **the execute cycle.**

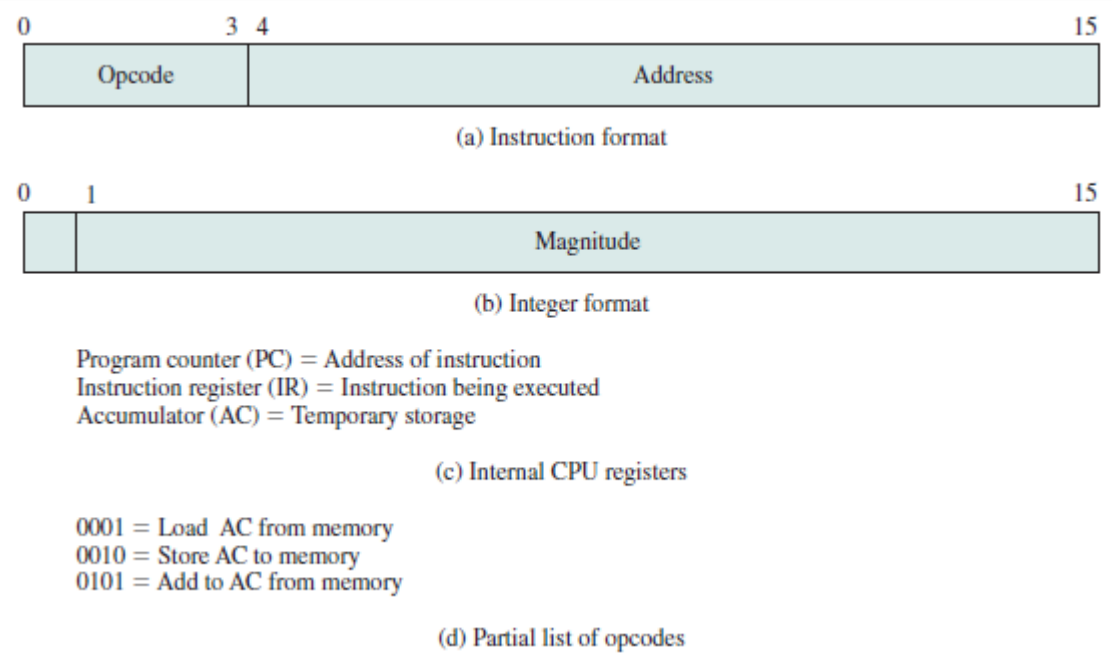


SOURCE: WILLIAM STALLINGS, NINTH EDITION

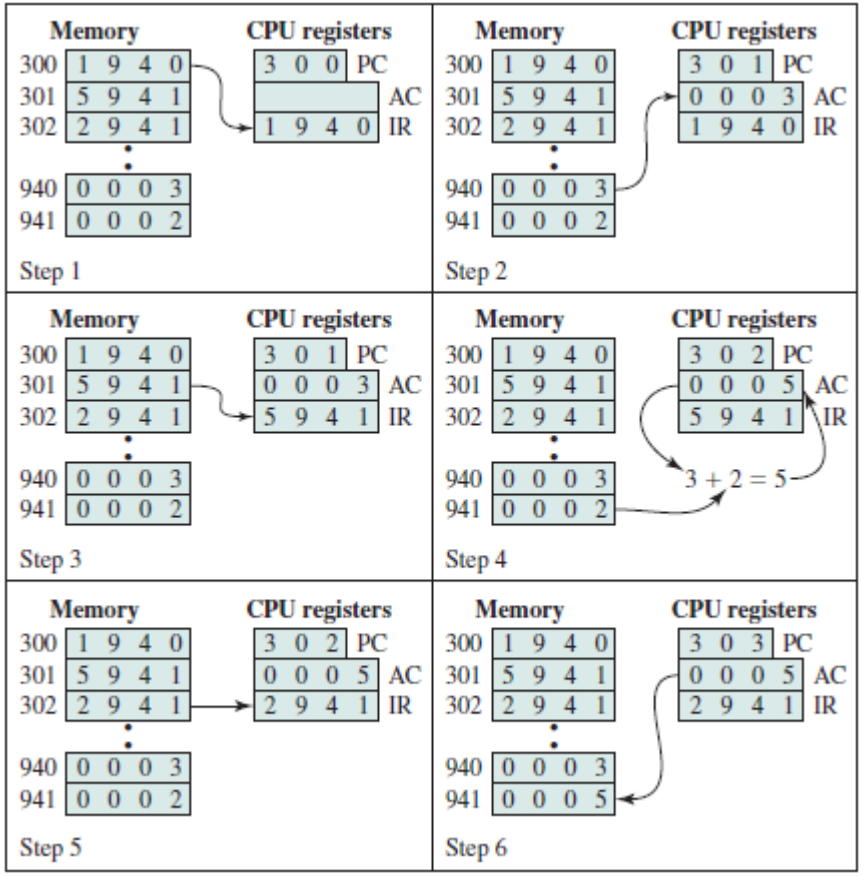
SOURCE: WILLIAM STALLINGS, NINTH EDITION

- At the beginning of each instruction cycle, the processor **fetches** an instruction from memory. In a typical processor, a register called the program counter (PC) holds the address of the instruction to be fetched next. Unless told otherwise, the processor always increments the PC after each instruction fetch so that it will fetch the next instruction in sequence (i.e., the instruction located at the next higher memory address).
- The fetched instruction is loaded into a register in the processor known as the *instruction register (IR)*. The instruction contains bits that specify the action the processor is to take. The processor interprets the instruction and performs the required action.

- In general, these actions fall into four categories:
  - **Processor-memory**: Data may be transferred from processor to memory or from memory to processor.
  - **Processor-I/O**: Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.
  - **Data processing**: The processor may perform some arithmetic or logic operation on data.
  - **Control**: An instruction may specify that the sequence of execution be altered.
- The **execution** cycle for a particular instruction may involve more than one reference to memory. Also, instead of memory references, an instruction may specify an I/O operation.



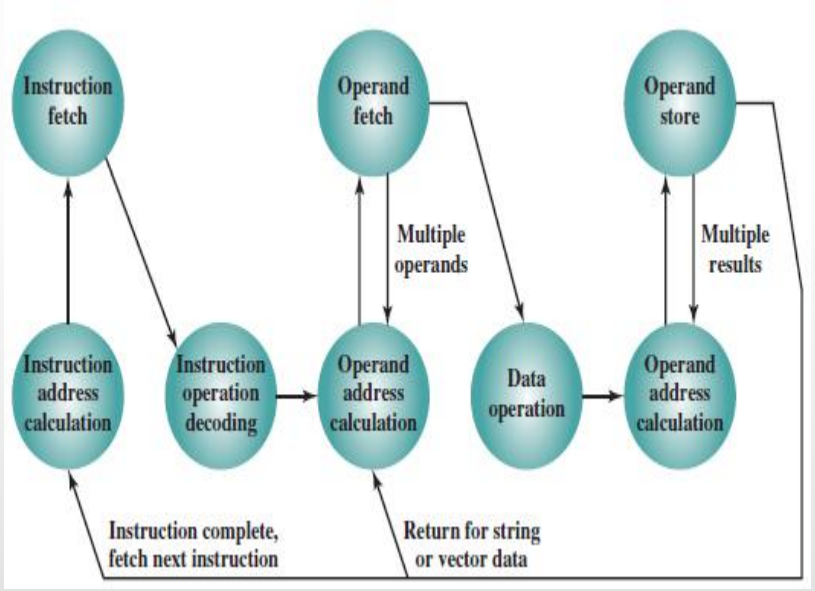
Characteristics of a Hypothetical Machine



Example of Program Execution
(contents of memory and registers in hexadecimal)

SOURCE: WILLIAM STALLINGS, NINTH EDITION

- **Instruction address calculation (iac):** Determine the address of the next instruction to be executed.
- **Instruction fetch (if):** Read instruction from its memory location into the processor.
- **Instruction operation decoding (iod):** Analyze instruction to determine type of operation to be performed and operand(s) to be used.
- **Operand address calculation (oac):** If the operation involves reference to an operand in memory or available via I/O, then determine the address of the operand.
- **Operand fetch (of):** Fetch the operand from memory or read it in from I/O.
- **Data operation (do):** Perform the operation indicated in the instruction.
- **Operand store (os):** Write the result into memory or out to I/O.

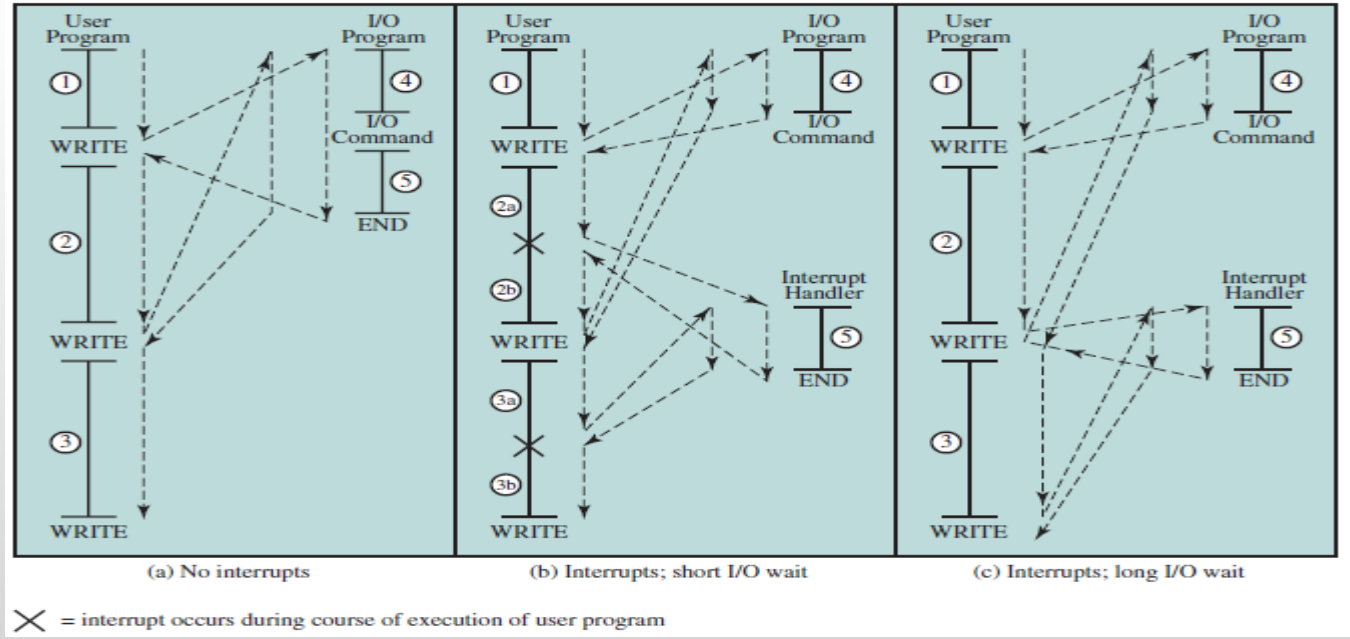


Instruction Cycle State Diagram

SOURCE: WILLIAM STALLINGS, NINTH EDITION

- INTERRUPTS

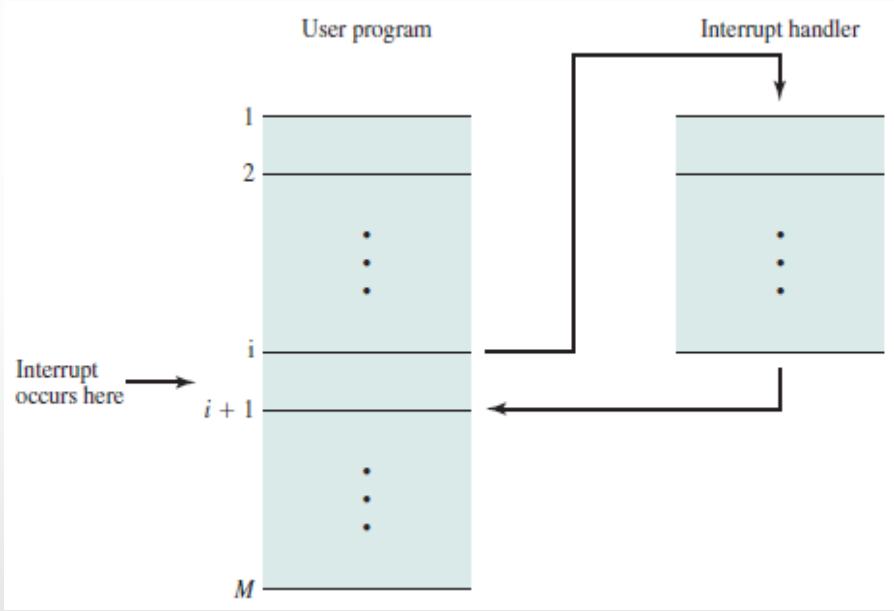
Virtually all computers provide a mechanism by which other modules (I/O, memory) may interrupt the normal processing of the processor. **Interrupts** are provided primarily as a way to improve processing efficiency.



Program Flow of Control without and with Interrupts

- INTERRUPTS AND THE INSTRUCTION CYCLE

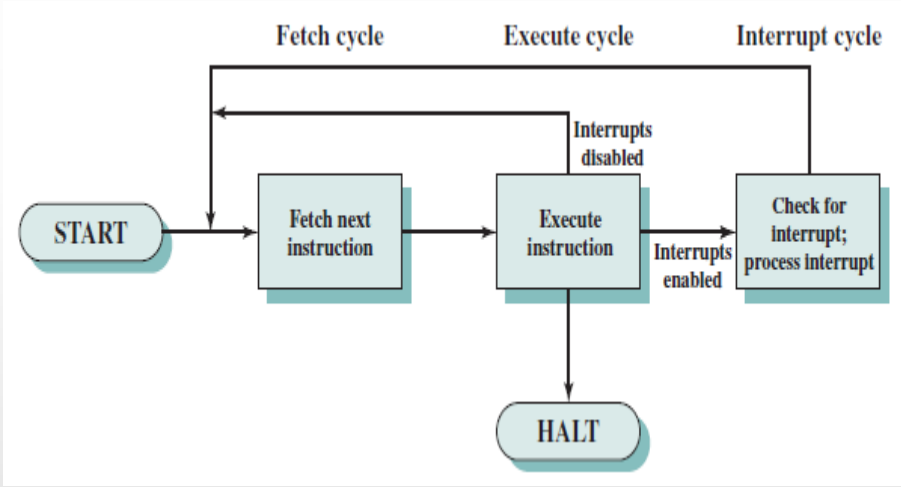
With interrupts, the processor can be engaged in executing other instructions while an I/O operation is in progress. When the external device becomes ready to be serviced—that is, when it is ready to accept more data from the processor—the I/O module for that external device sends an interrupt request signal to the processor. The processor responds by suspending operation of the current program, branching off to a program to service that particular I/O device, known as an **interrupt handler**, and resuming the original execution after the device is serviced.



Transfer of Control via Interrupts

SOURCE: WILLIAM STALLINGS, NINTH EDITION

- To accommodate interrupts, an interrupt cycle is added to the instruction cycle.
- In the interrupt cycle, the processor checks to see if any interrupts have occurred, indicated by the presence of an interrupt signal.
- If no interrupts are pending, the processor proceeds to the fetch cycle and fetches the next instruction of the current program.
- If an interrupt is pending, the processor does the following:
  - It suspends execution of the current program being executed and saves its context. This means saving the address of the next instruction to be executed (current contents of the program counter) and any other data relevant to the processor's current activity.
  - It sets the program counter to the starting address of an interrupt handler routine.



- MULTIPLE INTERRUPTS

For some scenarios in computer, multiple interrupts can occur. Two approaches can be taken to dealing with multiple interrupts.

1. The first is to disable interrupts while an interrupt is being processed. A **disabled interrupt** simply means that the processor can and will ignore that interrupt request signal.

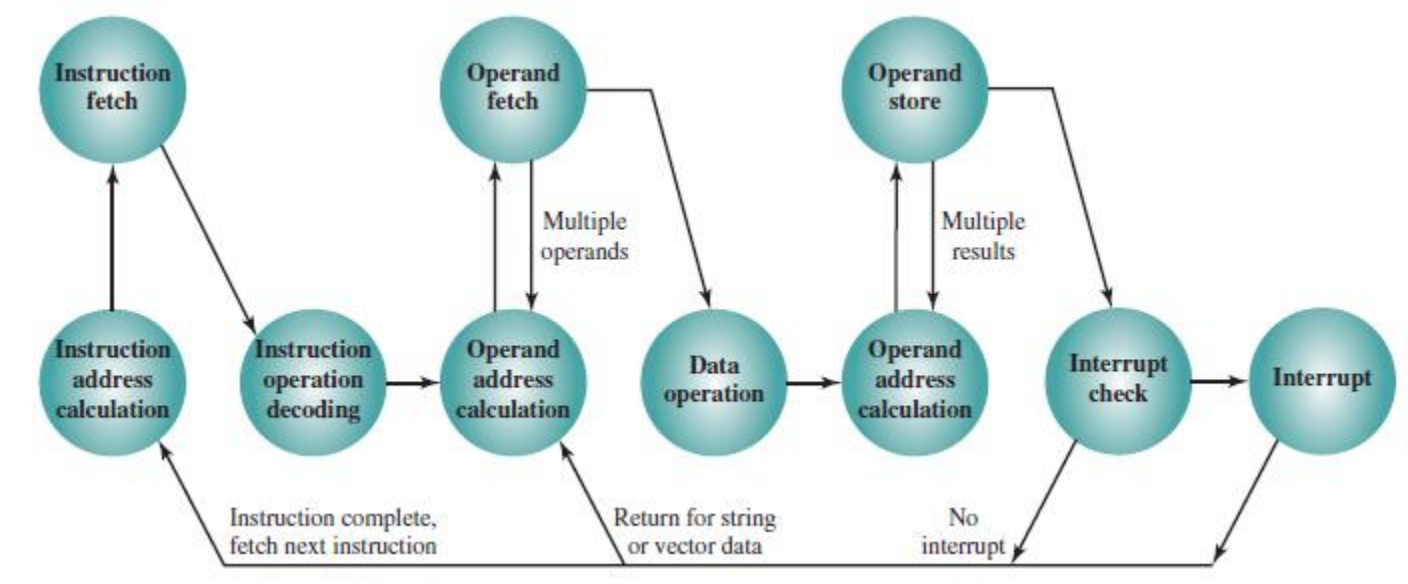
   If an interrupt occurs during this time, it generally remains pending and will be checked by the processor after the processor has enabled interrupts. Thus, when a user program is executing and an interrupt occurs, interrupts are disabled immediately. After the interrupt handler routine completes, interrupts are enabled before resuming the user program, and the processor checks to see if additional interrupts have occurred. This approach is nice and simple, as interrupts are handled in strict sequential order.

   The drawback to the preceding approach is that it does not take into account

   relative priority or time-critical needs.

2. A second approach is to define priorities for interrupts and to allow an interrupt of higher priority to cause a lower-priority interrupt handler to be itself interrupted.

SOURCE: WILLIAM STALLINGS, NINTH EDITION



Instruction Cycle State Diagram, with Interrupts

- I/O Function

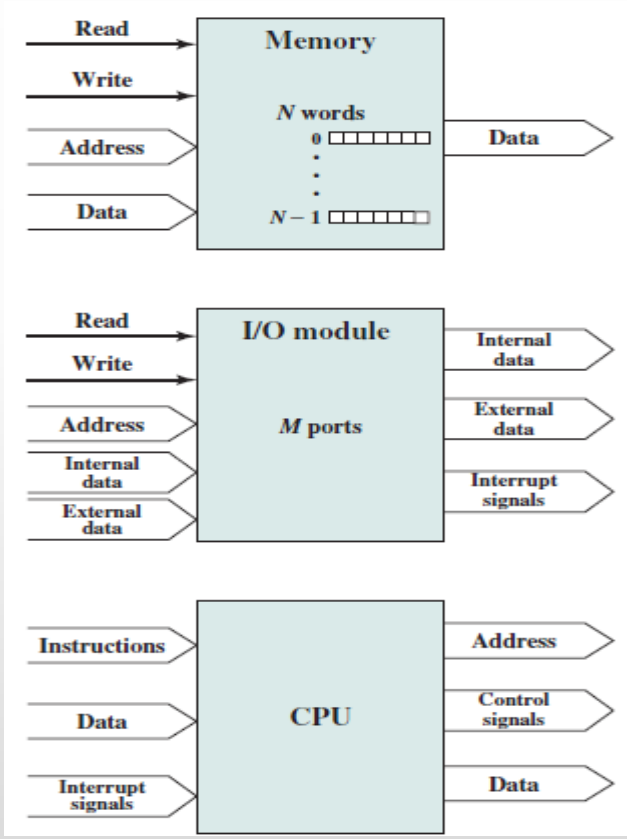
An I/O module (e.g., a disk controller) can exchange data directly with the processor. Just as the processor can initiate a read or write with memory, designating the address of a specific location, the processor can also read data from or write data to an I/O module. In this latter case, the processor identifies a specific device that is controlled by a particular I/O module. Thus, an instruction sequence is similar in form of program execution to that with I/O instructions rather than memory-referencing instructions.

In some cases, it is desirable to allow I/O exchanges to occur directly with memory. In such a case, the processor grants to an I/O module the authority to read from or write to memory, so that the I/O-memory transfer can occur without tying up the processor. During such a transfer, the I/O module issues read or write commands to memory, relieving the processor of responsibility for the exchange. This operation is known as **direct memory access (DMA).**

SOURCE: WILLIAM STALLINGS, NINTH EDITION

INTERCONNECTION STRUCTURES

- A computer consists of a set of components or modules of three basic types (*processor, memory, I/O*) that communicate with each other.
- The collection of paths connecting the various modules is called the *interconnection structure*. The design of this structure will depend on the exchanges that must be made among modules.
- The interconnection structure must support the following types of transfers:
  - **Memory to processor**: The processor reads an instruction or a unit of data from memory.
  - **Processor to memory**: The processor writes a unit of data to memory.
  - **I/O to processor**: The processor reads data from an I/O device via an I/O module.
  - **Processor to I/O**: The processor sends data to the I/O device.
  - **I/O to or from memory**: For these two cases, an I/O module is allowed to exchange data directly with memory, without going through the processor, using direct memory access.



Computer Modules

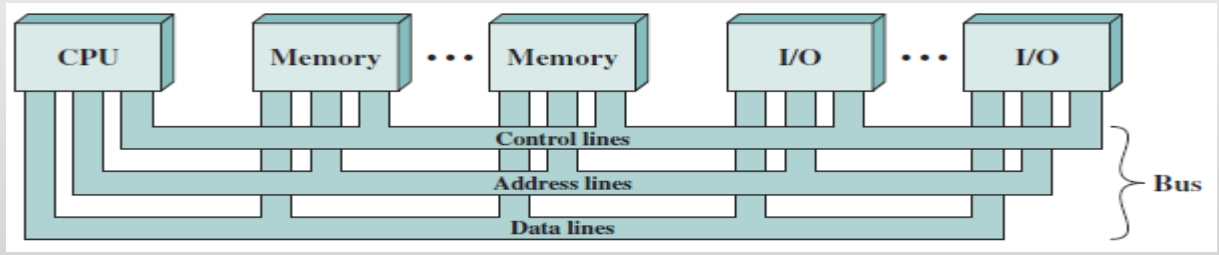
BUS INTERCONNECTION

- A bus is a communication pathway connecting two or more devices.
- A key characteristic of a bus is that it is a shared transmission medium where multiple devices connect to the bus, and a signal transmitted by any one device is available for reception by all other devices attached to the bus.
- If two devices transmit during the same time period, their signals will overlap and become garbled. Thus, only one device at a time can successfully transmit.
- Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy.
- A bus that connects major computer components (processor, memory, I/O) is called a **system bus**.
- The most common computer interconnection structures are based on the use of one or more system buses.

SOURCE: WILLIAM STALLINGS, NINTH EDITION

Bus Structure

- A system bus consists, typically, of from about fifty to hundreds of separate lines.
- Each line is assigned a particular meaning or function.
- Although there are many different bus designs, on any bus the lines can be classified into three functional groups:
  - The **data lines** provide a path for moving data among system modules. These lines, collectively, are called the **data bus**. The data bus may consist of 32, 64, 128, or even more separate lines, the number of lines being referred to as the width of the data bus.
  - The **address lines** are used to designate the source or destination of the data on the data bus. The width of the **address bus** determines the maximum possible memory capacity of the system.
  - The **control lines** are used to control the access to and the use of the data and address lines. The data and address lines are shared by all components, their use is being controlled by **control bus**.



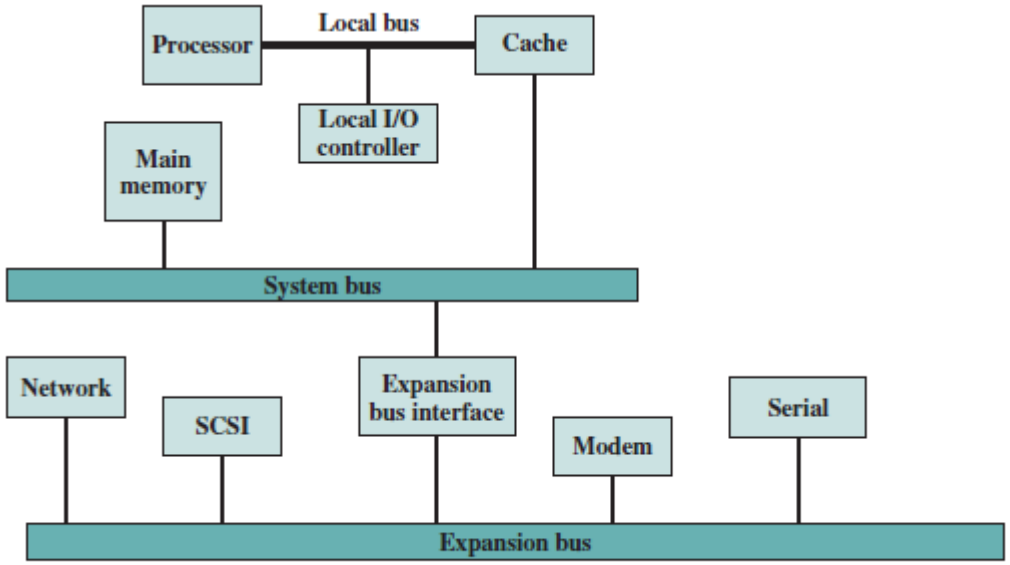
- Control signals transmit both *command* and *timing information* among system modules.
- Timing signals indicate the validity of data and address information. Command signals specify operations to be performed.
- Typical control lines include:
  - **Memory write**: causes data on the bus to be written into the addressed location
  - **Memory read**: causes data from the addressed location to be placed on the bus
  - **I/O write**: causes data on the bus to be output to the addressed I/O port
  - **I/O read**: causes data from the addressed I/O port to be placed on the bus
  - **Transfer ACK**: indicates that data have been accepted from or placed on the bus
  - **Bus request**: indicates that a module needs to gain control of the bus
  - **Bus grant**: indicates that a requesting module has been granted control of the bus
  - **Interrupt request**: indicates that an interrupt is pending
  - **Interrupt ACK**: acknowledges that the pending interrupt has been recognized
  - **Clock**: is used to synchronize operations
  - **Reset**: initializes all modules.

SOURCE: WILLIAM STALLINGS, NINTH EDITION

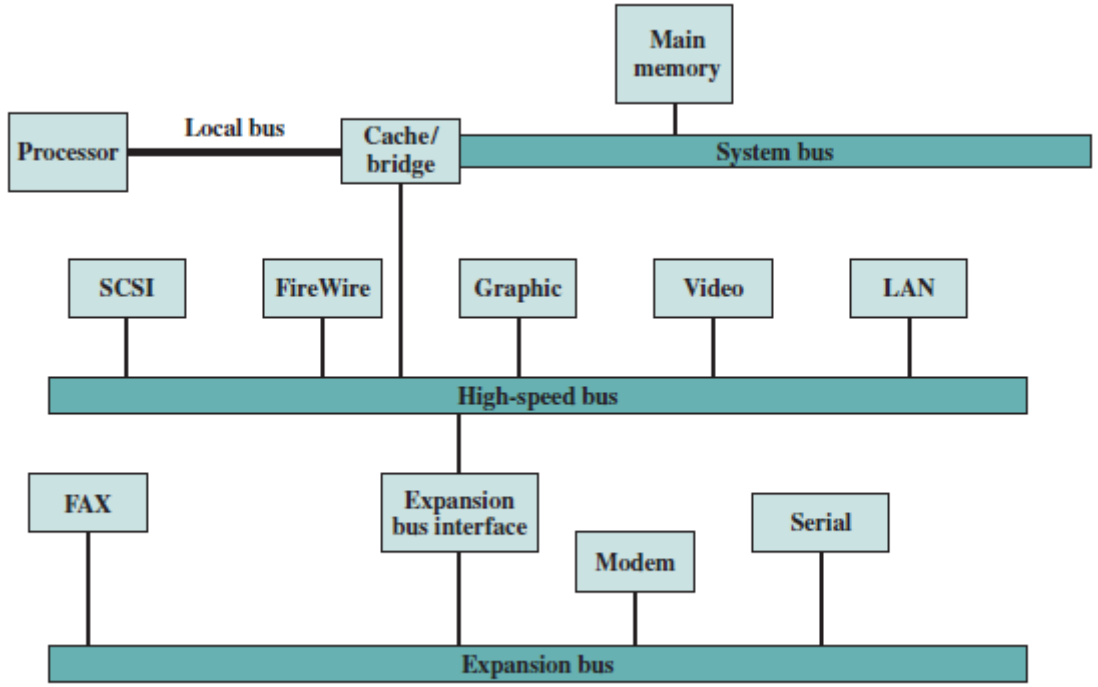
Multiple-bus Hierarchies

- If a great number of devices are connected to the bus, performance will suffer.
  - In general, the more devices attached to the bus, the greater the bus length and hence the greater the propagation delay.
  - The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus.
- The traditional bus architecture is reasonably efficient but begins to break down as higher and higher performance is seen in the I/O devices. In response to these growing demands, a common approach taken by industry is to build a highspeed bus that is closely integrated with the rest of the system, requiring only a bridge between the processor's bus and the high-speed bus. This arrangement is sometimes known as a **mezzanine architecture**.
- The advantage of this arrangement is that the high-speed bus brings high demand devices into closer integration with the processor and at the same time is independent of the processor. Thus, differences in processor and high-speed bus speeds and signal line definitions are tolerated. Changes in processor architecture do not affect the high-speed bus, and vice versa.

SOURCE: WILLIAM STALLINGS, NINTH EDITION



(a) Traditional bus architecture



(b) High-performance architecture

Elements of Bus Design

- **Bus Types**

  Bus lines can be separated into two generic types: *dedicated and multiplexed*.

  Physical dedication refers to the use of multiple buses, each of which connects only a subset of modules.

- **Method of Arbitration**

  In all but the simplest systems, more than one module may need control of the bus where only one unit at a time can successfully transmit over the bus to which some method of arbitration is needed.

  The various methods can be roughly classified as being either centralized *arbitration or distributed arbitration*. In a centralized scheme, a single hardware device, referred to as a bus controller or arbiter, is responsible for allocating time on the bus. In a distributed scheme, there is no central controller. Rather, each module contains access control logic and the modules act together to share the bus.

Type	**Bus Width**
Dedicated	Address
Multiplexed	Data
Method of Arbitration	**Data Transfer Type**
Centralized	Read
Distributed	Write
	Read-modify-write
Timing	Read-after-write
Synchronous	Block
Asynchronous	

With both methods of arbitration, the purpose is to designate one device, either the processor or an I/O module, as master. The master may then initiate a data transfer (e.g., read or write) with some other device, which acts as slave for this particular exchange.

- **Timing**

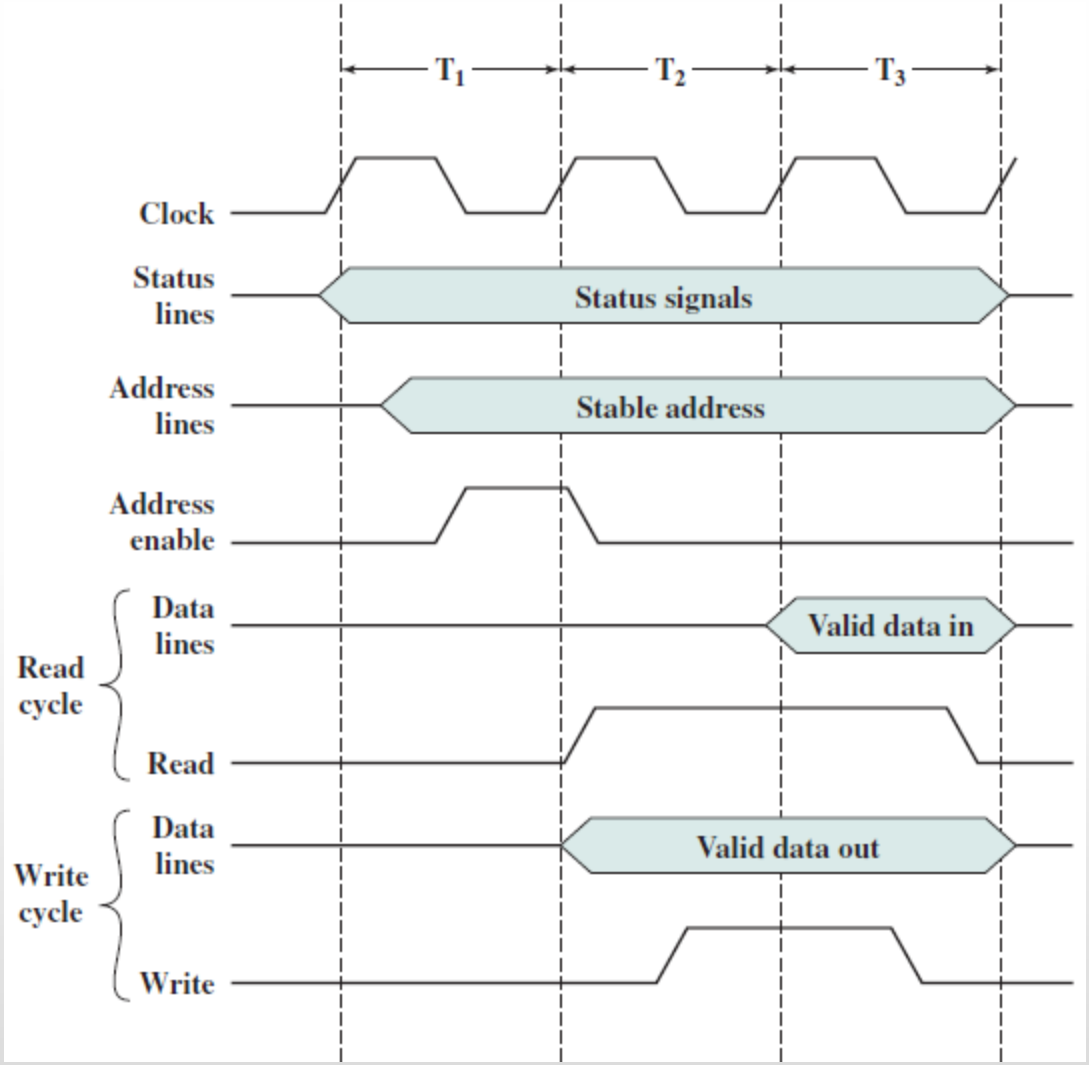
  Timing refers to the way in which events are coordinated on the bus. Buses use either synchronous timing or asynchronous timing.

  With synchronous timing, the occurrence of events on the bus is determined by a clock.

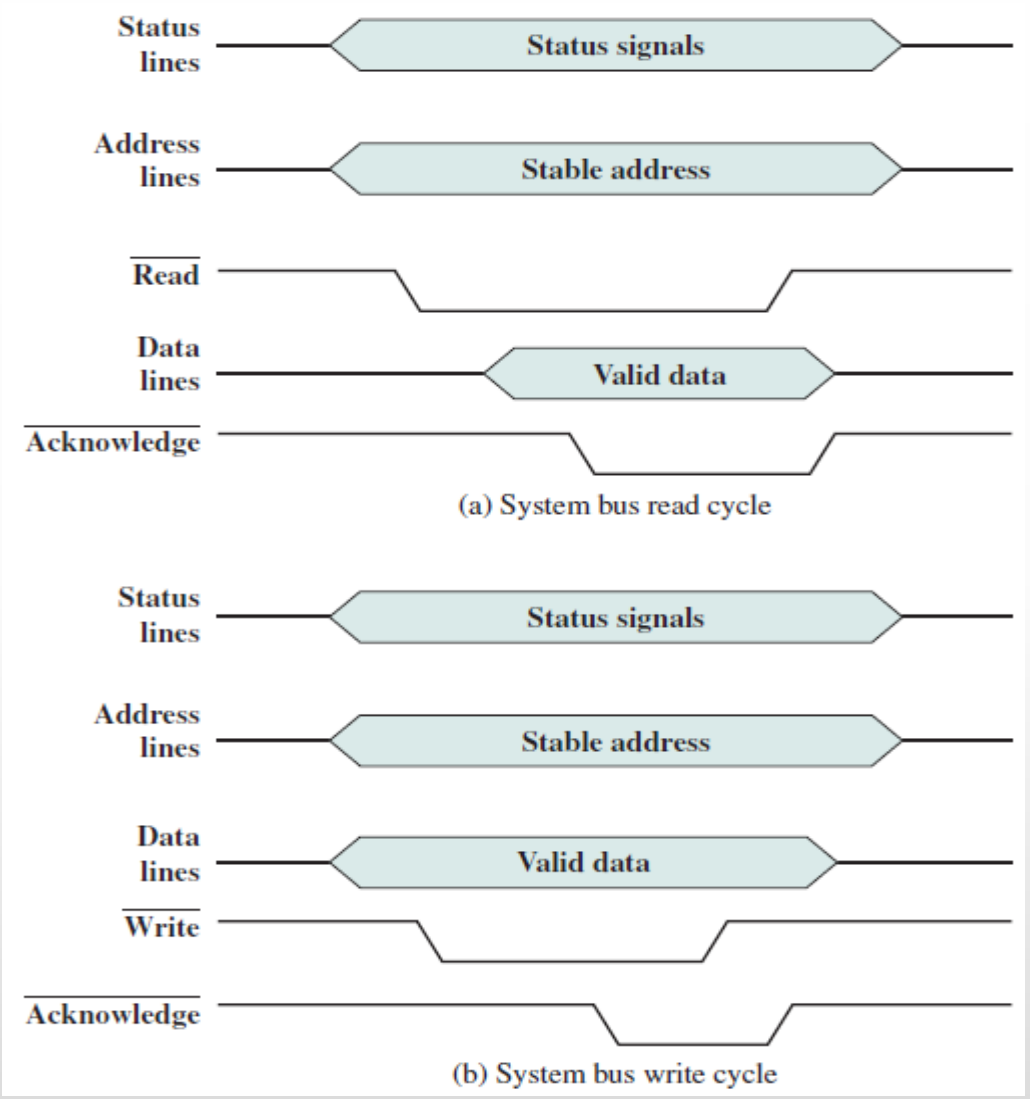
  A single 1–0 transmission is referred to as a clock cycle or bus cycle and defines a time slot.

  With asynchronous timing, the occurrence of one event on a bus follows and depends on the occurrence of a previous event.

SOURCE: WILLIAM STALLINGS, NINTH EDITION



Timing of Synchronous Bus Operations



Timing of Asynchronous Bus Operations

SOURCE: MORRIS MANO, THIRD EDITION

CHAPTER-3: CENTRAL PROCESSING UNIT

ADDITION AND SUBTRACTION OF SIGNED NUMBERS

When an integer binary number is positive, the sign is represented by 0 and the magnitude by a positive binary number. When the number is negative, the sign is represented by 1 but the rest of the number may be represented in one of three possible ways:

1. Signed-magnitude representation
2. Signed-1' s complement representation
3. Signed 2' s complement representation

Example, three different ways to represent – 14

In signed-magnitude representation 1 0001110

In signed-1's complement representation 1 11 10001

In signed-2's complement representation 1 11 10010

SOURCE: MORRIS MANO, THIRD EDITION

- The signed-magnitude system is used in ordinary arithmetic but is awkward when employed in computer arithmetic. Therefore, the signed-complement is normally used.
- The 1' s complement imposes difficulties because it has two representations of 0 (+ 0 and – 0).
- The 1's complement is useful as a logical operation since the change of 1 to 0 or 0 to 1 is equivalent to a logical complement operation.

Arithmetic Addition

- The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic.
- If the signs are the same, we add the two magnitudes and give the sum the common sign.
- If the signs are different, we subtract the smaller magnitude from the larger and give the result the sign of the larger magnitude. For example, (+ 25) + (– 37) = – (37 – 25) = – 12
- This is a process that requires the comparison of the signs and the magnitudes and then performing either addition or subtraction.

SOURCE: MORRIS MANO, THIRD EDITION

SOURCE: MORRIS MANO, THIRD EDITION

- By contrast, the rule for adding numbers in the signed–2's complement system does not require a comparison or subtraction, only addition and complementation.
- The procedure is very simple and can be stated as follows: Add the two numbers, including their sign bits, and discard any carry out of the sign (leftmost) bit position.
- Note that negative numbers must initially be in 2' s complement and that if the sum obtained after the addition is negative, it is in 2's complement form.

+6	00000110	−6	11111010
+13	00001101	+13	00001101
+19	00010011	+7	00000111
+6	00000110	−6	11111010
−13	11110011	−13	11110011
−7	11111001	−19	11101101

SOURCE: MORRIS MANO, THIRD EDITION

Arithmetic Subtraction

- Subtraction of two signed binary numbers when negative numbers are in 2's complement form is very simple and can be stated as follows: Take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including the sign bit). A carry out of the sign bit position is discarded.
- This procedure stems from the fact that a subtraction operation can be changed to an addition operation if the sign of the subtrahend is changed. This is demonstrated by the following relationship:

$$(\pm A) - (+B) = (\pm A) + (-B)$$

$$(\pm A) - (c\ B) = (\pm A) + (+B)$$

Consider the subtraction of (–6) – (– 13) = +7. In binary with eight bits this is written as 11111010 – 11110011. The subtraction is changed to addition by taking the 2's complement of the subtrahend (–13) to give (+13). In binary this is 11111010 + 00001101 = 100000111. Removing the end carry, we obtain the correct answer 00000111 (+ 7).

It is worth noting that binary numbers in the signed-2's complement system are added and subtracted by the same basic addition and subtraction rules as unsigned numbers. Therefore, computers need only one common hardware circuit to handle both types of arithmetic. The user or programmer must interpret the results of such addition or subtraction differently depending on whether it is assumed that the numbers are signed or unsigned.

SOURCE: MORRIS MANO, THIRD EDITION

Overflow

- When two numbers of n digits each are added and the sum occupies n + 1 digits, we say that an **overflow** occurred.
- An overflow is a problem in digital computers because the width of registers is finite. A result that contains n + 1 bits cannot be accommodated in a register with a standard length of n bits.
- For this reason, many computers detect the occurrence of an overflow, and when it occurs, a corresponding flip-flop is set which can then be checked by the user.
- If two numbers are added, and they are both positive or both negative, then overflow occurs if and only if the result has the opposite sign.
- An *overflow condition* can be detected by observing the carry into the sign bit position and the carry out of the sign bit position. If these two carries are not equal, an overflow condition is produced.

Example: Two signed binary numbers, +70 and +80, are stored in two eight-bit registers. The range of numbers that each register can accommodate is from binary +127 to binary –128. Since the sum of the two numbers is +150, it exceeds the capacity of an eight-bit register.

carries:	0	1	carries:	1	0
+70	0	1000110	−70	1	0111010
+80	0	1010000	−80	1	0110000
+150	1	0010110	−150	0	1101010

SOURCE: MORRIS MANO, THIRD EDITION

1001 = −7 +0101 = 5 1110 = −2 (a) (−7) + (+5)	1100 = −4 +0100 = 4 10000 = 0 (b) (−4) + (+4)
0011 = 3 +0100 = 4 0111 = 7 (c) (+3) + (+4)	1100 = −4 +1111 = −1 11011 = −5 (d) (−4) + (−1)
0101 = 5 +0100 = 4 1001 = Overflow (e) (+5) + (+4)	1001 = −7 +1010 = −6 10011 = Overflow (f) (−7) + (−6)

Addition of Numbers in Twos Complement Representation

0010 = 2 +1001 = −7 1011 = −5 (a) M = 2 = 0010 S = 7 = 0111 −S = 1001	0101 = 5 +1110 = −2 10011 = 3 (b) M = 5 = 0101 S = 2 = 0010 −S = 1110
1011 = −5 +1110 = −2 11001 = −7 (c) M = −5 = 1011 S = 2 = 0010 −S = 1110	0101 = 5 +0010 = 2 0111 = 7 (d) M = 5 = 0101 S = −2 = 1110 −S = 0010
0111 = 7 +0111 = 7 1110 = Overflow (e) M = 7 = 0111 S = −7 = 1001 −S = 0111	1010 = −6 +1100 = −4 10110 = Overflow (f) M = −6 = 1010 S = 4 = 0100 −S = 1100

Subtraction of Numbers in Twos Complement Representation (M - S)

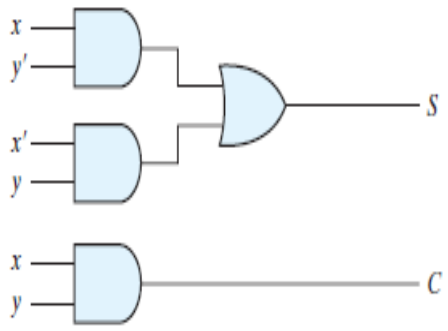
Half Adder

- The half adder circuit needs two binary inputs and two binary outputs. The input variables designate the augend and addend bits; the output variables produce the sum and carry. We assign symbols x and y to the two inputs and S (for sum) and C (for carry) to the outputs. The C output is 1 only when both inputs are 1. The S output represents the least significant bit of the sum.
- The simplified sum-of-products expressions are:

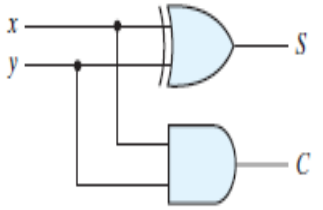
$$S = x'y + xy'$$
$$C = xy$$

Half Adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



(a) $S = xy' + x'y$
$C = xy$



(b) $S = x \oplus y$
$C = xy$

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

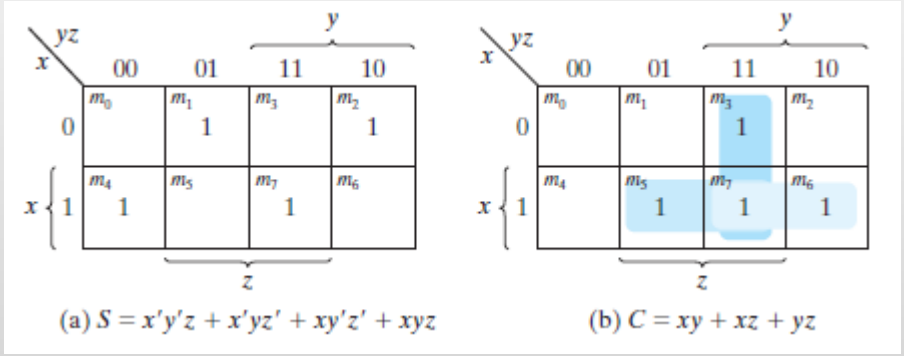
Full Adder

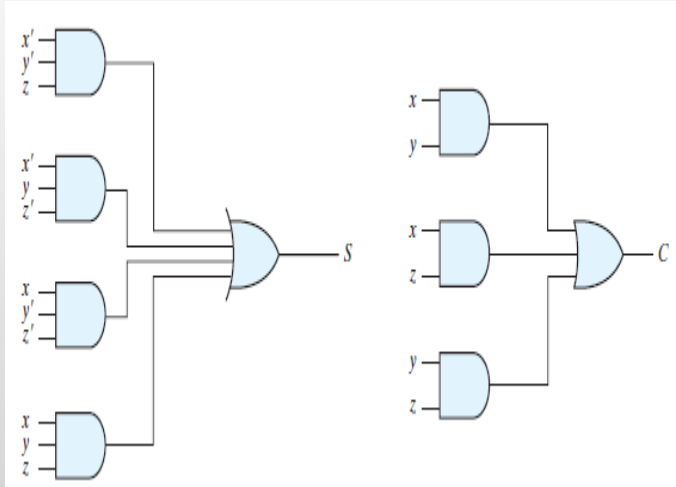
- Addition of n-bit binary numbers requires the use of a full adder, and the process of addition proceeds on a bit-by-bit basis, right to left, beginning with the least significant bit. A full adder is a combinational circuit that forms the arithmetic sum of three bits. It consists of three inputs and two outputs.
- The simplified expressions are: S = x'y'z + x'yz' + xy'z' + xyz

  C = xy + xz + yz

Full Adder

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1





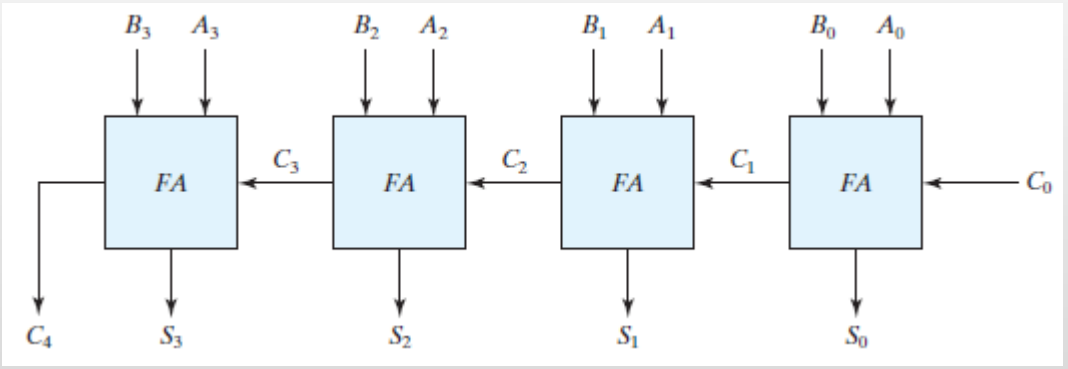
SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Binary Adder

- A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.
- Addition of n-bit numbers requires a chain of n full adders or a chain of one-half adder and n-1 full adders.
- Example, 4-bit Ripple Carry Adder

Subscript *i*:	**3**	**2**	**1**	**0**	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}



SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Carry Propagation

- As in any combinational circuit, the signal must propagate through the gates before the correct output sum is available in the output terminals. The total propagation time is equal to the propagation delay of a typical gate, times the number of gate levels in the circuit. The longest propagation delay time in an adder is the time it takes the carry to propagate through the full adders.
- The number of gate levels for the carry propagation can be found from the circuit of the full adder.
- The carry propagation time is an important attribute of the adder because it limits the speed with which two numbers are added.
- Since all other arithmetic operations are implemented by successive additions, the time consumed during the addition process is critical. An obvious solution for reducing the carry propagation delay time is to employ faster gates with reduced delays. However, physical circuits have a limit to their capability.
- Another solution is to increase the complexity of the equipment in such a way that the carry delay time is reduced. There are several techniques for reducing the carry propagation time in a parallel adder. The most widely used technique employs the principle of ***carry lookahead logic***.

Consider the circuit of the full adder with new binary variables:

$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

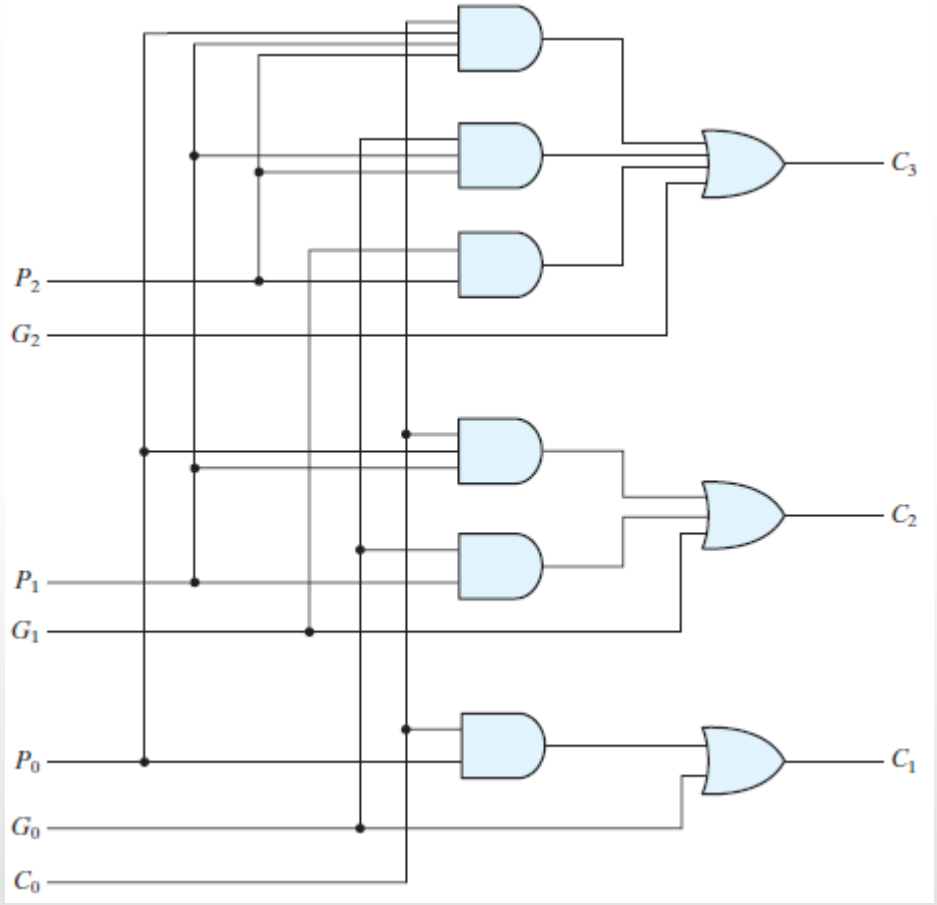
the output sum and carry can respectively be expressed as

$$S_i = P_i \oplus C_i$$

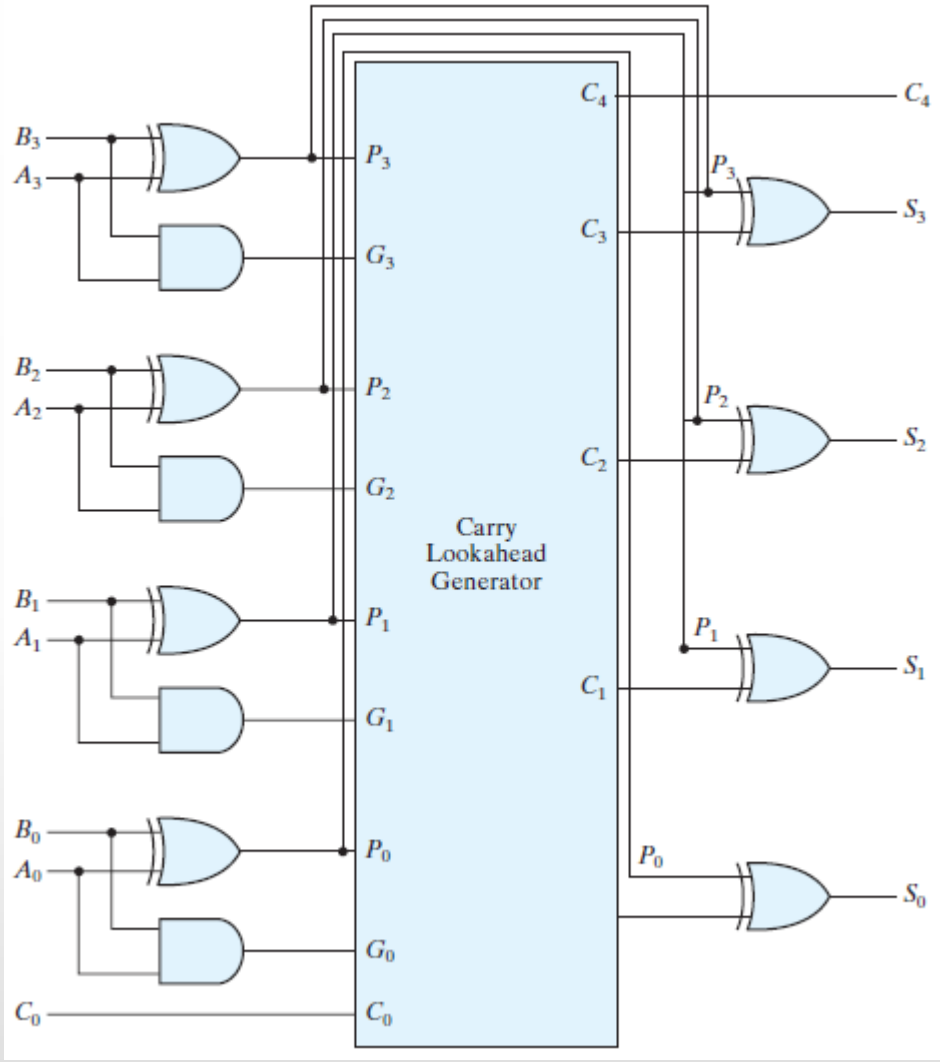
$$C_{i+1} = G_i + P_i C_i$$

G_i is called a *carry generate*, and it produces a carry of 1 when both A_i and B_i are 1, regardless of the input carry C_i. P_i is called a *carry propagate*, because it determines whether a carry into stage i will propagate into stage i + 1 (i.e., whether an assertion of C_i will propagate to an assertion of C_{i+1}).

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1



Logic diagram of carry lookahead generator



Four-bit adder with carry lookahead

KARNAUGH MAP (K-MAP)

- Karnaugh maps are used to simplify a Boolean function without using Boolean algebra theorems.
- A K-map is also another way to represent the truth table of a function.
- K-maps are made of cells where each cell represents a minterm. Cells marked with a one will be the minterms used for the sum of the minterms representation of a function.
- Conversely, cells marked with a zero will be used for the product of the maxterms representation.
- **Adjacent Cell**

  Two cells are adjacent if they differ on only one variable. The cells X'Y' and X'Y are adjacent because their only difference is Y' and Y. Adjacent cells can be combined in order to simplify a K-map's function.

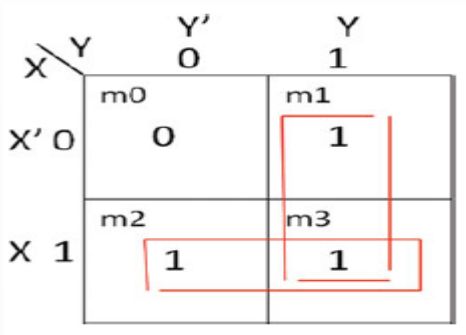
SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Simplify the following function:

$$F(X,Y)=X'Y+XY'+XY$$

or

$$F(X,Y) = m1+m2+m3$$



The cells m2 and m3 are adjacent, so they can be combined. Likewise, the cells m1 and m3 can be combined. By reading the map, you will have the simplified function.

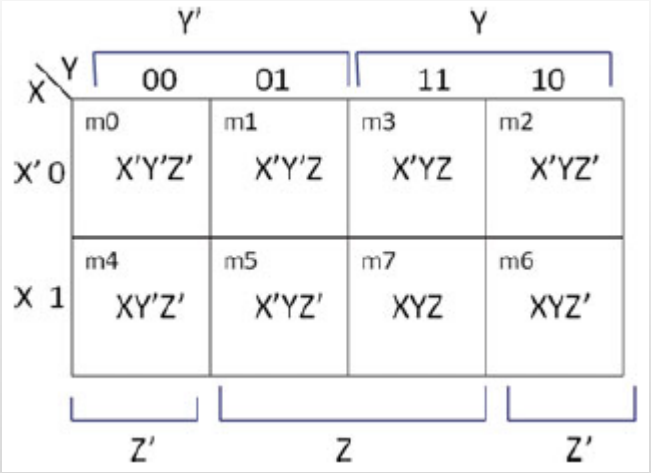
Cells m2 and m3 are the entire row X, and cells m1 and m3 are the entire column Y, with the other cell being zero. Therefore,

$$F(X,Y) = X+Y$$

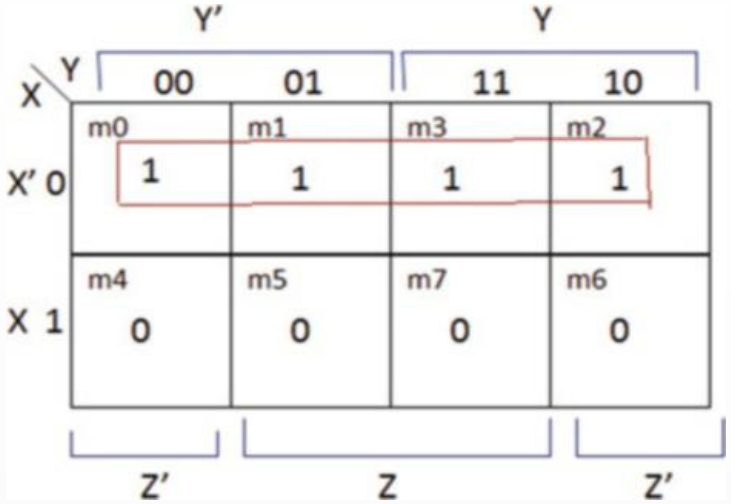
SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

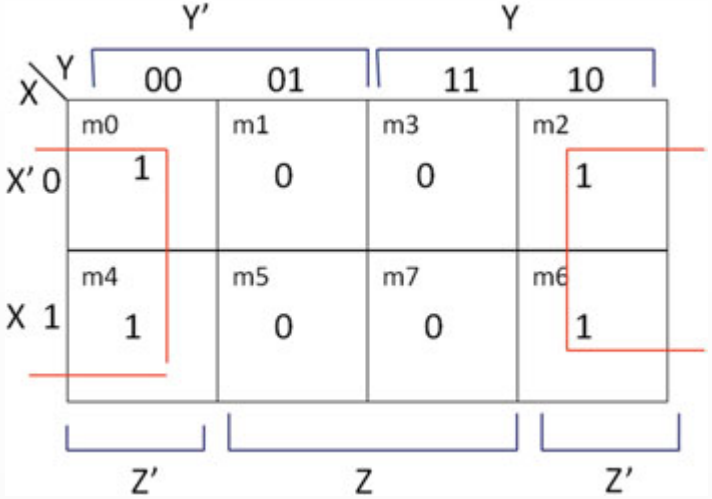
THREE-VARIABLE MAP

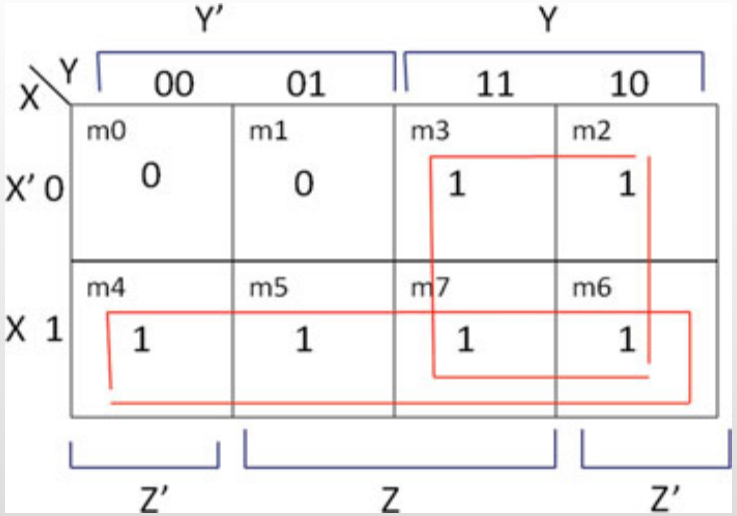
- A three-variable K-map contains eight cells, and each cell represents a minterm.
- Adjacent cells can be grouped together in a K-map; in a K-map it can combine 2 cells, 4 cells, 8 cells, and 16 cells.

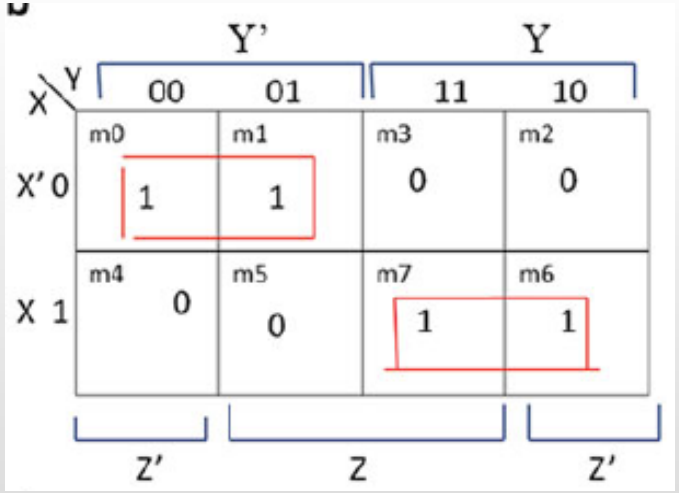


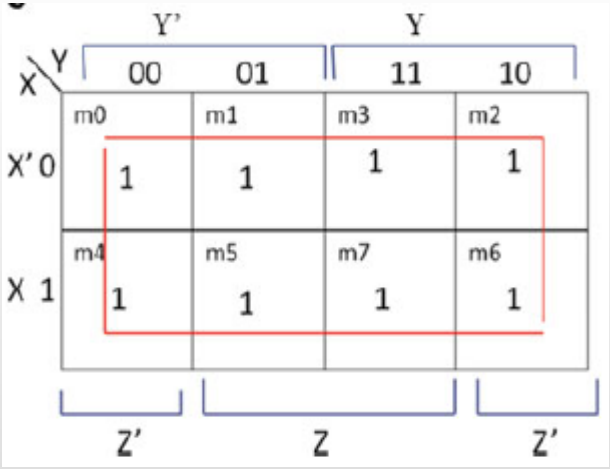
SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1











SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

(a) All adjacent cells in columns Z' and columns Y are one.

$$F(X,Y,Z) = Z' + Y'$$

(b) The cells in row X', columns Y' are adjacent, as are the cells in row X, columns Y.

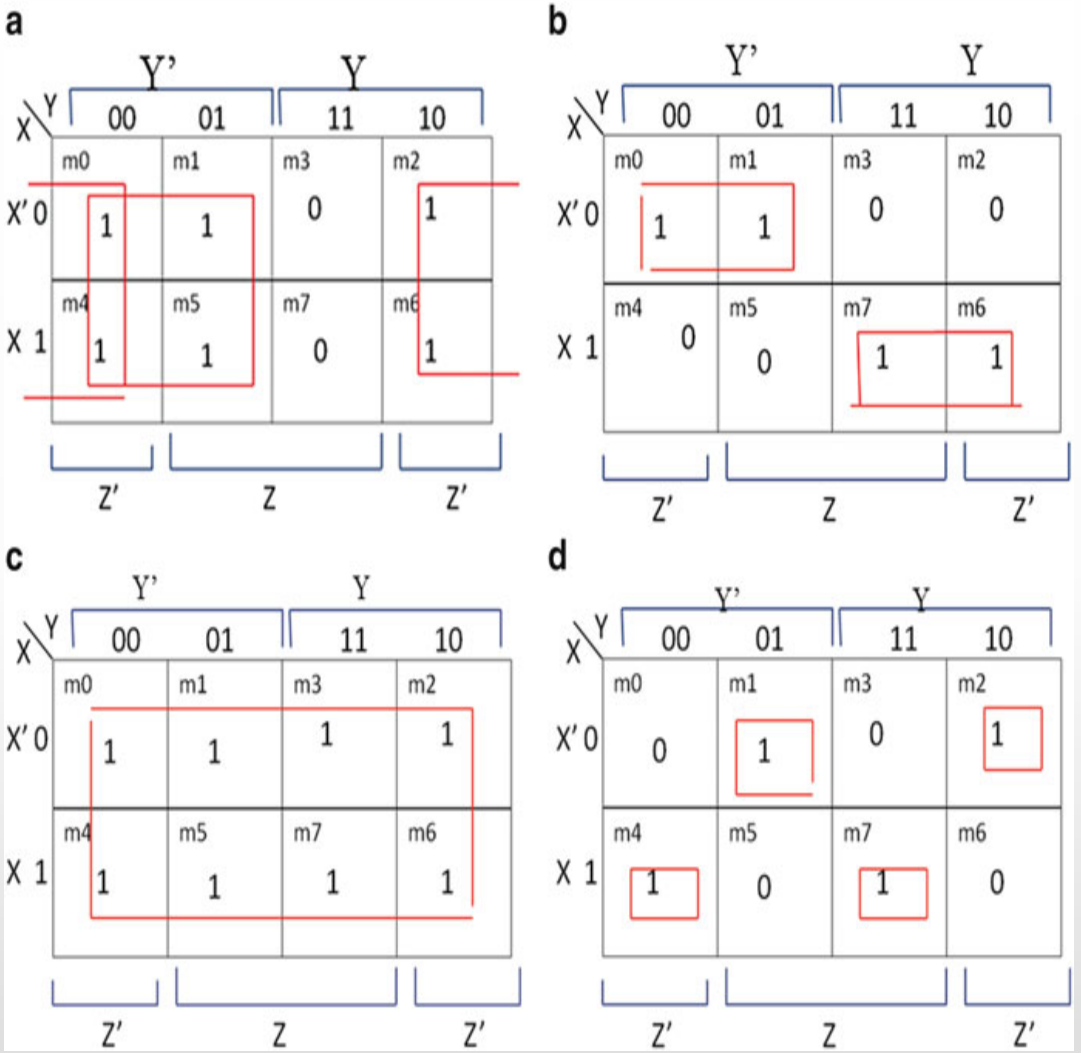
$$F(X,Y,Z) = X'Y' + XY$$

(c) All cells are one, so the function is always equal to one.

$$F(X,Y,Z) = 1$$

(d) Without adjacent cells to simplify the terms, the function equals the ones.

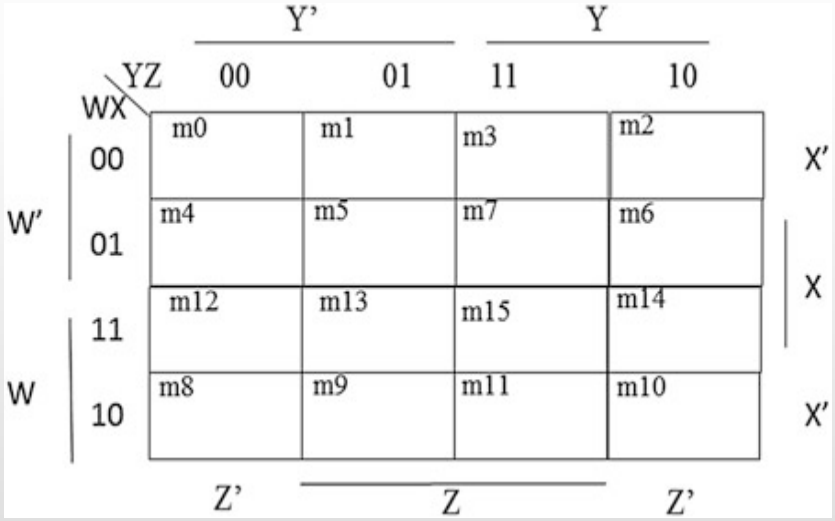
$$F(X,Y,Z) = X'Y'Z + X'YZ' + XY'Z' + XYZ$$



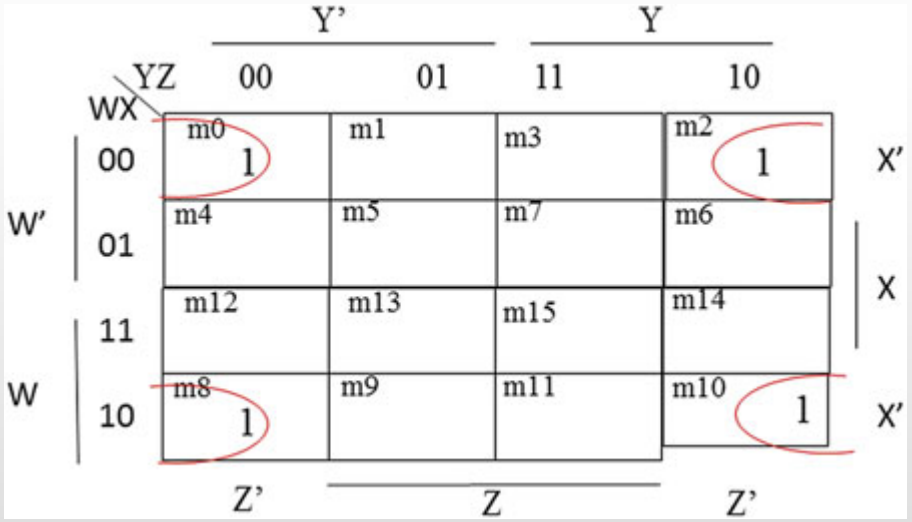
SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

FOUR-VARIABLE MAP

- Four-variable K-maps contain 16 cells.



Example, K-map for F(W,X,Y,Z) = m0 + m2 + m8 + m10



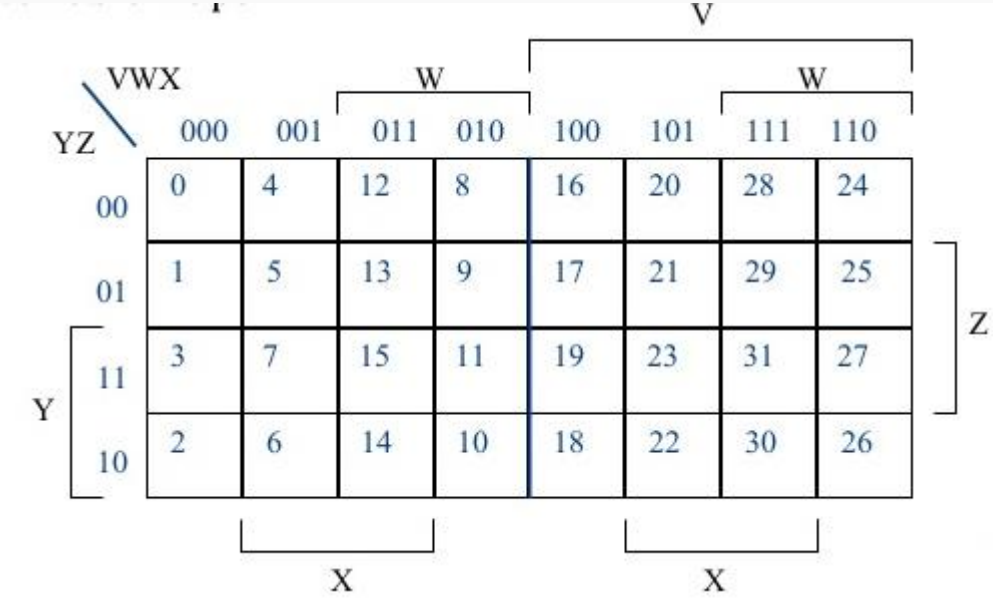
The simplified function is F(W,X,Y,Z) = X'Z'.

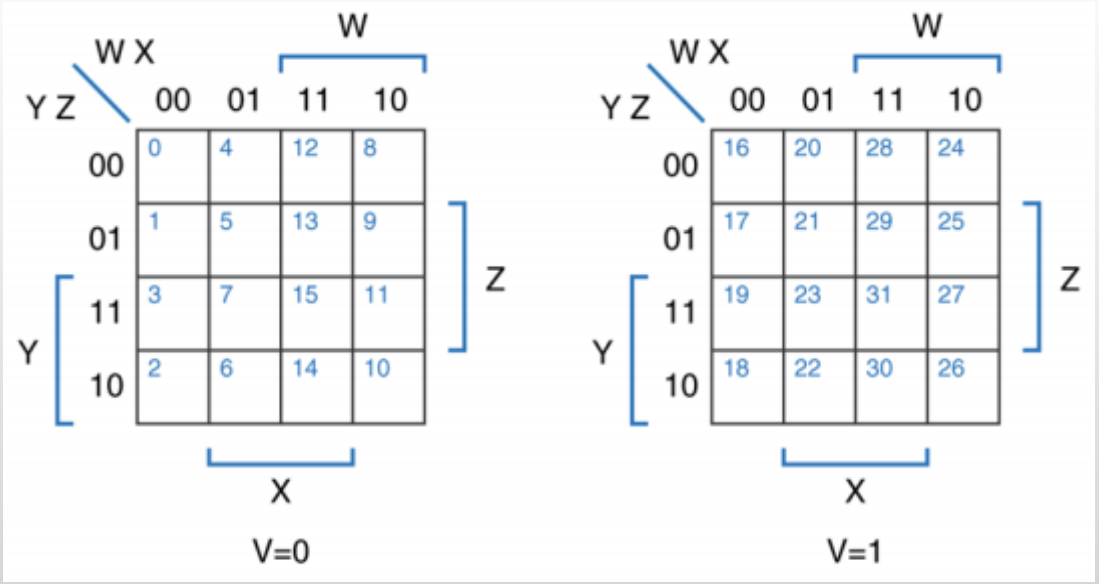
SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

FIVE-VARIABLE MAP

- A 5-variable Boolean function can have a maximum of 32 minterms. Hence, 5-variable K-Map contains 32 cells.
- A 5-variable K-map is formed using two connected 4-variable map.

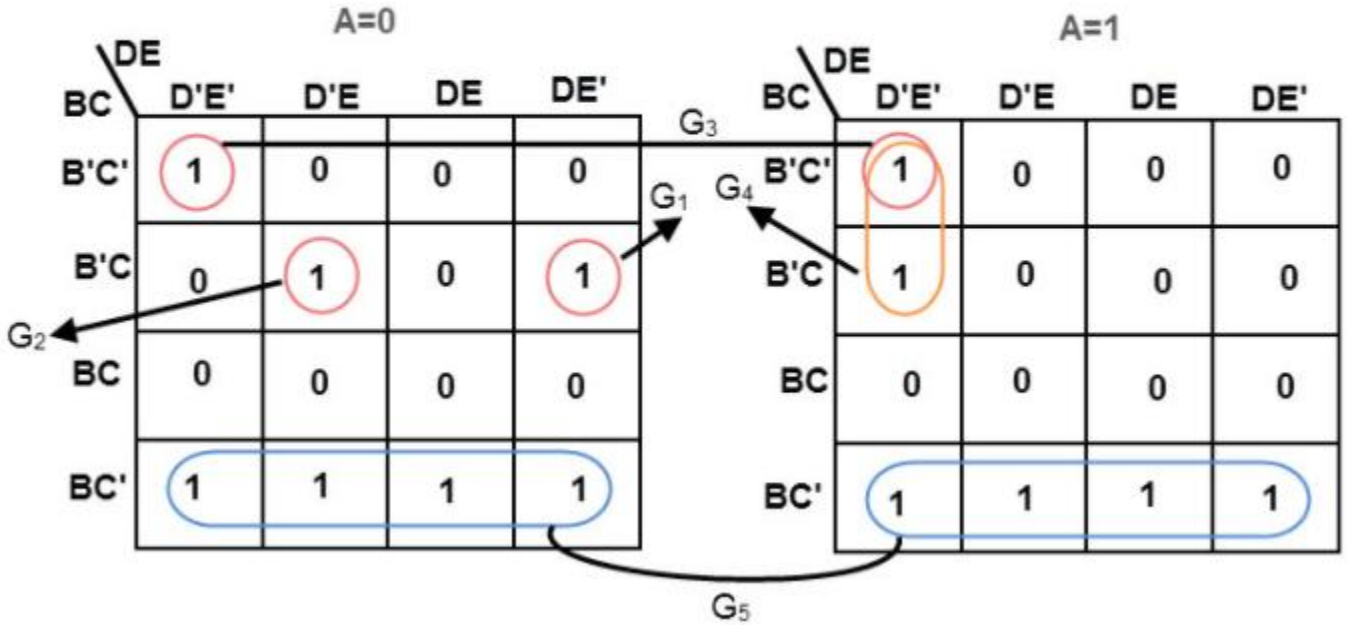
SOURCE: INTERNET





SOURCE: MORRIS MANO, THIRD EDITION

$f(A, B, C, D, E) = \sum m(0, 5, 6, 8, 9, 10, 11, 16, 20, 42, 25, 26, 27)$



SUM OF PRODUCTS (SOP) AND PRODUCT OF SUMS (POS)

- The sum of products of a function is its simplified sum of minterms while product of sums of a function is its simplified product of maxterms.

  Example,

  Consider function F1(X,Y,Z) = XY' + YZ' + XZ which is represented by the sum of products.

  Consider function F2(X,Y,Z) = (X + Y')(Y + Z')(X + Z) which is represented by the product of sums.

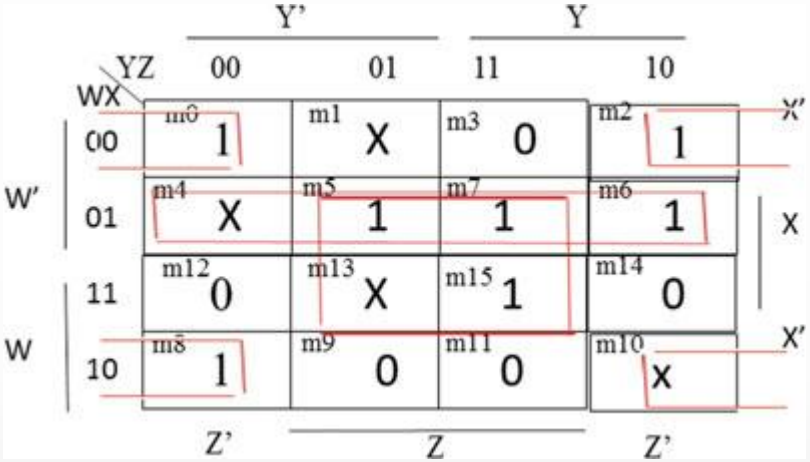
SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

DON'T CARE CONDITIONS

- In a truth table, if certain combinations of the input variables are impossible, they are considered don't care conditions.
- These conditions are where the output of the function does not matter.
- A truth table or K-map cell marked with a "X" or "d" is a don't care term, and output will not be affected whether it is a one or zero.
- The don't care can be used to expand the adjacency of cells in a K-map to further simplify a function, since their output does not matter.
- Since a don't care can output either a zero or one, we can assume it is a one in order to expand a grouping of adjacent cells.

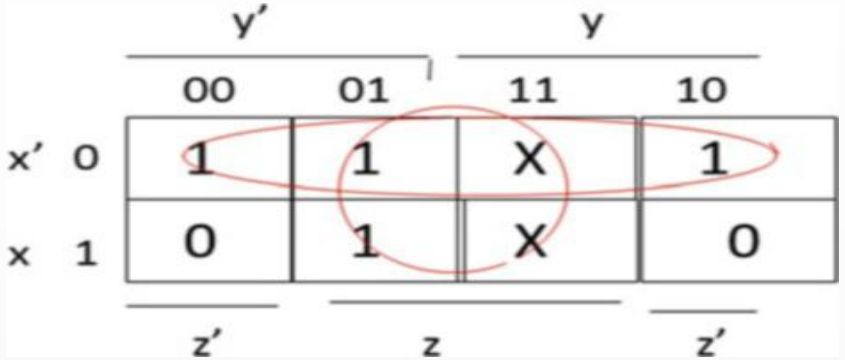
SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1



K-map with don't care minterms

F(W,X,Y,Z) = XZ + X'Z' + XW '



K-map for F(X,Y,Z) = m0 + m1 + m2 + m5 and D(X,Y,Z) = m3 + m7

Simplified: F(X,Y,Z) = X'+ Z.

When minterms for function F are don't care terms, the don't care function D is equal to the sum of the don't care minterm(s).

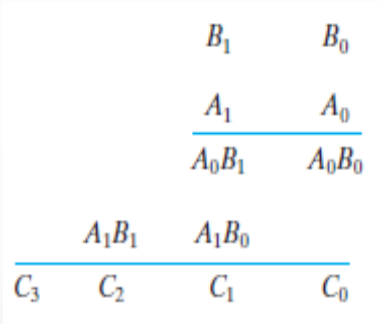
SIGNED OPERAND MULTIPLICATION

- Multiplication of binary numbers is performed in the same way as multiplication of decimal numbers.
- The multiplicand is multiplied by each bit of the multiplier, starting from the least significant bit. Each such multiplication forms a partial product.
- Successive partial products are shifted one position to the left. The final product is obtained from the sum of the partial products.

For instance, the multiplicand bits are B1 and B0, the multiplier bits are A1 and A0, and the product is C3C2C1C0. The first partial product is formed by multiplying B1B0 by A0.

The multiplication of two bits such as A0 and B0 produces a 1 if both bits are 1; otherwise, it produces a 0. This is identical to an AND operation.

Therefore, the partial product can be implemented with AND gates. The second partial product is formed by multiplying B1B0 by A1 and shifting one position to the left.



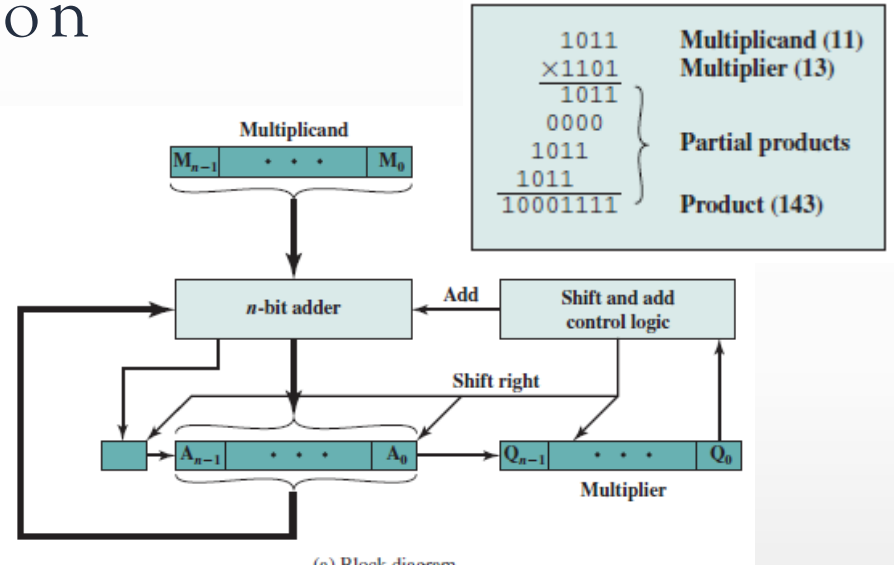
					1	1	1	0	1	1
				x	1	1	1	1	0	0
1	1	1	1	1	0	0	1	1		
1	1	1	1	0	0	1	1			
1	1	1	0	0	1	1				
1	1	0	0	1	1					
.	.	.	.	.	0	1	0	1	0	0

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

SOURCE: MORRIS MANO, THIRD EDITION

Twos Complement Multiplication

- If the two numbers are considered to be unsigned integers, then suppose we multiplied 11 (1011) by 13 (1101) to get 143 (10001111). If we interpret these as twos complement numbers, we have –5 (1011) times –3 (1101) equals –113 (10001111). Unfortunately, this simple scheme will not work for multiplication.
- Straightforward multiplication will not work if both the multiplicand and multiplier are negative. In fact, it will not work if either the multiplicand or the multiplier is negative.
- From the block diagram, the multiplier and multiplicand are loaded into two registers (Q and M). A third register, the A register, is also needed and is initially set to 0. There is also a 1-bit C register, initialized to 0, which holds a potential carry bit resulting from addition.



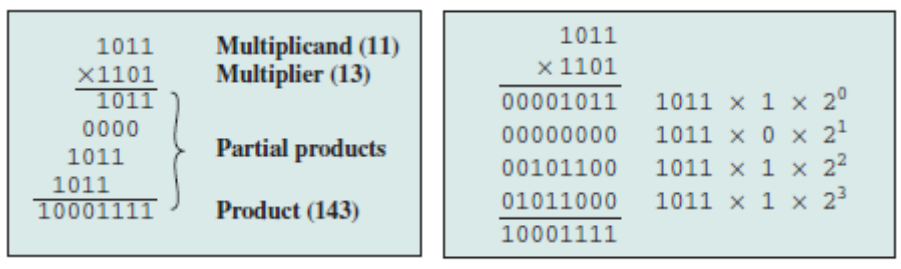
(a) Block diagram

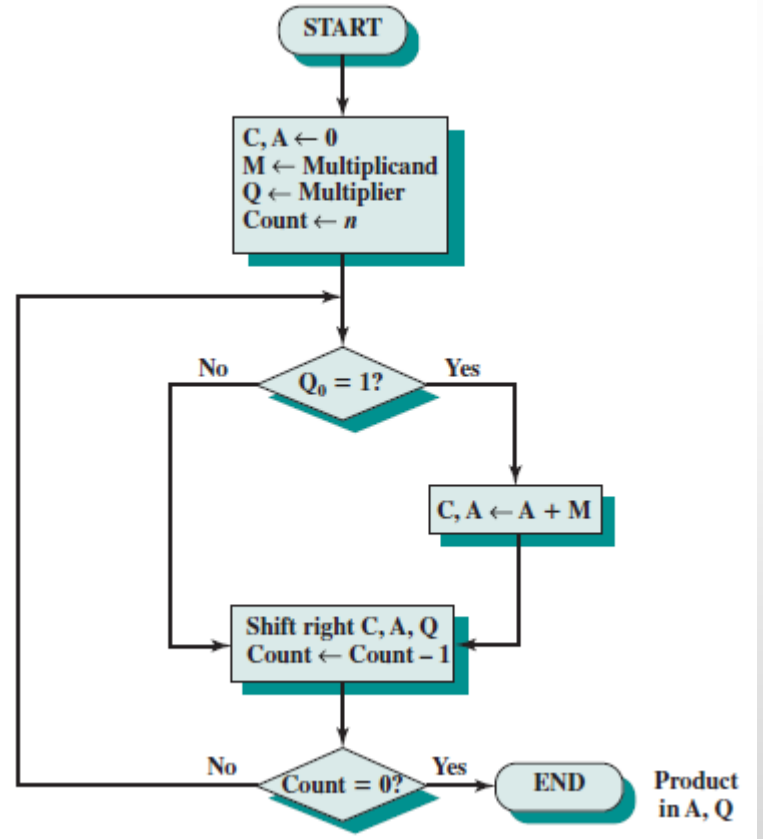
C	A	Q	M		
0	0000	1101	1011	Initial values	
0	1011	1101	1011	Add	First cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second cycle
0	1101	1111	1011	Add	Third cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth cycle
0	1000	1111	1011	Shift	

(b) Example from Figure 10.7 (product in A, Q)

SOURCE: MORRIS MANO, THIRD EDITION

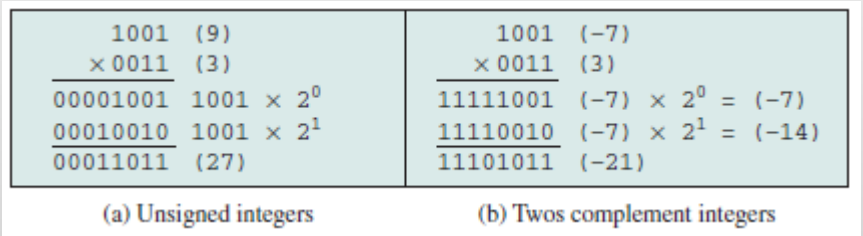
- Recall that any unsigned binary number can be expressed as a sum of powers of 2. Thus, $1101 = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 2^3 + 2^2 + 2^0$
- Further, the multiplication of a binary number by 2^n is accomplished by shifting that number to the left n bits.
- Hence, the multiplication process is re-casted to make the generation of partial products by multiplication explicit.
- The only difference is that it recognizes that the partial products should be viewed as 2n-bit numbers generated from the n-bit multiplicand.
- Thus, as an unsigned integer, the 4-bit multiplicand 1011 is stored in an 8-bit word as 00001011. Each partial product (other than that for 20) consists of this number shifted to the left, with the unoccupied positions on the right filled with zeros (e.g., a shift to the left of two places yields 00101100).





SOURCE: MORRIS MANO, THIRD EDITION

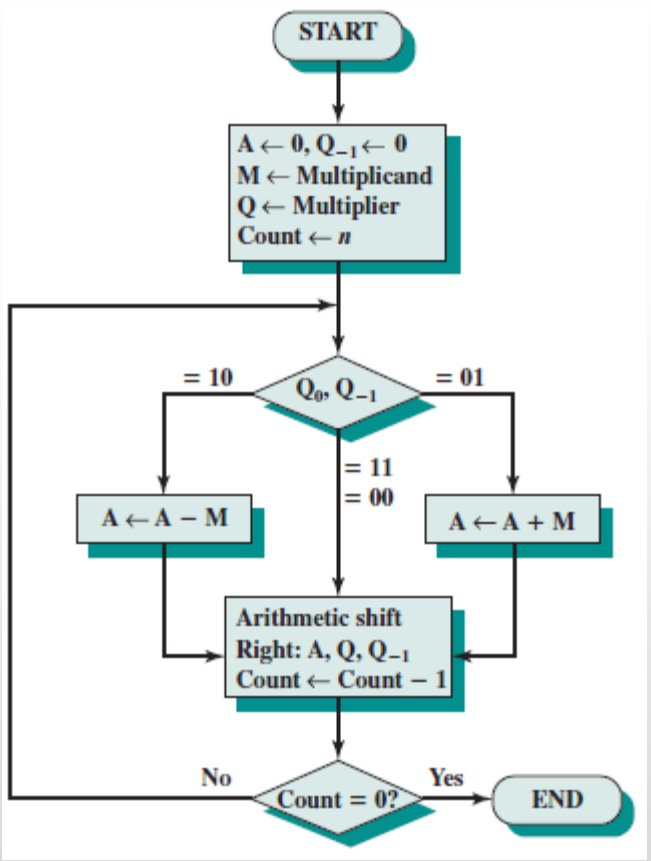
- The problem i.e. straightforward multiplication will not work if the multiplicand is negative is that each contribution of the negative multiplicand as a partial product must be a negative number on a 2n-bit field; the sign bits of the partial products must line up.
- Considering the same example, if these are treated as unsigned integers, the multiplication of 9 * 3 = 27 proceeds simply. However, if 1001 is interpreted as the twos complement value -7, then each partial product must be a negative twos complement number of 2n (8) bits.
- Note that this is accomplished by padding out each partial product to the left with binary 1s.
- Again, if the multiplier is negative, straightforward multiplication also will not work. The reason is that the bits of the multiplier no longer correspond to the shifts or multiplications that must take place.
- Example, the 4-bit decimal number –3 is written 1101 in twos complement. If we simply took partial products based on each bit position, we would have the following correspondence: $1101 = -(1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0) = -(2^3 + 2^2 + 2^0)$
- In fact, what is desired is $-(2^1 + 2^0)$. So this multiplier cannot be used directly in the manner we have been describing.



1001 (9) ×0011 (3) 00001001 1001 × 2^0 00010010 1001 × 2^1 00011011 (27)	1001 (−7) ×0011 (3) 11111001 (−7) × 2^0 = (−7) 11110010 (−7) × 2^1 = (−14) 11101011 (−21)
(a) Unsigned integers	(b) Twos complement integers

Booth's Algorithm

- To overcome the dilemma of binary multiplication, one of the solution would be to convert both multiplier and multiplicand to positive numbers, perform the multiplication, and then take the twos complement of the result if and only if the sign of the two original numbers differed. Implementers have preferred to use techniques that do not require this final transformation step.
- One of the most common of these is Booth's algorithm. This algorithm also has the benefit of speeding up the multiplication process, relative to a more straightforward approach.



SOURCE: MORRIS MANO, THIRD EDITION