Dec	Hex	Bin
0	0	00000000

ORG ; ZERO

Introduction To Computing

The x86 PC
assembly language, design, and interfacing
fifth edition
Prentice Hall

The x86 PC

assembly language, design, and interfacing

fifth edition

MUHAMMAD ALI MAZIDI
JANICE GILLISPIE MAZIDI
DANNY CAUSEY

OBJECTIVES
this chapter enables the student to:

- Convert any number from base 2, base 10, or base 16 to any of the other two bases.
- Add and subtract hex numbers.
- Add binary numbers.
- Represent any binary number in 2's complement.
- Represent an alphanumeric string in ASCII code.
- Describe logical operations AND, OR, NOT, XOR, NAND, NOR.
- Use logic gates to diagram simple circuits.

OBJECTIVES (*cont*)
this chapter enables the student to:

- Explain the difference between a bit, a nibble, a byte, and a word.
- Give precise mathematical definitions of the terms *kilobyte*, *megabyte*, *gigabyte*, and *terabyte*.
- Explain the difference between RAM and ROM, and describe their use.
- Describe the purpose of the major components of a computer system.
- List the three types of buses in a computer system and describe the purpose of each bus.

OBJECTIVES (*cont*)
this chapter enables the student to:

- Describe the role of the CPU in computer systems.
- List the major components of the CPU and describe the purpose of each.

0.1 Numbering and Coding Systems

- Speculation is that origin of the base 10 system is the fact that human beings have 10 fingers.
- Humans use base 10 (*decimal*) arithmetic, and computers use the base 2 (*binary*) system.
  - In base 10 are 10 distinct symbols, 0, 1, 2, ...9.
  - In base 2 there are only two symbols, the binary digits 0 & 1 - commonly referred to as *bits.*
- The binary system is used in computers because 0 & 1 represent the two voltage levels of *off* & *on*.

0.1 Numbering and Coding Systems converting from decimal to binary

- To convert decimal to binary, divide the decimal number by 2 repeatedly until the quotient is zero.
  - Remainders are written in reverse order to obtain the binary number.

Convert 25_{10} to binary.

Solution:

	Quotient	Remainder	
25/2 =	12	1	LSB (least significant bit)
12/2 =	6	0	
6/2 =	3	0	
3/2 =	1	1	
1/2 =	0	1	MSB (most significant bit)

Therefore, $25_{10} = 11001_2$.

0.1 Numbering and Coding Systems
converting from binary to decimal

- To convert from binary to decimal, it is important to understand the concept of weight associated with each digit position.
  - Each digit position of a number in base 2 has a weight associated with it.

$110101_2 =$				Decimal	Binary
1×2^0	=	1×1	=	1	1
0×2^1	=	0×2	=	0	00
1×2^2	=	1×4	=	4	100
0×2^3	=	0×8	=	0	0000
1×2^4	=	1×16	=	16	10000
1×2^5	=	1×32	=	32	100000
				53	110101

740683_{10}	=	
3×10^0	=	3
8×10^1	=	80
6×10^2	=	600
0×10^3	=	0000
4×10^4	=	40000
7×10^5	=	700000
		740683

0.1 Numbering and Coding Systems
converting from binary to decimal

- Knowing the weight of each bit in a binary number makes it simple to add them together to get the number's decimal equivalent.
  - Shown here in Example 0-2.

Convert 11001_2 to decimal.

Solution:

Weight:	16	8	4	2	1
Digits:	1	1	0	0	1
Sum:	16 +	8 +	0 +	0 +	$1 = 25_{10}$

0.1 Numbering and Coding Systems converting from decimal to binary

- Knowing the weight associated with each binary bit position allows one to convert a decimal number to binary directly instead of going through the process of repeated division.
  – This is Example 0-3.

Use the concept of weight to convert 39_{10} to binary.

Solution:

Weight:	32	16	8	4	2	1
	1	0	0	1	1	1
	32 +	0 +	0 +	4 +	2 +	1 = 39

Therefore, $39_{10} = 100111_2$.

0.1 Numbering and Coding Systems hexadecimal system

- Base 16, or the *hexadecimal* system is a convenient representation of binary numbers, using 16 digits.

The first ten digits, 0 to 9, are the same as in decimal.

For the remaining six digits, letters A, B, C, D, E, and F are used. subtraction.

It is easier for humans to express strings of 0s & 1s like 100010010110 as the hexadecimal number 896H.

Base 16 Number System

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

0.1 Numbering and Coding Systems
converting binary to hex

- To represent a binary number as its equivalent hexadecimal number, start from the right & group 4 bits at a time, replacing each 4-bit binary number with its hex equivalent shown.

Represent binary 100111110101 in hex.

Solution:

First the number is grouped into sets of 4 bits: 1001 1111 0101.

Then each group of 4 bits is replaced with its hex equivalent:

1001	1111	0101
9	F	5

Therefore, 100111110101_2 = 9F5 hexadecimal.

0.1 Numbering and Coding Systems
converting hex to binary

- To convert from hex to binary, each hex digit is replaced with its 4-bit binary equivalent.
  - Shown here in Example 0-5.

Convert hex 29B to binary.

Solution:

		2	9	B
29B	=	0010	1001	1011

Dropping the leading zeros gives 1010011011.

0.1 Numbering and Coding Systems
two ways of converting decimal to hex

- Convert directly from decimal to hex by repeated division, keeping track of the remainders.

Convert the following hexadecimal numbers to decimal.

(a) $6B2_{16} = 0110\ 1011\ 0010_2$

1024	512	256	128	64	32	16	8	4	2	1
1	1	0	1	0	1	1	0	0	1	0

$1024 + 512 + 128 + 32 + 16 + 2 = 1714_{10}$

(b) $9F2D_{16} = 1001\ 1111\ 0010\ 1101_2$

32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
1	0	0	1	1	1	1	1	0	0	1	0	1	1	0	1

$32768 + 4096 + 2048 + 1024 + 512 + 256 + 32 + 8 + 4 + 1 = 40{,}749_{10}$

- Convert to binary first, then convert to hex.

(a) Convert 45_{10} to hex.

32	16	8	4	2	1
1	0	1	1	0	1

First, convert to binary.
$32 + 8 + 4 + 1 = 45$

$45_{10} = 0010\ 1101_2 = 2D$ hex

(b) Convert 629_{10} to hex.

512	256	128	64	32	16	8	4	2	1
1	0	0	1	1	1	0	1	0	1

$629_{10} = (512 + 64 + 32 + 16 + 4 + 1) = 0010\ 0111\ 0101_2 = 275$ hex

(c) Convert 1714_{10} to hex.

1024	512	256	128	64	32	16	8	4	2	1
1	1	0	1	0	1	1	0	0	1	0

$1714_{10} = (1024 + 512 + 128 + 32 + 16 + 2) = 0110\ 1011\ 0010_2 = 6B2$ hex

counting in bases 10, 2 & 16

- Table 0-2 shows the relationship between all three bases in the sequence of numbers from 0 to 31 in decimal, with equivalent binary & hex numbers.

In each base, when one more is added to the highest digit, that digit becomes zero.

A 1 is carried to the next-highest digit position.

Decimal	Binary	Hex	Decimal	Binary	Hex
0	00000	0	16	10000	10
1	00001	1	17	10001	11
2	00010	2	18	10010	12
3	00011	3	19	10011	13
4	00100	4	20	10100	14
5	00101	5	21	10101	15
6	00110	6	22	10110	16
7	00111	7	23	10111	17
8	01000	8	24	11000	18
9	01001	9	25	11001	19
10	01010	A	26	11010	1A
11	01011	B	27	11011	1B
12	01100	C	28	11100	1C
13	01101	D	29	11101	1D
14	01110	E	30	11110	1E
15	01111	F	31	11111	1F

0.1 Numbering and Coding Systems
addition of binary and hex numbers

- Table 0-3 shows the addition of two bits.

A + B	Carry	Sum
0 + 0	0	0
0 + 1	0	1
1 + 0	0	1
1 + 1	1	0

- Example 0-8 shows addition of binary numbers.

Add the following binary numbers. Check against decimal equivalents.

Solution:	***Binary***	***Decimal***
	1101	13
+	1001	9
	10110	22

subtraction of binary & hex numbers

- All computers use the addition process to implement subtraction.
  - Computers have adder circuitry, but no separate circuitry for subtracters.
- Adders are used in conjunction with *2's complement* circuitry to perform subtraction.
  - To implement "*x* – *y*", the computer takes the 2's complement of *y* and adds it to *x*.

2’ s compliment of a binary number

- To get the 2’s complement of a binary number, invert all the bits and then add 1 to the result.

Take the 2’s complement of 100111

Solution:	10011101	binary number
	01100010	1’s complement
+	1	
	01100011	2’s complement

- Inverting the bits is simply a matter of changing all 0s to 1s and 1s to 0s.
  - This is called the *1’s complement.*

0.1 Numbering and Coding Systems
addition & subtraction of hex numbers

- To add, start with the least significant digit, & add the digits together.
  - If the result is less than 16, write that digit as the sum for that position.
  - If greater than 16, subtract 16 from it to get the digit and carry 1 to the next digit.

Perform hex addition: 23D9 + 94BE.

Solution:

	23D9	LSD:	9 + 14 = 23	23 – 16 = 7 with a carry
+	94BE		1 + 13 + 11 = 25	25 – 16 = 9 with a carry
	B897		1 + 3 + 4 = 8	
		MSD:	2 + 9 = B	

0.1 Numbering and Coding Systems
addition & subtraction of hex numbers

- In subtracting two hex numbers, if the second digit is greater than the first, borrow 16 from the preceding digit.

Perform hex addition: 23D9 + 94BE.

Solution:

	23D9	LSD:	9 + 14 = 23	23 – 16 = 7 with a carry
+	<u>94BE</u>		1 + 13 + 11 = 25	25 – 16 = 9 with a carry
	B897		1 + 3 + 4 = 8	
		MSD:	2 + 9 = B	

Mastery of these techniques is *essential*.

0.1 Numbering and Coding Systems
ASCII code

- Because all information in the computer must be represented by 0s & 1s, binary patterns must be assigned to letters and other characters.
- In the 1960s, a standard representation called ***ASCII*** was established.
  - Named for the **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange, pronounced *“Ask-E”.*
- It assigns binary patterns for numbers 0 to 9, and all English alphabet letters, upper- and lower-case.
  - Also many control codes & punctuation marks.

0.1 Numbering and Coding Systems
ASCII code

- ASCII code is used by most computers, so information can be shared among them.
  - ASCII uses a total of 7 bits to represent each code.

Often, a zero is placed in the most-significant bit position to make it an 8-bit code.

A complete list of ASCII codes is given in Appendix F.

Hex	*Symbol*	*Hex*	*Symbol*
41	A	61	a
42	B	62	b
43	C	63	c
44	D	64	d
...	...	...	...
59	Y	79	y
5A	Z	7A	z

Figure 0-1 Selected ASCII Codes

0.1 Numbering and Coding Systems
ASCII code

- ASCII is standard for keyboards and provides a standard for printing & displaying characters by output devices.
  - The pattern of ASCII was designed to allow for easy manipulation of ASCII data.
- Digits 0 - 9 are represented by ASCII codes 30 - 39.
  - This enables a program to convert ASCII to decimal by masking off (changing to zero) the “3” in the upper nibble.
- Conversion between uppercase and lowercase is as simple as changing bit 5 of the ASCII code.

0.1 Numbering and Coding Systems
selected ASCII codes

Hex	Ch	Hex	Ch	Hex	Ch	Hex	Ch	Hex	Ch	Hex	Ch	Hex	Ch	Hex	Ch
00		10	►	20		30	0	40	@	50	P	60	`	70	p
01	☺	11	◄	21	!	31	1	41	A	51	Q	61	a	71	q
02	☻	12	↕	22	"	32	2	42	B	52	R	62	b	72	r
03	♥	13	‼	23	#	33	3	43	C	53	S	63	c	73	s
04	♦	14	¶	24	$	34	4	44	D	54	T	64	d	74	t
05	♣	15	§	25	%	35	5	45	E	55	U	65	e	75	u
06	♠	16	▬	26	&	36	6	46	F	56	V	66	f	76	v
07	•	17	↨	27	'	37	7	47	G	57	W	67	g	77	w
08	◘	18	↑	28	(	38	8	48	H	58	X	68	h	78	x
09	○	19	↓	29	)	39	9	49	I	59	Y	69	i	79	y
0A	◙	1A	→	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	♂	1B	←	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	♀	1C	∟	2C	,	3C	<	4C	L	5C	\	6C	l	7C	¦
0D	♪	1D	↔	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	♫	1E	▲	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	☼	1F	▼	2F	/	3F	?	4F	O	5F	_	6F	o	7F	⌂

0.2 Digital Primer
binary logic

- Signals in digital electronics have two distinct voltage levels.
  - A system may define 0 V as logic 0 and +5 V as logic 1.
- A valid digital signal in this example should be within either of the two shaded areas.
  - Fig. 0-2 shows this with built-in tolerances for variation in voltage.

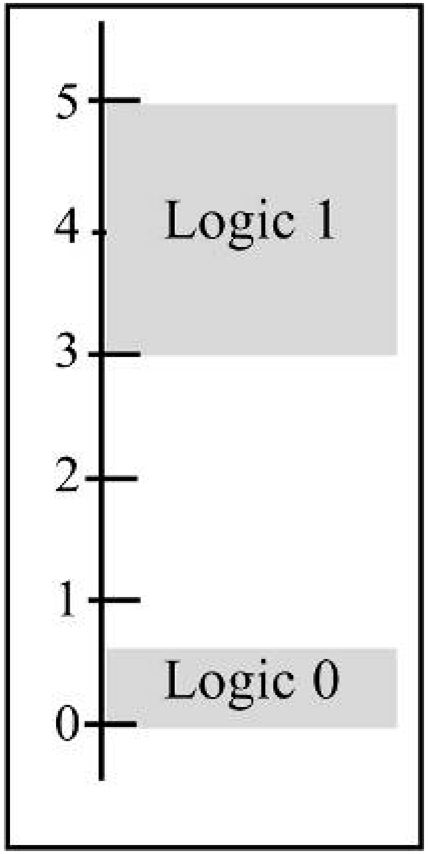


Figure 0-2 Binary signals

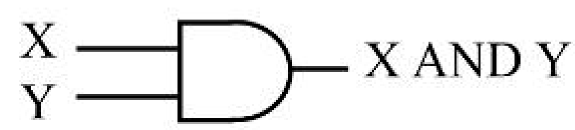
0.2 Digital Primer
logic gates

- Binary logic gates are simple circuits that take one or more input signals & send out one output signal.

- ***AND gate*** **-** takes two or more inputs and performs a logic AND on them.

Logical AND Function

Inputs	**Output**
X Y	**X AND Y**
0 0	0
0 1	0
1 0	0
1 1	1



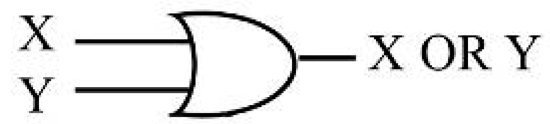
0.2 Digital Primer
logic gates

- Binary logic gates are simple circuits that take one or more input signals & send out one output signal.

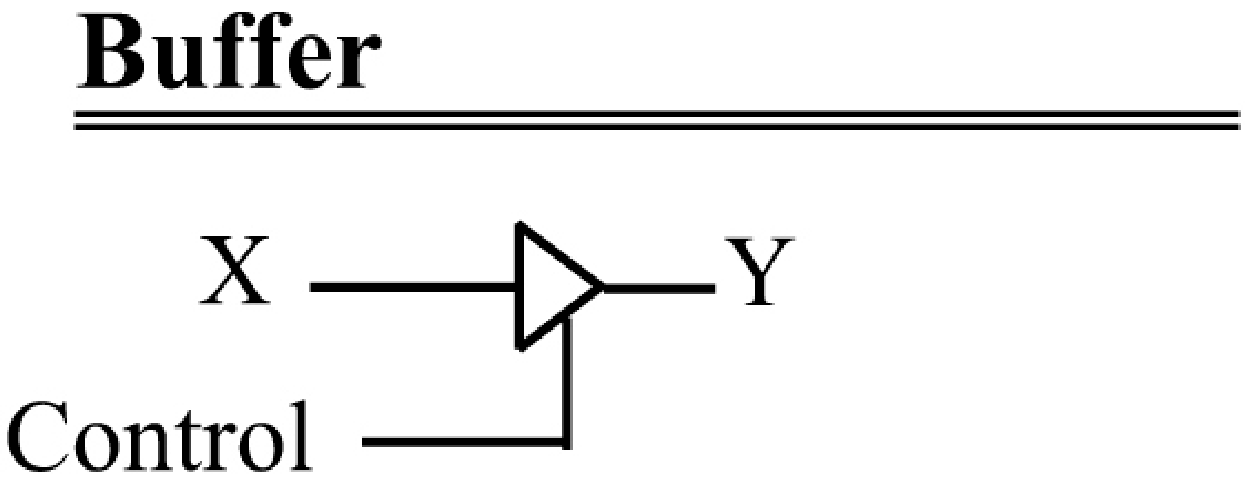
- ***OR gate -*** will output a 1 if one or more inputs is 1.
  - If all inputs are 0, then, and only then, will output be 0.

Logical OR Function

Inputs	Output
X Y	**X OR Y**
0 0	0
0 1	1
1 0	1
1 1	1



- ***Tri-state buffer -*** does not change the logic level of the input.
  - It is used to isolate or amplify the signal.



logic gates

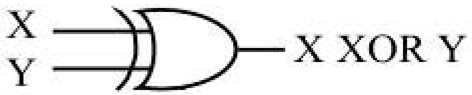
- ***Inverter -*** also called NOT. Outputs value opposite to that input to the gate.
  - A 1 input will give a 0 output.
  - 0 input will give a 1 output.

Logical Inverter

Input	**Output**
X	**NOT X**
0	1
1	0

Logical XOR Function

Inputs	**Output**
X Y	**X XOR Y**
0 0	0
0 1	1
1 0	1
1 1	0



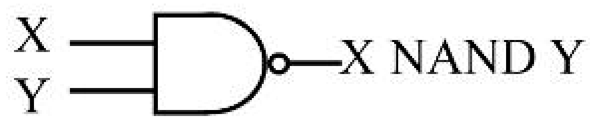
- ***XOR gate -*** performs an exclusive-OR operation on the inputs. gate.
  - Exclusive-OR produces a 1 output if one (but only one) input is 1.

0.2 Digital Primer
logic gates

- ***NAND -*** functions like an AND gate with an inverter on the output.

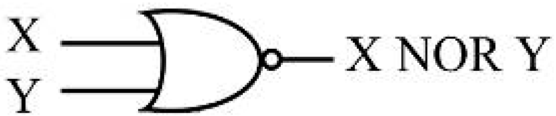
Logical NAND Function

Inputs X Y	Output X NAND Y
0 0	1
0 1	1
1 0	1
1 1	0



Logical NOR Function

Inputs X Y	Output X NOR Y
0 0	1
0 1	0
1 0	0
1 1	0



- ***NOR gates -*** function like an OR gate with an inverter on the output.

0.2 Digital Primer
logic design using gates

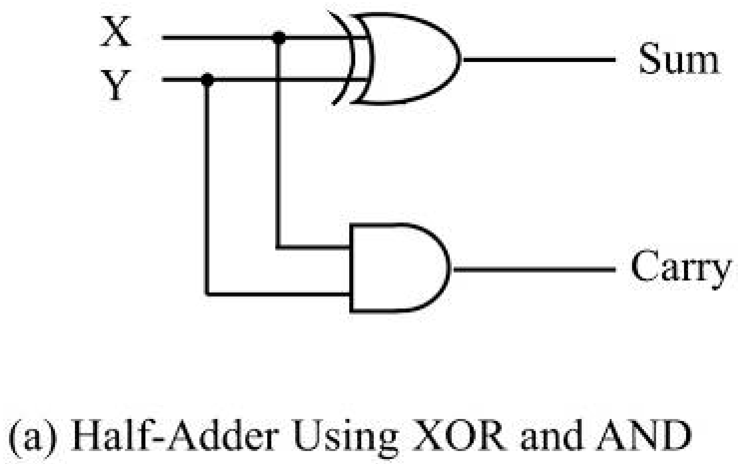
- A simple logic design to add two binary digits.
  - If we add two binary digits there are four possible.

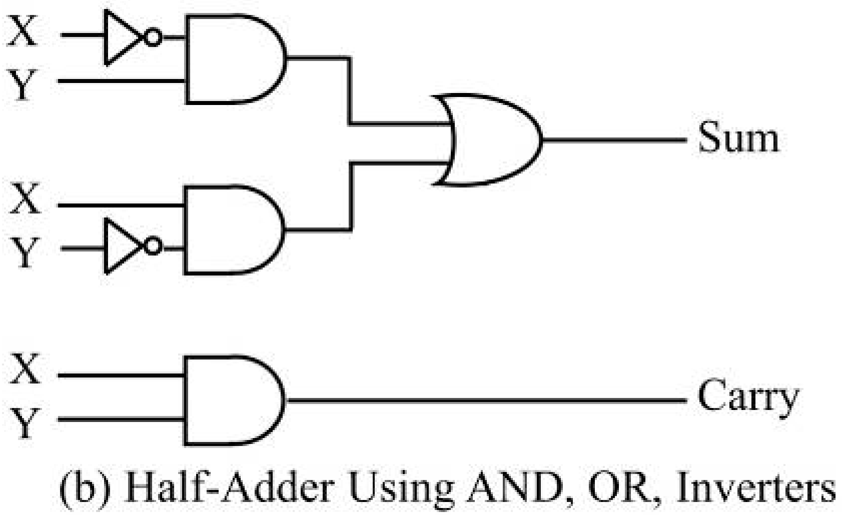
The sum column matches the output for the XOR function, and the carry column matches the output for the AND function.

	Carry	Sum
$0 + 0 =$	0	0
$0 + 1 =$	0	1
$1 + 0 =$	0	1
$1 + 1 =$	1	0

0.2 Digital Primer
logic design using gates

- Fig. 0-3(a) shows a simple adder implemented with XOR and AND gates.





- Figure 0-3(b) shows the same logic circuit implemented with AND and OR gates and inverters.

Figure 0-3 Two Implementations of a Half-Adder

- Fig. 0-4 shows a block diagram of a half-adder
  - Two half-adders can be combined to form an adder that can add three input digits.
  - Called a full-adder.

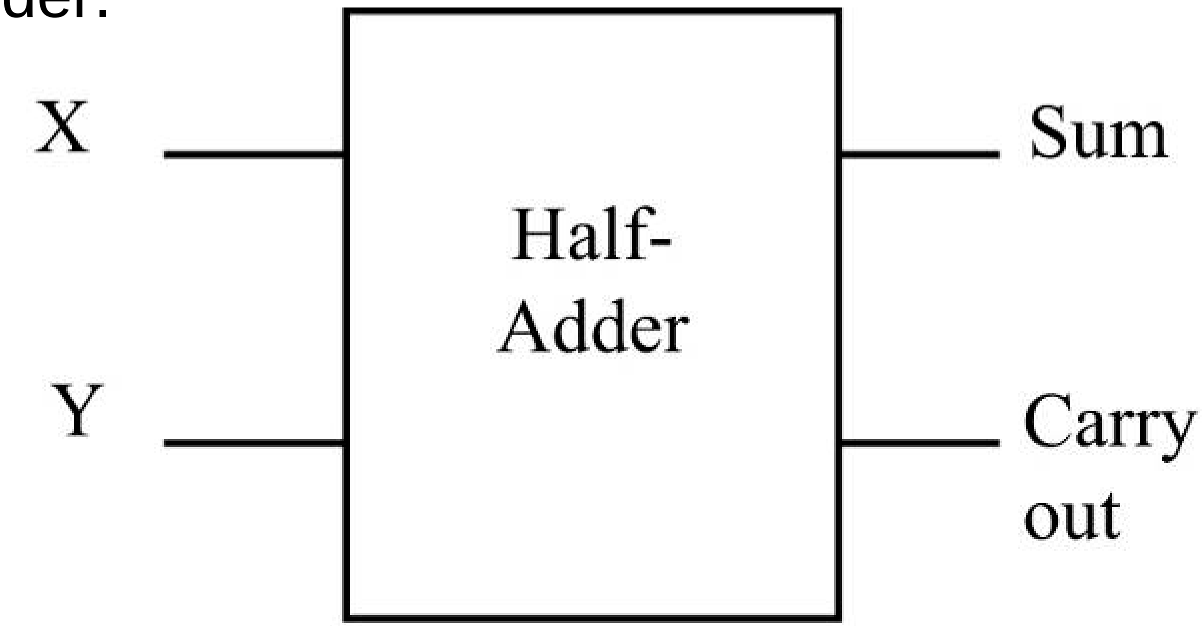


Figure 0-4 Block Diagram of a Half-Adder

0.2 Digital Primer
logic design using gates

- Fig. 0-5 shows the logic diagram of a full-adder, along with a block diagram that masks the details of the circuit.

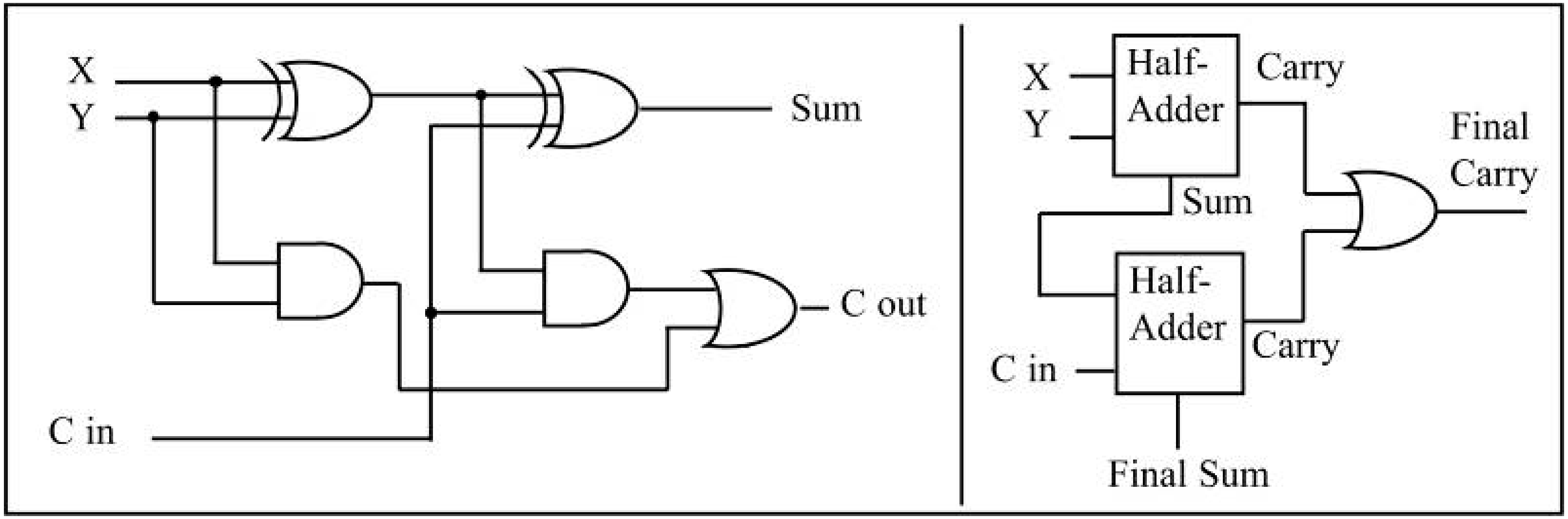


Figure 0-5 Full-Adder Built from a Half-Adder

- Fig. 0-6 shows a 3-bit adder using three full-adders.

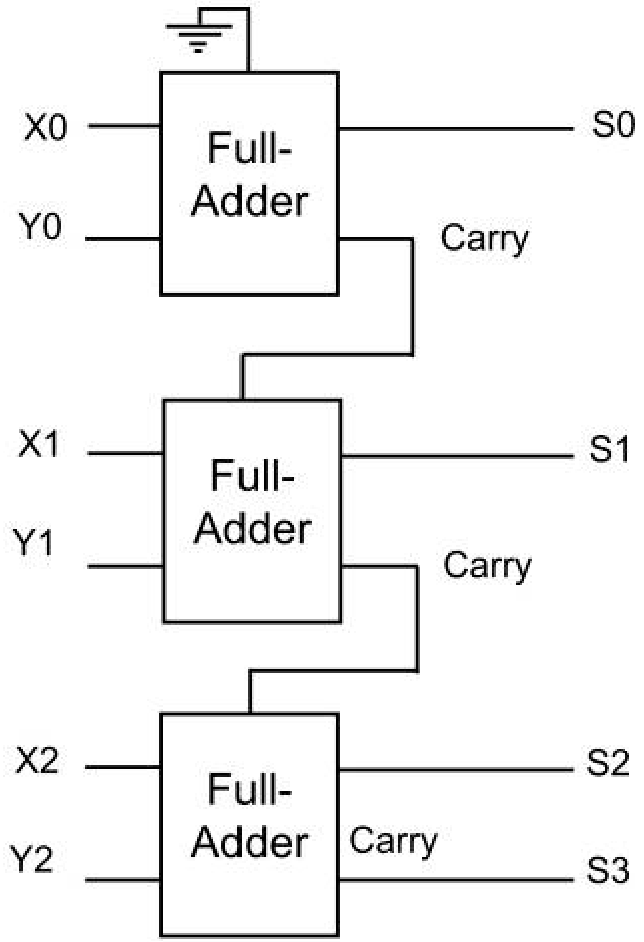


Figure 0-6 3-Bit Adder Using Three Full-Adders

0.2 Digital Primer
decoders

- Another application of logic gates is the decoder.
  - Widely used for address decoding in computer design.
  - Figure 0-7 shows decoders for 9 (1001 binary) and 5 (0101) using inverters and AND gates.

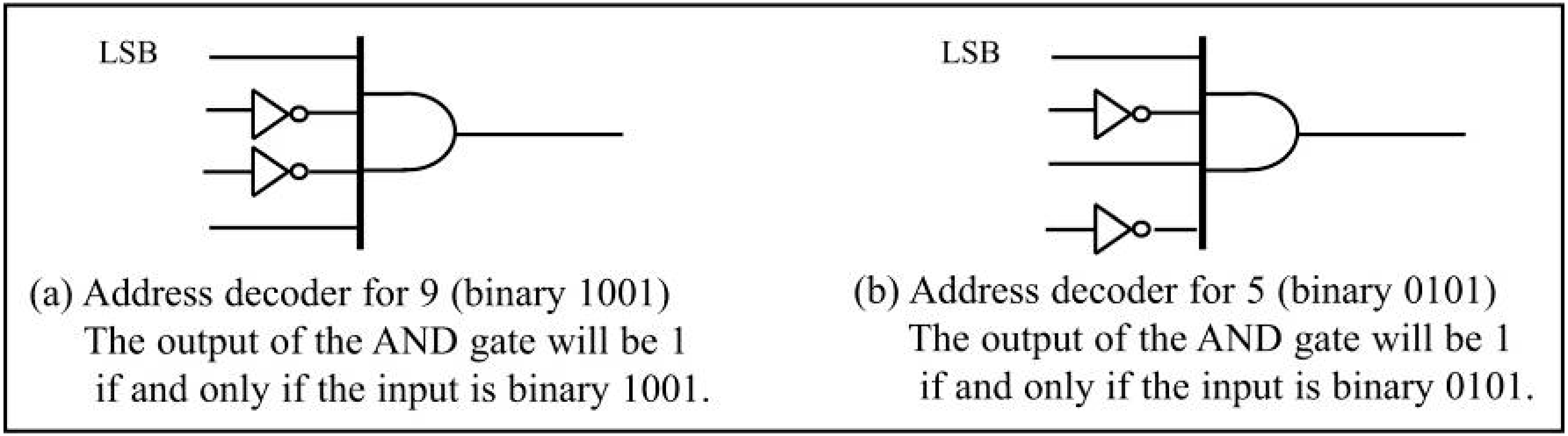
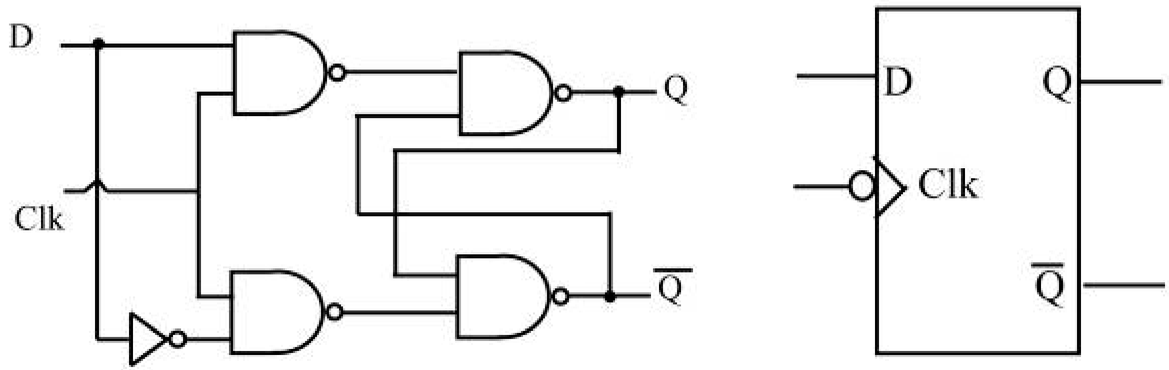


Figure 0-7 Address Decoders

0.2 Digital Primer
flip-flops

- A widely used component in digital systems
  - Frequently used to store data.
- D flip-flop is widely used widely used to latch data.
  - D-FF grabs data at the input as the clock is activated.
  - D-FF holds the data as long as the power is on.



(a) Circuit diagram (b) Block diagram

Clk	D	Q
No	x	no change
↓	0	0
↓	1	1

x = don't care

(c) Truth table

Figure 0-8 D Flip-Flops

0.3 Inside the Computer
terminology

- A *bit* is a binary digit that can have the value 0 or 1.

Bit				0
Nibble				0000
Byte			0000	0000
Word	0000	0000	0000	0000

- A *nibble* is 4 bits.
- A *byte* is defined as 8 bits.
- A *word* is two bytes, or 16 bits.

0.3 Inside the Computer terminology

- A *kilobyte* is 2^{10} bytes, which is 1,024 bytes.
  - The abbreviation K is often used to represent kilobytes.
- A *megabyte*, or *meg*, is 2^{20} bytes.
  - A little over 1 million bytes; exactly 1,048,576 bytes.
- A *gigabyte* is 2^{30} bytes (over 1 billion).
- A *terabyte* is 2^{40} bytes (over 1 trillion).

Power of 2
$2^{10} = 1024 = 1K$
$2^{20} = 1024K = 1M$
$2^{30} = 1024M = 1G$
$2^{40} = 1024G = 1T$

0.3 Inside the Computer
two common memory types

- ***RAM*** **-** which stands for "random access memory" (sometimes called *read/write memory*).
  - Used for temporary storage of programs while running.
    - Data is lost when the computer is turned off.
    - RAM is sometimes called *volatile memory.*
- ***ROM*** **-** stands for "read-only memory".
  - Contains programs and information essential to the operation of the computer.
    - Information in ROM is permanent, cannot be changed by the user, and is not lost when the power is turned off.
    - ROM is called *nonvolatile memory.*

0.3 Inside the Computer
internal organization of computers

- Internal workings of every computer can be broken down into three parts:

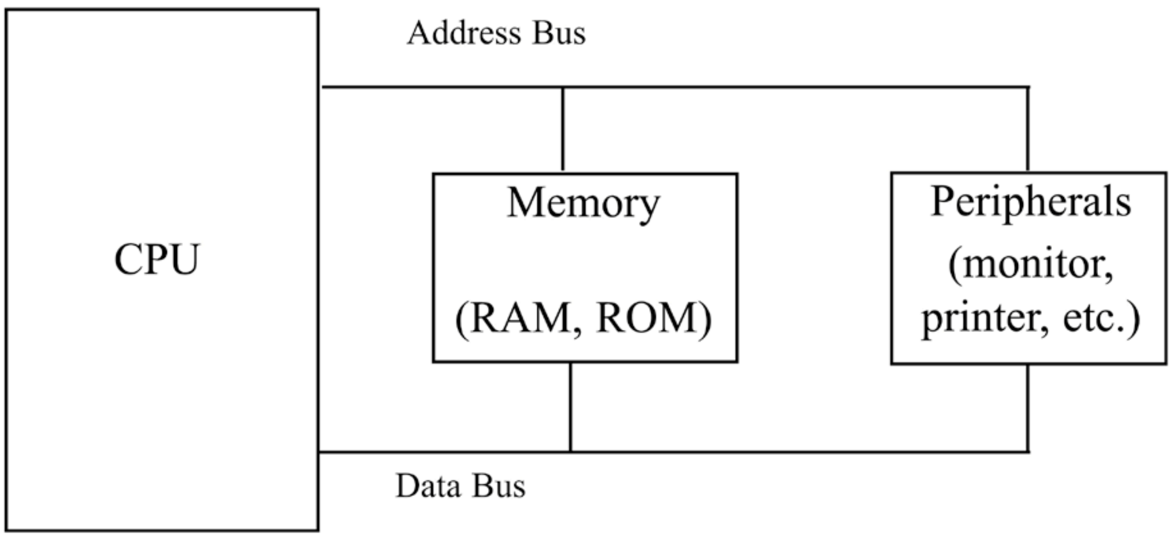


Figure 0-9
Inside the
Computer

0.3 Inside the Computer
internal organization of computers

- Internal workings of every computer can be broken down into three parts:
  - **CPU** (central processing unit).

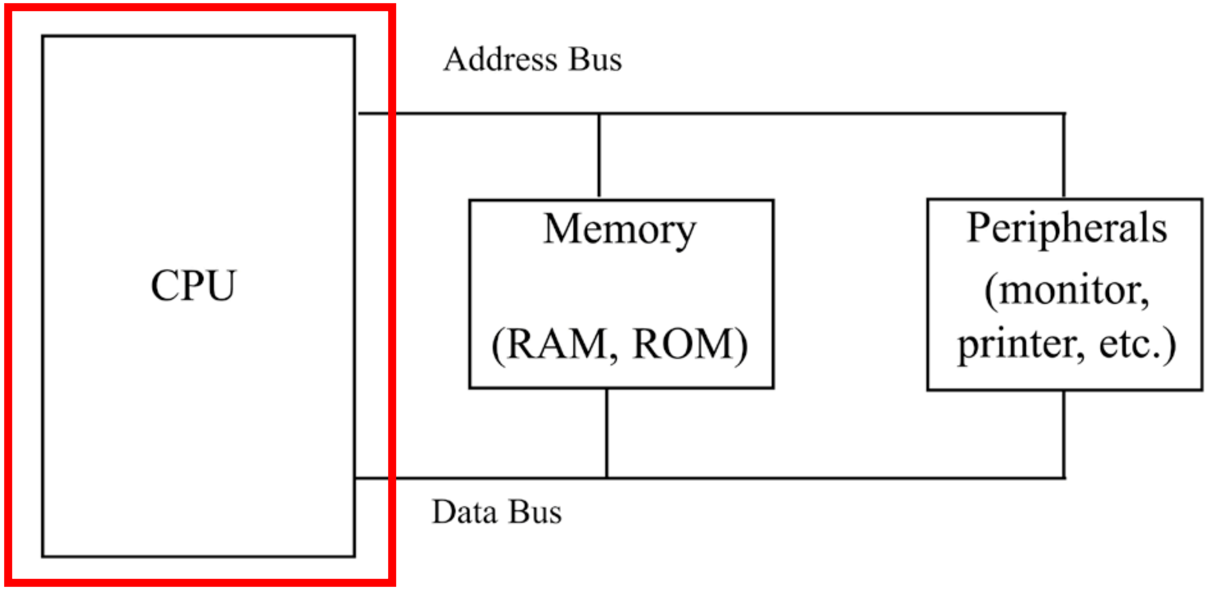


Figure 0-9 Inside the Computer

0.3 Inside the Computer
internal organization of computers

- Internal workings of every computer can be broken down into three parts:
  - **CPU** (central processing unit).
  - **Memory.**

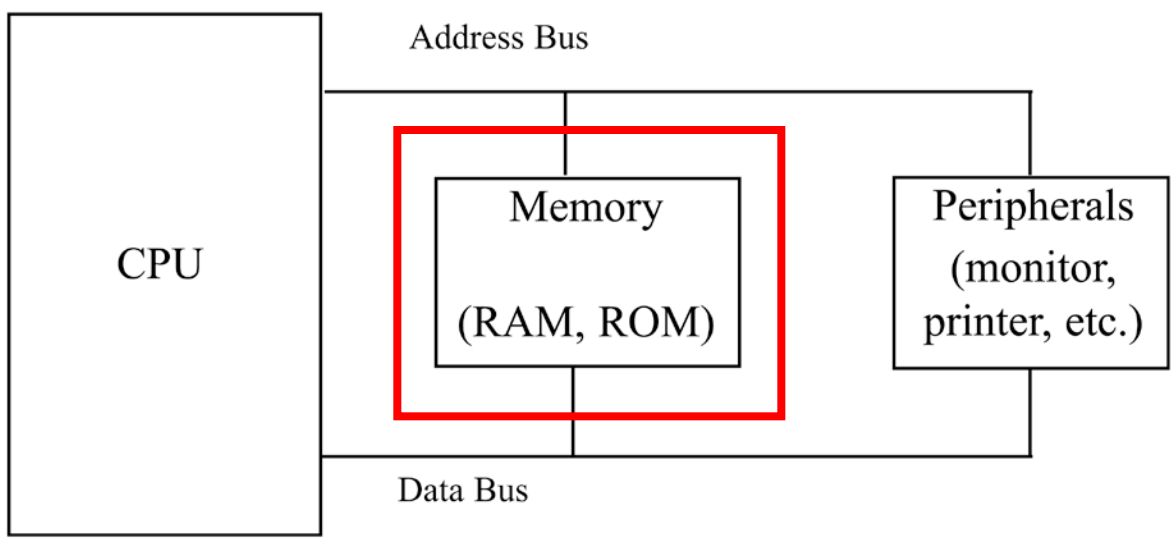


Figure 0-9
Inside the Computer

0.3 Inside the Computer
internal organization of computers

- Internal workings of every computer can be broken down into three parts:
  - **CPU** (central processing unit).
  - **Memory**.
  - **I/O** (input/output) devices.

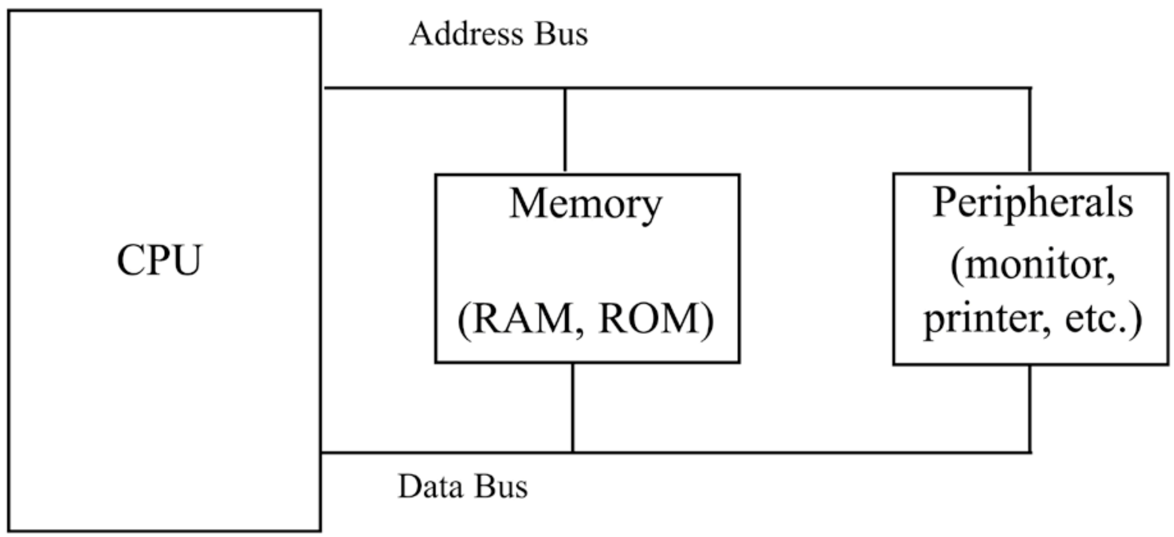


Figure 0-9
Inside the Computer

0.3 Inside the Computer
internal organization of computers

- CPU function is to execute (process) information stored in memory.
- I/O devices, such as keyboard & monitor provide a means of communicating with the CPU.
- The CPU is connected to memory and I/O through a group of wires called a *bus*.
  - Allows signals to carry information from place to place.
- In every computer there are three types of buses:
  - Address bus; Data bus; Control bus.

0.3 Inside the Computer
internal organization of computers

- For a device (memory or I/O) to be recognized by the CPU, it must be assigned an address.
  - No two devices can have the same address.
    - The address assigned to a given device must be unique.
- The CPU puts the address (in binary) on the address bus & decoding circuitry finds the device.
  - The CPU then uses the data bus either to get data from that device or to send data to it.
- Control buses provide device read/write signals.
  - To indicate if the CPU is asking for, or sending information.

0.3 Inside the Computer
internal organization of computers

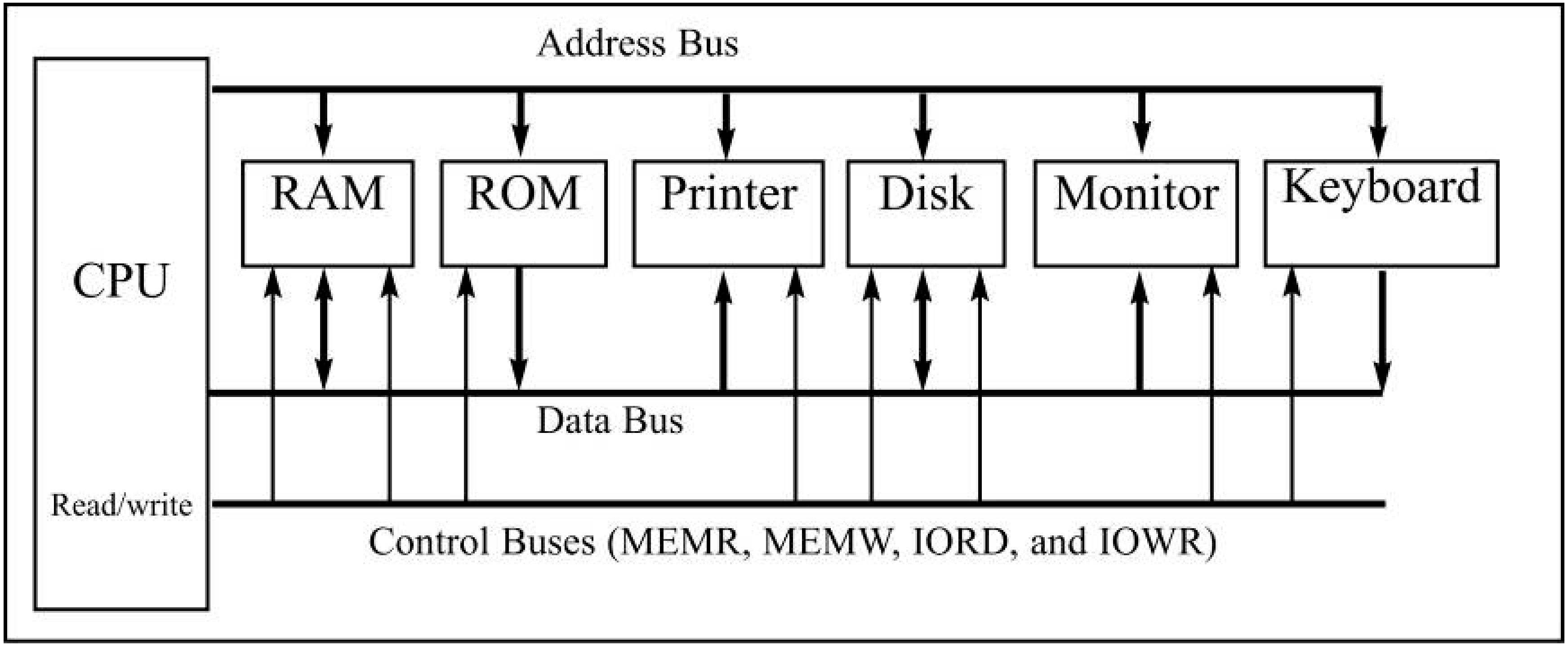


Figure 0-10 Internal Organization Of Computers

more about the data bus

- As data buses carry information in/out of a CPU, the more data buses available, the better the CPU.
  - More buses mean a more expensive CPU & computer.
- Data buses are bidirectional, because the CPU must use them either to receive or to send data.
  - Average bus size is between 8 and 64.
- Computer processing power is related to bus size.
  - An 8-bit bus can send out 1 byte a time.
  - A 16-bit bus can send out 2 bytes at a time.
    - Twice as fast.

more about the address bus

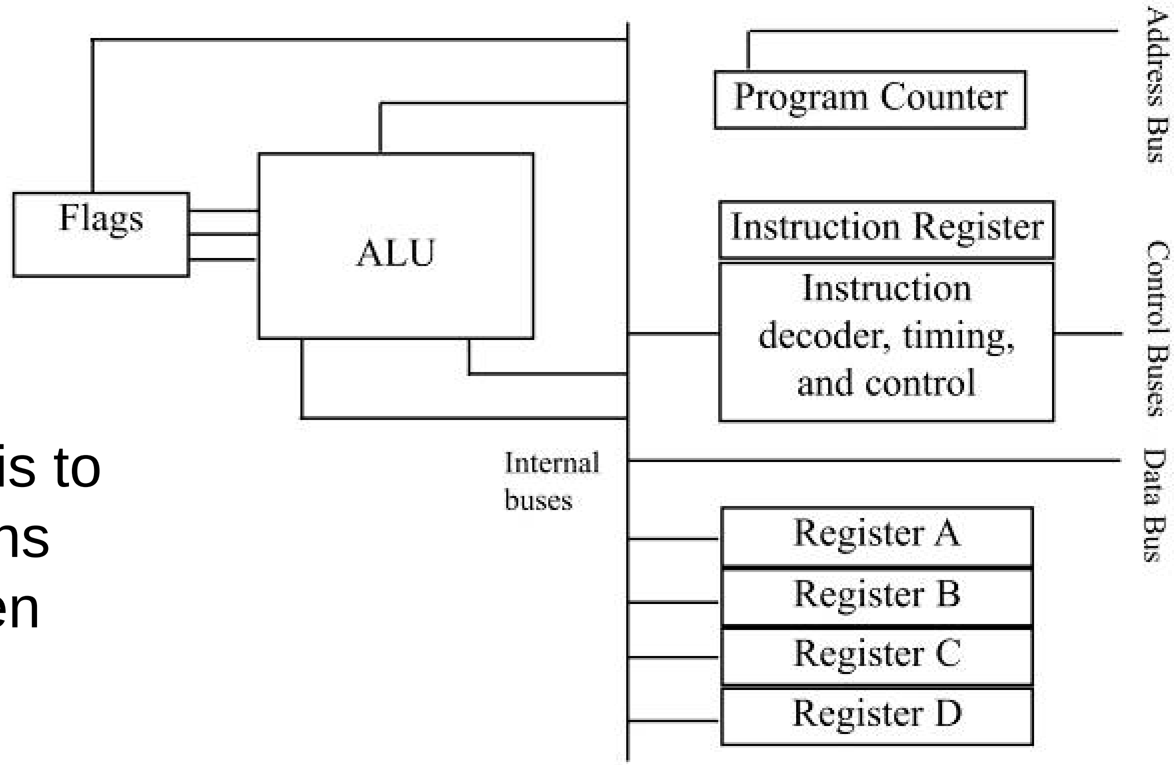
- The address bus is used to identify devices and memory connected to the CPU.
  - The more address bits available, the larger the number of devices that can be addressed.
- The number of CPU address bits determines the number of locations with which it can communicate.
  - Always equal to 2^x, where *x* is the number of address lines, regardless of the size of the data bus.
- The address bus is *unidirectional.*
  - The CPU uses the bus only to send addresses *out.*

0.3 Inside the Computer
CPU & relation to RAM and ROM

- For the CPU to process information, the data must be stored in RAM or ROM.
  - The CPU cannot get the information from the disk directly because the disk is too slow.
  - RAM & ROM are often referred to as *primary memory*.
    - Disks are called *secondary memory*.

0.3 Inside the Computer inside CPUs

- A program stored in memory provides instructions to the CPU to perform an action.
  – Adding payroll numbers or controlling a robot.



Function of the CPU is to fetch these instructions from memory and then execute them.

Figure 0-19 Internal Block Diagram of a CPU

CPU and relation to RAM and ROM

- To perform the actions of fetch and execute, all CPUs are equipped with resources such as…
  - ***Registers*** - to store information temporarily.
    - 8, 16, 32, 64 bit, depending on CPU.
  - ***ALU*** (arithmetic/logic unit) - for arithmetic functions such as add, subtract, multiply, and divide.
    - Also logic functions such as AND, OR, and NOT.
  - ***Program counter*** - to point to the address of the next instruction to be executed.
    - In the IBM PC, a register called IP or *instruction pointer.*
  - ***Instruction decoder*** - to interpret the instruction fetched into the CPU.

0.3 Inside the Computer
internal workings of computers

- A step-by-step analysis of CPU processes to add three numbers, with steps & code shown.
  - Assume a CPU has registers A, B, C, and D.
    - An 8-bit data bus and a 16-bit address bus.
  - The CPU can access memory addresses 0000 to FFFFH.
    - A total of 10000H locations.

Action	Code	Data
Move value 21H into register A	B0H	21H
Add value 42H to register A	04H	42H
Add value 12H to register A	04H	12H

internal workings of computers

- If the program to perform the actions listed above is stored in memory locations starting at 1400H, the following would represent the contents for each memory address location…

Memory address	***Contents of memory address***
1400	(B0) code for moving a value to register A
1401	(21) value to be moved
1402	(04) code for adding a value to register A
1403	(42) value to be added
1404	(04) code for adding a value to register A
1405	(12) value to be added
1406	(F4) code for halt

0.3 Inside the Computer
internal workings of computers

- The CPU's program counter can have a value between 0000 and FFFFH.
  - The program counter must be set to the address of the first instruction code to be executed - 1400H.

Memory address	***Contents of memory address***
$\boxed{1400}$	(B0)code for moving a value to register A
1401	(21)value to be moved
1402	(04)code for adding a value to register A
1403	(42)value to be added
1404	(04)code for adding a value to register A
1405	(12)value to be added
1406	(F4)code for halt

0.3 Inside the Computer
internal workings of computers

- The CPU puts the address 1400H on the address bus and sends it out.
  - Memory finds the location while the CPU activates the READ signal, indicating it wants the byte at 1400H.
    - The content (B0) is put on the data bus & brought to the CPU.

Memory address	***Contents of memory address***
$\boxed{1400}$	$\boxed{\text{(B0) code for moving a value to register A}}$
1401	(21) value to be moved
1402	(04) code for adding a value to register A
1403	(42) value to be added
1404	(04) code for adding a value to register A
1405	(12) value to be added
1406	(F4) code for halt

0.3 Inside the Computer
internal workings of computers

- The CPU decodes the instruction B0 with the help of its instruction decoder dictionary.
  - Bring the byte of the next memory location into CPU Register A.

Memory address	***Contents of memory address***
1400	(B0) code for moving a value to register A
1401	(21) value to be moved
1402	(04) code for adding a value to register A
1403	(42) value to be added
1404	(04) code for adding a value to register A
1405	(12) value to be added
1406	(F4) code for halt

0.3 Inside the Computer
internal workings of computers

- From memory location 1401H, the CPU fetches code 21H directly to Register A.
  - After completing the instruction, the program counter points to the address of the next instruction - 1402H.
    - Address 1402H is sent out on the address bus, to fetch the next instruction.

Memory address	***Contents of memory address***
1400	(B0) code for moving a value to register A
1401	(21) value to be moved
1402	(04) code for adding a value to register A
1403	(42) value to be added
1404	(04) code for adding a value to register A
1405	(12) value to be added
1406	(F4) code for halt

0.3 Inside the Computer
internal workings of computers

- From 1402H, the CPU fetches code 04H.
  - After decoding, the CPU knows it must add the byte at the next address (1403) to the contents of register A.
    - After it brings the value (42H) into the CPU, it provides the contents of Register A, along with this value to the ALU to perform the addition.
    - Program counter becomes 1404, the next instruction address.

Memory address	***Contents of memory address***
1400	(B0) code for moving a value to register A
1401	(21) value to be moved
1402	(04) code for adding a value to register A
1403	(42) value to be added
1404	(04) code for adding a value to register A
1405	(12) value to be added
1406	(F4) code for halt

0.3 Inside the Computer
internal workings of computers

- Address 1404H is put on the address bus and the code is fetched, decoded, and executed.
  - Again adding a value to Register A.
    - The program counter is updated to 1406H

Memory address	***Contents of memory address***
1400	(B0) code for moving a value to register A
1401	(21) value to be moved
1402	(04) code for adding a value to register A
1403	(42) value to be added
1404	(04) code for adding a value to register A
1405	(12) value to be added
1406	(F4) code for halt

0.4 Harvard and von Neumann CPU Architectures - internal workings

- The contents of address 1406 (HALT code) are fetched in and executed.
  - The HALT instruction tells the CPU to stop incrementing the program counter and asking for the next instruction.
    - Without HALT, the CPU would continue updating the program counter and fetching instructions.

Memory address	***Contents of memory address***
1400	(B0) code for moving a value to register A
1401	(21) value to be moved
1402	(04) code for adding a value to register A
1403	(42) value to be added
1404	(04) code for adding a value to register A
1405	(12) value to be added
1406	(F4) code for halt

Dec Hex Bin

0 0 00000000

ENDS ; ZERO

The x86 PC
assembly language, design, and interfacing

fifth edition

Prentice Hall

The x86 PC

assembly language, design, and interfacing

fifth edition

MUHAMMAD ALI MAZIDI
JANICE GILLISPIE MAZIDI
DANNY CAUSEY