THE 8051 MICROCONTROLLER AND EMBEDDED SYSTEMS

Muhammad Ali Mazidi
Janice Gillispie Mazidi

Prentice Hall
Upper Saddle River, New Jersey   Columbus, Ohio

.. man's glory lieth in his knowledge,
his upright conduct, his praiseworthy character,
his wisdom, and not in his nationality or rank.

Baha'u'llah

CONTENTS AT A GLANCE

CHAPTERS

APPENDICES

CONTENTS

INTRODUCTION

Products using microprocessors generally fall into two categories. The first category uses high-performance microprocessors such as the Pentium in applications where system performance is critical. We have an entire book dedicated to this topic, *The 80x86 IBM PC and Compatible Computers, Volumes I and II*, from Prentice Hall. In the second category of applications, performance is secondary; issues of space, power, and rapid development are more critical than raw processing power. The microprocessor for this category is often called a microcontroller.

This book is for the second category of applications. The 8051 is a widely used microcontroller. There are many reasons for this, including the existence of multiple producers and its simple architecture. This book is intended for use in college-level courses teaching microcontrollers and embedded systems. It not only establishes a foundation of assembly language programming, but also provides a comprehensive treatment of 8051 interfacing for engineering students. From this background, the design and interfacing of microcontroller-based embedded systems can be explored. This book can be also used by practicing technicians, hardware engineers, computer scientists, and hobbyists. It is an ideal source for those building stand-alone projects, or projects in which data is collected and fed into a PC for distribution on a network.

Prerequisites

Readers should have had an introductory digital course. Knowledge of a programming language would be helpful but is not necessary. Although the book is written for those with no background in assembly language programming, students with prior assembly language experience will be able to gain a mastery of 8051 architecture very rapidly and start on their projects right away.

Overview

A systematic, step-by-step approach is used to cover various aspects of 8051 Assembly language programming and interfacing. Many examples and sample programs are given to clarify the concepts and provide students with an opportunity to learn by doing. Review questions are provided at the end of each section to reinforce the main points of the section.

Chapter 0 covers number systems (binary, decimal, and hex), and provides an introduction to basic logic gates and computer terminology. This is designed especially for students, such as mechanical engineering students, who have not taken a digital logic course or those who need to refresh their memory on these topics.

Chapter 1 discusses 8051 history and features of other 8051 family members such as the 8751, 89C51, DS5000, and 8031. It also provides a list of various producers of 8051 chips.

Chapter 2 discusses the internal architecture of the 8051 and explains the use of an 8051 assembler to create ready-to-run programs. It also explores the stack and the flag register.

In Chapter 3 the topics of loop, jump, and call instructions are discussed, with many programming examples.

Chapter 4 is dedicated to the discussion of I/O ports. This allows students who are working on a project to start experimenting with 8051 I/O interfacing and start the project as soon as possible.

Chapter 5 covers the 8051 addressing modes and explains how to use the code space of the 8051 to store data, as well as how to access data.

Chapter 6 is dedicated to arithmetic instructions and programs.

Logic instructions and programs are covered in Chapter 7.

In Chapter 8 we discuss one of the most important features of the 8051, bit manipulation, as well as single-bit instructions of the 8051.

Chapter 9 describes the 8051 timers and how to use them as event-counters.

Chapter 10 is dedicated to serial data communication of the 8051 and its interfacing to the RS232. It also shows 8051 communication with COM ports of the IBM PC and compatible computers.

Chapter 11 provides a detailed discussion of 8051 interrupts with many examples on how to write interrupt handler programs.

Chapter 12 shows 8051 interfacing with real-world devices such as LCDs, ADCs, and sensors.

Chapter 13 shows 8051 interfacing with real world devices such as the keyboard, stepper motors, and DAC devices.

In Chapter 14 we cover 8031/51 interfacing with external memories, both ROM and RAM.

Finally, in Chapter 15 the issue of adding more ports to the 8031/51 is discussed, and the interfacing of an 8255 chip with the microcontroller is covered in detail.

The appendices have been designed to provide all reference material required for the topics covered in the book. Appendix A describes each 8051 instruction in detail, with examples. Appendix A also provides the clock count for instructions, 8051 register diagrams, and RAM memory maps. Appendix B describes wire wrapping, and how to design your own 8051 trainer board based on 89C51 or DS5000 chips. Appendix C covers IC technology and logic families, as well as 8051 I/O port interfacing and fan-out. Make sure you study this before connecting the 8051 to an external device. In Appendix D, the use of flowcharts and psuedocode is explored. Appendix E is for students familiar with x86 architecture who need to make a rapid transition to 8051 architecture. Appendix F provides the table for ASCII characters. Appendix G lists resources for assembler shareware, and electronics parts. Appendix H contains data sheets for the 8051 and other IC chips.

Diskette contents

The diskette attached to the book contains the lab manual, which has many experiments for software programming and hardware interfacing of the 8051. These are in Microsoft Word 97 format. In addition, the diskette contains the source code for all the programs in the book (in ASCII files). Also on the diskette are two guides for using 8051 assemblers and simulators from Franklin Software and Keil Corporation.

Acknowledgments

This book is the result of the dedication and encouragement of many individuals. Our sincere and heartfelt appreciation goes to all of them.

First, we would like to thank Professor Danny Morse, the most knowledgeable and experienced person on the 8051 that we know. He felt a strong need for a book such as this, and due to his lack of time he encouraged us to write it. He is the one who introduced us to this microcontroller and was always there, ready to discuss issues related to 8051 architecture.

Also we would like to express our sincere thanks to Professor Clyde Knight of Devry Institute of Technology for his helpful suggestions on the organization of the book.

In addition, the following professors and students found errors while using the book in its pre-publication form in their microcontroller course, and we thank them sincerely: Professor Phil Golden and John Berry of DeVry Institute of Technology, Robert Wrightson, Priscilla Martinez, Benjamin Fombon, David Bergman, John Higgins, Scot Robinson, Jerry Chrane, James Piott, Daniel Rusert, Michael Beard, Landon Hull, Jose Lopez, Larry Hill, David Johnson, Jerry Kelso, Michael Marshall, Marc Hoang, Trevor Isra.

Mr. Rolin McKinlay, an excellent student of the 8051, made many valuable suggestions, found many errors, and helped to produce the solution manual for the end-of-chapter problems. We sincerely appreciate his enthusiasm for this book.

Finally, we would like to thank the people at Prentice Hall, in particular our publisher, Mr. Charles Stewart, who continues to support and encourage our writing, and our production editor Alex Wolf who made the book a reality.

We enjoyed writing this book, and hope you enjoy reading it and using it for your courses and projects. Please let us know if you have any suggestions or find any errors.

Assemblers

The following gives two sites where you can download assemblers:

www.fsinc.com	for Franklin Software, Inc.
www.keil.com	for Keil Corporation

Another interesting web site is www.8052.com for more discussion on the microcontroller. Finally, the following site provides useful Intel manuals:

http://developer.intel.com/design/auto/mcs51/manuals

ABOUT THE AUTHORS

Muhammad Ali Mazidi holds Master's degrees from both Southern Methodist University and the University of Texas at Dallas, and currently is completing his Ph.D. in the Electrical Engineering Department of Southern Methodist University. He is a co-founder and chief researcher of Microprocessor Education Group, a company dedicated to bringing knowledge of microprocessors to the widest possible audience. He also teaches microprocessor-based system design at DeVry Institute of Technology in Dallas, Texas.

Janice Gillispie Mazidi has a Master of Science degree in Computer Science from the University of North Texas. After several years experience as a software engineer in Dallas, she co-founded Microprocessor Education Group, where she is the chief technical writer and production manager, and is responsible for software development and testing.

The Mazidis have been married since 1985 and have two sons, Robert Nabil and Michael Jamal.

The authors can be contacted at the following address if you have any comments or suggestions, or if you find any errors.

Microprocessor Education Group
P.O. Box 381970
Duncanville, TX 75138
U.S.A.

mmazidi@dal.devry.edu

This volume is dedicated to the memory of Dr. A. Davoodi, Professor of Tehran University, who in the tumultuous years of my youth taught me the importance of an independent search for truth. -- Muhammad Ali Mazidi

CHAPTER 0

INTRODUCTION TO COMPUTING

OBJECTIVES

Upon completion of this chapter, you will be able 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
- ⋙ Explain the difference between a bit, a nibble, a byte, and a word
- ⋙ Give precise mathematical definitions of the terms *kilobyte*, *megabyte*, *terabyte*, and *gigabyte*
- ⋙ 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 found in computers and describe the purpose of each type of bus
- ⋙ Describe the role of the CPU in computer systems
- ⋙ List the major components of the CPU and describe the purpose of each

To understand the software and hardware of a microcontroller-based system, one must first master some very basic concepts underlying computer design. In this chapter (which in the tradition of digital computers can be called Chapter 0), the fundamentals of numbering and coding systems are presented. After an introduction to logic gates, an overview of the workings inside the computer is given. Finally, in the last section we give a brief history of CPU architecture. Although some readers may have an adequate background in many of the topics of this chapter, it is recommended that the material be scanned, however briefly.

SECTION 0.1: NUMBERING AND CODING SYSTEMS

Whereas human beings use base 10 *(decimal)* arithmetic, computers use the base 2 (*binary*) system. In this section we explain how to convert from the decimal system to the binary system, and vice versa. The convenient representation of binary numbers, called *hexadecimal,* also is covered. Finally, the binary format of the alphanumeric code, called *ASCII*, is explored.

Decimal and binary number systems

Although there has been speculation that the origin of the base 10 system is the fact that human beings have 10 fingers, there is absolutely no speculation about the reason behind the use of the binary system in computers. The binary system is used in computers because 1 and 0 represent the two voltage levels of on and off. Whereas in base 10 there are 10 distinct symbols, 0, 1, 2, ..., 9, in base 2 there are only two, 0 and 1, with which to generate numbers. Base 10 contains digits 0 through 9; binary contains digits 0 and 1 only. These two binary digits, 0 and 1, are commonly referred to as *bits*.

Converting from decimal to binary

One method of converting from decimal to binary is to divide the decimal number by 2 repeatedly, keeping track of the remainders. This process continues until the quotient becomes zero. The remainders are then written in reverse order to obtain the binary number. This is demonstrated in Example 0-1.

Example 0-1

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

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. First, as an analogy, recall the weight of numbers in the base 10 system, as shown in the diagram. By the same token, each digit position in a number in base 2 has a weight associated with it:

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	=	$\underline{700000}$
		740683

				Decimal	***Binary***
110101_2 =					
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	=	$\underline{32}$	$\underline{100000}$
				53	110101

Knowing the weight of each bit in a binary number makes it simple to add them together to get its decimal equivalent, as shown in Example 0-2.

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}

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 shown in Example 0-3.

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

Hexadecimal system

Base 16, the *hexadecimal* system as it is called in computer literature, is used as a convenient representation of binary numbers. For example, it is much easier for a human being to represent a string of 0s and 1s such as 100010010110 as its hexadecimal equivalent of 896H. The binary system has 2 digits, 0 and 1. The base 10 system has 10 digits, 0 through 9. The hexadecimal (base 16) system has 16 digits. In base 16, the first 10 digits, 0 to 9, are the same as in decimal, and for the remaining six digits, the letters A, B, C, D, E, and F are used. Table 0-1 shows the equivalent binary, decimal, and hexadecimal representations for 0 to 15.

Table 0-1: Base 16 Number Systems

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

Converting between binary and hex

To represent a binary number as its equivalent hexadecimal number, start from the right and group 4 bits at a time, replacing each 4-bit binary number with its hex equivalent shown in Table 0-1. To convert from hex to binary, each hex digit is replaced with its 4-bit binary equivalent. See Examples 0-4 and 0-5.

Example 0-4

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.

Example 0-5

Convert hex 29B to binary.

Solution:

```
       2      9      B
  =  0010   1001   1011
```

Dropping the leading zeros gives 1010011011.

Converting from decimal to hex

Converting from decimal to hex could be approached in two ways:

1. Convert to binary first and then convert to hex. Example 0-6 shows this method of converting decimal to hex.
2. Convert directly from decimal to hex by repeated division, keeping track of the remainders. Experimenting with this method is left to the reader.

Example 0-6

(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

Converting from hex to decimal

Conversion from hex to decimal can also be approached in two ways:

1. Convert from hex to binary and then to decimal. Example 0-7 demonstrates this method of converting from hex to decimal.
2. Convert directly from hex to decimal by summing the weight of all digits.

Example 0-7

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

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

Table 0-2: Counting in Bases

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

Counting in bases 10, 2, and 16

To show the relationship between all three bases, in Table 0-2 we show the sequence of numbers from 0 to 31 in decimal, along with the equivalent binary and hex numbers. Notice in each base that when one more is added to the highest digit, that digit becomes zero and a 1 is carried to the next-highest digit position. For example, in decimal, 9 + 1 = 0 with a carry to the next-highest position. In binary, 1 + 1 = 0 with a carry; similarly, in hex, F + 1 = 0 with a carry.

Table 0 - 3: Binary Addition

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

Addition of binary and hex numbers

The addition of binary numbers is a very straightforward process. Table 0-3 shows the addition of two bits. The discussion of subtraction of binary numbers is bypassed since all computers use the addition process to implement subtraction. Although computers have adder circuitry, there is no separate circuitry for subtractors. Instead, adders are used in conjunction with *2's complement* circuitry to perform subtraction. In other words, to implement "*x* - *y*", the computer takes the 2's complement of y and adds it to x. The concept of 2's complement is reviewed next. Example 0-8 shows the addition of binary numbers.

Example 0-8

Add the following binary numbers. Check against their decimal equivalents.

Solution:

	Binary	*Decimal*
	1101	13
+	1001	9
	10110	22

2's complement

To get the 2's complement of a binary number, invert all the bits and then add 1 to the result. Inverting the bits is simply a matter of changing all 0s to 1s and 1s to 0s. This is called the *1's complement*. See Example 0-9.

Example 0-9

Take the 2's complement of 10011101.

Solution:

```
     10011101    binary number
     01100010    1's complement
  +         1
     01100011    2's complement
```

Addition and subtraction of hex numbers

In studying issues related to software and hardware of computers, it is often necessary to add or subtract hex numbers. Mastery of these techniques is essential. Hex addition and subtraction are discussed separately below.

Addition of hex numbers

This section describes the process of adding hex numbers. Starting with the least significant digits, the digits are added together. If the result is less than 16, write that digit as the sum for that position. If it is greater than 16, subtract 16 from it to get the digit and carry 1 to the next digit. The best way to explain this is by example, as shown in Example 0-10.

Example 0-10

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
```

Subtraction of hex numbers

In subtracting two hex numbers, if the second digit is greater than the first, borrow 16 from the preceding digit. See Example 0-11.

ASCII code

The discussion so far has revolved around the representation of number systems. Since all information in the computer must be represented by 0s and 1s, binary patterns must be assigned to letters and other characters. In the 1960s a standard representation called *ASCII* (American Standard Code for Information Interchange) was established. The ASCII (pronounced "ask-E") code assigns

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

binary patterns for numbers 0 to 9, all the letters of the English alphabet, both uppercase (capital) and lowercase, and many control codes and punctuation marks. The great advantage of this system is that it is used by most computers, so that information can be shared among computers. The ASCII system uses a total of 7 bits to represent each code. For example, 100 0001 is assigned to the uppercase letter "A" and 110 0001 is for the lowercase "a". Often, a zero is placed in the most significant bit position to make it an 8-bit code. Figure 0-1 shows selected ASCII codes. A complete list of ASCII codes is given in Appendix F. The use of ASCII is not only standard for keyboards used in the United States and many other countries but also provides a standard for printing and displaying characters by output devices such as printers and monitors.

Notice that the pattern of ASCII codes was designed to allow for easy manipulation of ASCII data. For example, digits 0 through 9 are represented by ASCII codes 30 through 39. This enables a program to easily convert ASCII to decimal by masking off the "3" in the upper nibble. Also notice that there is a relationship between the uppercase and lowercase letters. The uppercase letters are represented by ASCII codes 41 through 5A while lowercase letters are represent-represented by codes 61 through 7A. Looking at the binary code, the only bit that is different between the uppercase "A" and lowercase "a" is bit 5. Therefore, conversion between uppercase of lowercase is as simple as changing bit 5 of the ASCII code.

Example 0-11

Perform hex subtraction: 59F - 2B8.

Solution:

```
      59F        LSD:  8 from 15 = 7
    - 2B8              11 from 25 (9 + 16) = 14 (E)
      2E7              2 from 4 (5 - 1) = 2
```

Review Questions

1. Why do computers use the binary number system instead of the decimal system?
2. Convert 34_{10} to binary and hex.
3. Convert 110101_2 to hex and decimal.
4. Perform binary addition: 101100 + 101.
5. Convert 101100_2 to its 2's complement representation.
6. Add 36BH + F6H.
7. Subtract 36BH - F6H.
8. Write "80x86 CPUs" in its ASCII code (in hex form).

SECTION 0.2: DIGITAL PRIMER

This section gives an overview of digital logic and design. First, we cover binary logic operations, then we show gates that perform these functions. Next, logic gates are put together to form simple digital circuits. Finally, we cover some logic devices commonly found in microcontroller interfacing.

Binary logic

As mentioned earlier, computers use the binary number system because the two voltage levels can be represented as the two digits 0 and 1. Signals in digital electronics have two distinct voltage levels. For example, a system may define 0 V as logic 0 and +5 V as logic 1. Figure 0-2 shows this system with the built-in tolerances for variations in the voltage. A valid digital signal in this example should be within either of the two shaded areas.

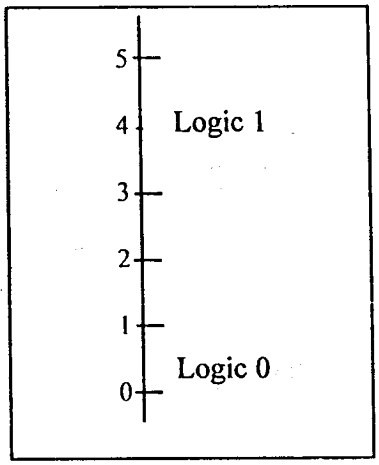


Figure 0-2. Binary Signals

Logic gates

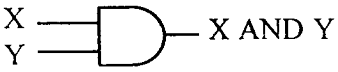
Binary logic gates are simple circuits that take one or more input signals and send out one output signal. Several of these gates are defined below.

AND gate

The AND gate takes two or more inputs and performs a logic AND on them. See the truth table and diagram of the AND gate. Notice that if both inputs to the AND gate are 1, the output will be 1. Any other combination of inputs will give a 0 output. The example shows two inputs, x and y. Multiple outputs are also possible for logic gates. In the case of AND, if all inputs are 1, the output is 1. If any input is 0, the output is zero.

Logical AND Function

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

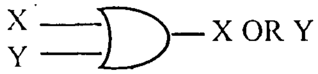


OR gate

The OR logic function will output a 1 if one or more inputs is 1. If all inputs are 0, then and only then will the output be 0.

Logical OR Function

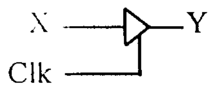
Inputs	**Output**
X Y	**X OR Y**
0 0	0
0 1	1
1 0	1
1 1	1



Tri-state buffer

A buffer gate does not change the logic level of the input. It is used to isolate or amplify the signal.

Buffer

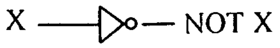


Inverter

The inverter, also called NOT, outputs the value opposite to that input to the gate. That is, a 1 input will give a 0 output, while a 0 input will give a 1 output.

Logical Inverter

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

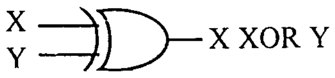


XOR gate

The XOR gate performs an exclusive-OR operation on the inputs. Exclusive-OR produces a 1 output if one (but only one) input is 1. If both operands are 0, the output is zero. Likewise, if both operands are 1, the output is also zero. Notice from the XOR truth table, that whenever the two inputs are the same, the output is zero. This function can be used to compare two bits to see if they are the same.

Logical XOR Function

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

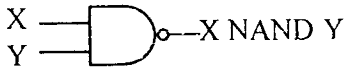


NAND and NOR gates

The NAND gate functions like an AND gate with an inverter on the output. It produces a zero output when all inputs are 1; otherwise, it produces a 1 output. The NOR gate functions like an OR gate with an inverter on the output. It produces a 1 if all inputs are 0; otherwise, it produces a 0. NAND and NOR gates are used extensively in digital design because they are easy and inexpensive to fabricate. Any circuit that can be designed with AND, OR, XOR, and INVERTER gates can be implemented using only NAND and NOR gates. A simple example of this is given below. Notice in NAND, that if any input is zero, the output is one. Notice in NOR, that if any input is one, the output is zero.

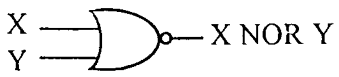
Logical NAND Function

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



Logical NOR Function

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

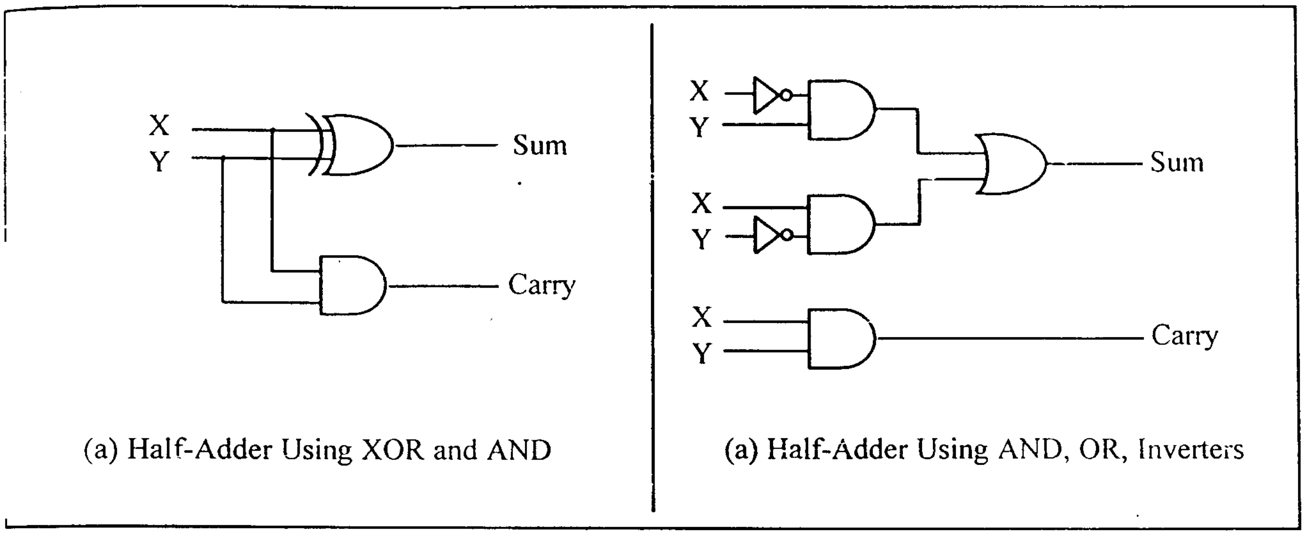


Logic design using gates

Next we will show a simple logic design to add two binary digits. If we add two binary digits there are four possible outcomes:

	Carry	*Sum*
0 + 0 =	0	0
0 + 1 =	0	1
1 + 0 =	0	1
1 + 1 =	1	0

Notice that when we add 1 + 1 we get 0 with a carry to the next higher place. We will need to determine the sum and the carry for this design. Notice that the sum column above matches the output for the XOR function, and that the carry column matches the output for the AND function. Figure 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.



(a) Half-Adder Using XOR and AND

(a) Half-Adder Using AND, OR, Inverters

Figure 0-3. Two Implementations of a Half-Adder

Figure 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. This is called a full-adder. Figure 0-5 shows the logic diagram of a full adder, along with a block diagram which masks the details of the circuit. Figure 0-6 shows a 3-bit adder using 3 full-adders.

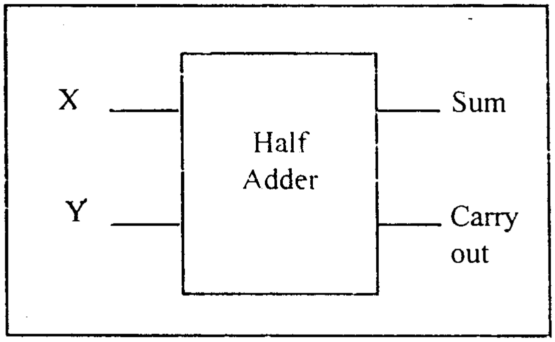


Figure 0-4. Block Diagram of a Half-Adder

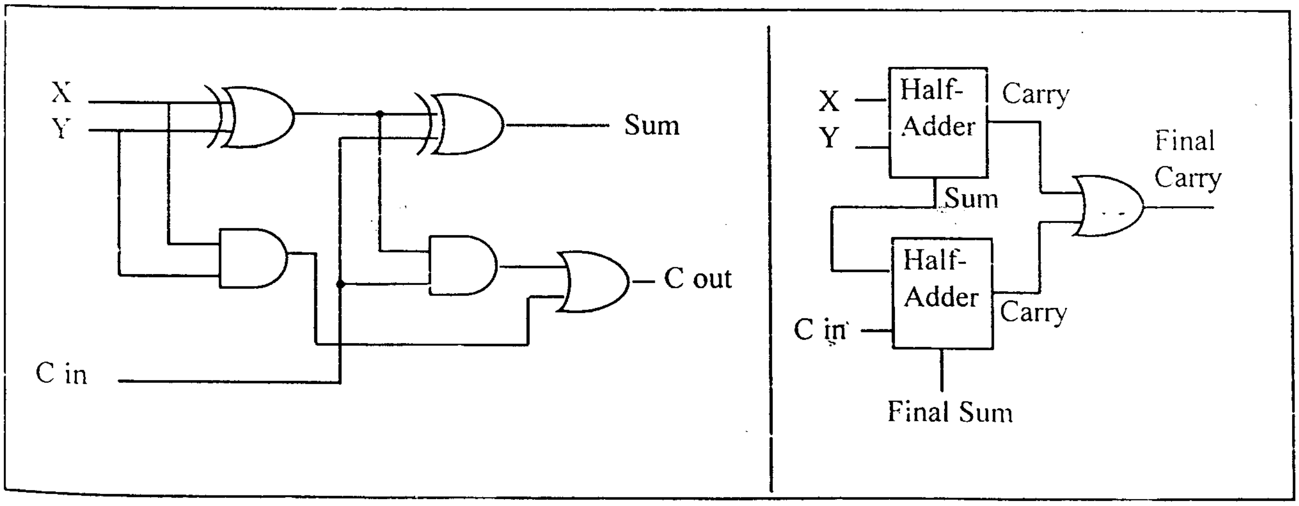


Figure 0-5. Full-Adder Built From a Half-Adder

Decoders

Another example of the application of logic gates is the decoder. Decoders are 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.

Flip-flops

A widely used component in digital systems is the flip-flop. Frequently, flip-flops are used to store data. Figure 0-8 shows the logic diagram, block diagram, and truth table for a flip-flop.

The D flip-flop is widely used to latch data. Notice from the truth table that a D-FF grabs the data at the input as the clock is activated. A D-FF holds the data as long as the power is on.

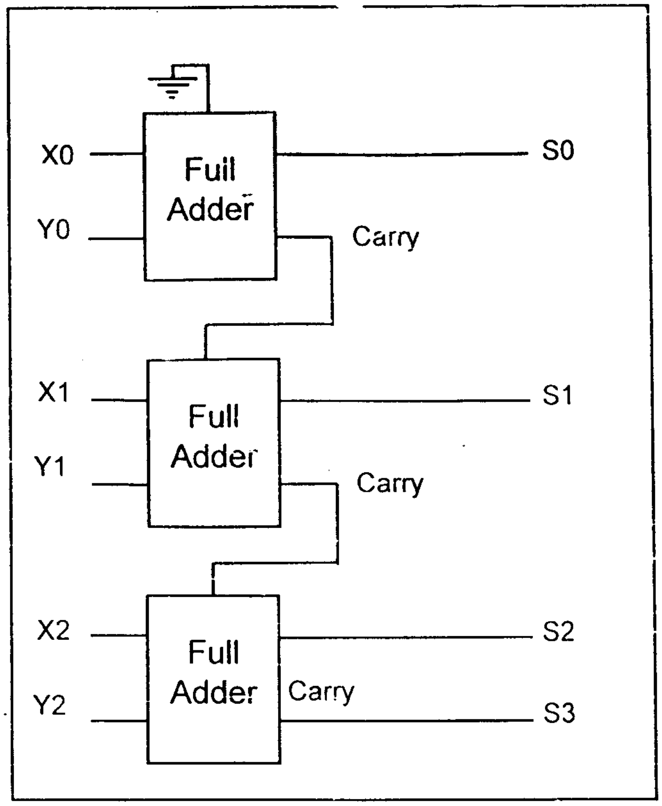
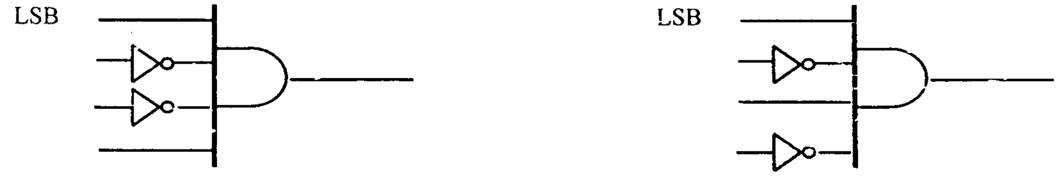


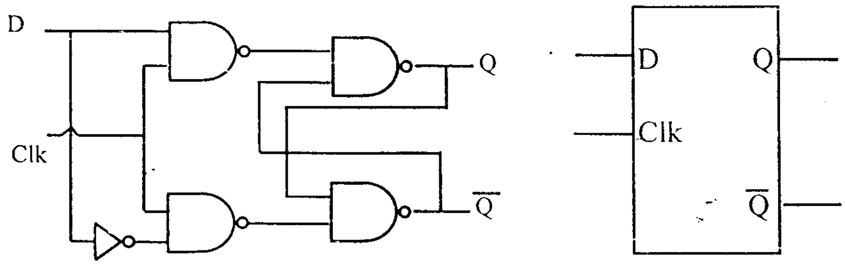
Figure 0-6. 3-Bit Adder Using 3 Full-Adders



(a) Address decoder for 9 (binary 1001)
The output of the AND gate will be 1
if and only if the input is binary 1001.

(b) Address decoder for 5 (binary 0101)
The output of the AND gate will be 1
if and only if the input is binary 0101.

Figure 0-7. Address Decoders



Clk	D	Q
No	x	no change
↓	0	0
↓	1	1

x = don't care

(a) Circuit diagram (b) Block diagram (c) Truth table

Figure 0-8. D Flip-Flops

Review Questions

1. The logical operation _____ gives a 1 output when all inputs are 1.
2. The logical operation _____ gives a 1 output when 1 or more of its inputs is 1.
3. The logical operation _____ is often used to compare if two inputs have the same value.
4. A _____ gate does not change the logic level of the input.
5. Name a common use for flip-flops.
6. An address ______ is used to identify a pre-determined binary address.

SECTION 0.3: INSIDE THE COMPUTER

In this section we provide an introduction to the organization and internal working of computers. The model used is generic, but the concepts discussed are applicable to all computers, including the IBM PC, PS/2, and compatibles. Before embarking on this subject, it will be helpful to review definitions of some of the most widely used terminology in computer literature, such as *K*, *mega*, *giga*, *byte*, *ROM*, *RAM*, and so on.

Some important terminology

One of the most important features of a computer is how much memory it has. Next we review terms used to describe amounts of memory in IBM PCs and compatibles. Recall from the discussion above that a *bit* is a binary digit that can have the value 0 or 1. A *byte* is defined as 8 bits. A *nibble* is half a byte, or 4 bits. A *word* is two bytes, or 16 bits. The display is intended to show the relative size of these units. Of course, they could all be composed of any combination of zeros and ones.

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

A *kilobyte* is 2^{10} bytes, which is 1024 bytes. The abbreviation K is often used. For example, some floppy disks hold 356K bytes of data. A *megabyte*, or meg as some call it, is 2^{20} bytes. That is a little over 1 million bytes; it is exactly 1,048,576 bytes. Moving rapidly up the scale in size, a *gigabyte* is 2^{30} bytes (over 1 billion), and a *terabyte* is 2^{40} bytes (over 1 trillion). As an example of how some of these terms are used, suppose that a given computer has 16 megabytes of memory. That would be 16×2^{20}, or $2^4 \times 2^{20}$, which is 2^{24}. Therefore 16 megabytes is 2^{24} bytes.

Two types of memory commonly used in microcomputers are *RAM*, which stands for "random access memory" (sometimes called *read/write memory*), and *ROM*, which stands for "read-only memory." RAM is used by the computer for temporary storage of programs that it is running. That data is lost when the computer is turned off. For this reason, RAM is sometimes called *volatile memory*. ROM contains programs and information essential to operation of the computer. The information in ROM is permanent, cannot be changed by the user, and is not lost when the power is turned off. Therefore, it is called *nonvolatile memory*.

Internal organization of computers

The internal working of every computer can be broken down into three parts: CPU (central processing unit), memory, and I/O (input/output) devices (see Figure 0-9). The function of the CPU is to execute (process) information stored in memory. The function of I/O devices such as the keyboard and video monitor is to provide a means of communicating with the CPU. The CPU is connected to memory and I/O through strips of wire called a *bus*. The bus inside a computer carries information from place to place just as a street bus carries people from place to place. In every computer there are three types of buses: address bus, data bus, and control bus.

For a device (memory or I/O) to be recognized by the CPU, it must be assigned an address. The address assigned to a given device must be unique; no two devices are allowed to have the same address. The CPU puts the address (of course, in binary) on the address bus, and the decoding circuitry finds the device. Then the CPU uses the data bus either to get data from that device or to send data to it. The control buses are used to provide read or write signals to the device to indicate if the CPU is asking for information or sending it information. Of the three buses, the address bus and data bus determine the capability of a given CPU.

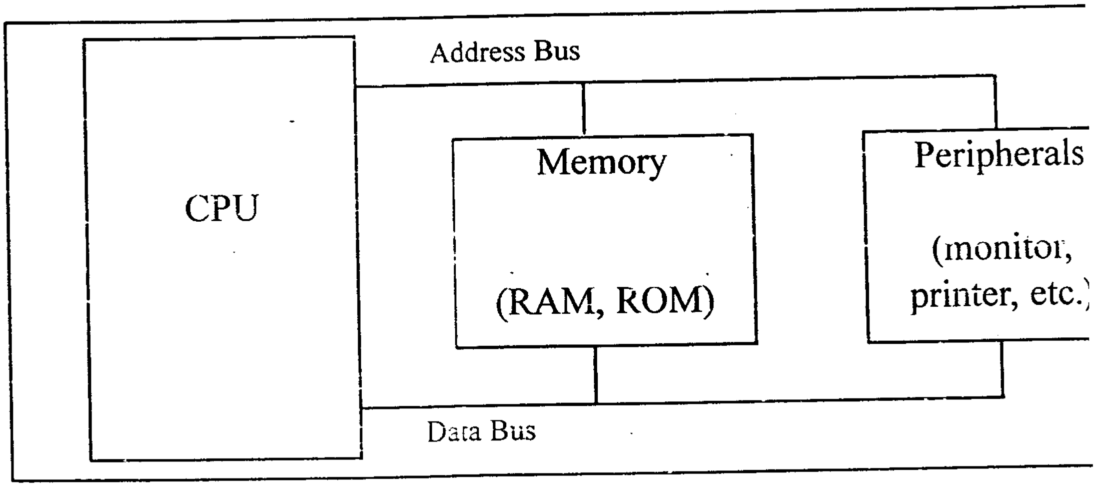


Figure 0-9: Inside the Computer

More about the data bus

Since data buses are used to carry information in and out of a CPU, the more data buses available, the better the CPU. If one thinks of data buses as highway lanes, it is clear that more lanes provide a better pathway between the CPU and its external devices (such as printers, RAM, ROM, etc.; see Figure 0-10). By the same token, that increase in the number of lanes increases the cost of construction. More data buses mean a more expensive CPU and computer. The average size of data buses in CPUs varies between 8 and 64. Early computers such as Apple 2 used an 8-bit data bus, while supercomputers such as Cray use a 64-bit data bus. Data buses are bidirectional, since the CPU must use them either to receive or to send data. The processing power of a computer is related to the size of its buses, since an 8-bit bus can send out 1 byte a time, but a 16-bit bus can send out 2 bytes at a time, which is twice as fast.

More about the address bus

Since the address bus is used to identify the devices and memory connected to the CPU, the more address buses available, the larger the number of devices that can be addressed. In other words, the number of address buses for a CPU determines the number of locations with which it can communicate. The number of locations is always equal to 2^x, where x is the number of address lines, regardless of the size of the data bus. For example, a CPU with 16 address lines can provide a total of 65,536 (2^{16}) or 64K bytes of addressable memory. Each location can have a maximum of 1 byte of data. This is due to the fact that all general-purpose microprocessor CPUs are what is called *byte addressable*. As another example, the IBM PC AT uses a CPU with 24 address lines and 16 data lines. In this case the total accessible memory is 16 megabytes (2^{24} = 16 megabytes). In this example there would be 2^{24} locations, and since each location is one byte, there would be 16 megabytes of memory. The address bus is a *unidirectional* bus, which means that the CPU uses the address bus only to send out addresses. To summarize: The total number of memory locations addressable by a given CPU is always equal to 2^x where x is the number of address bits, regardless of the size of the data bus.

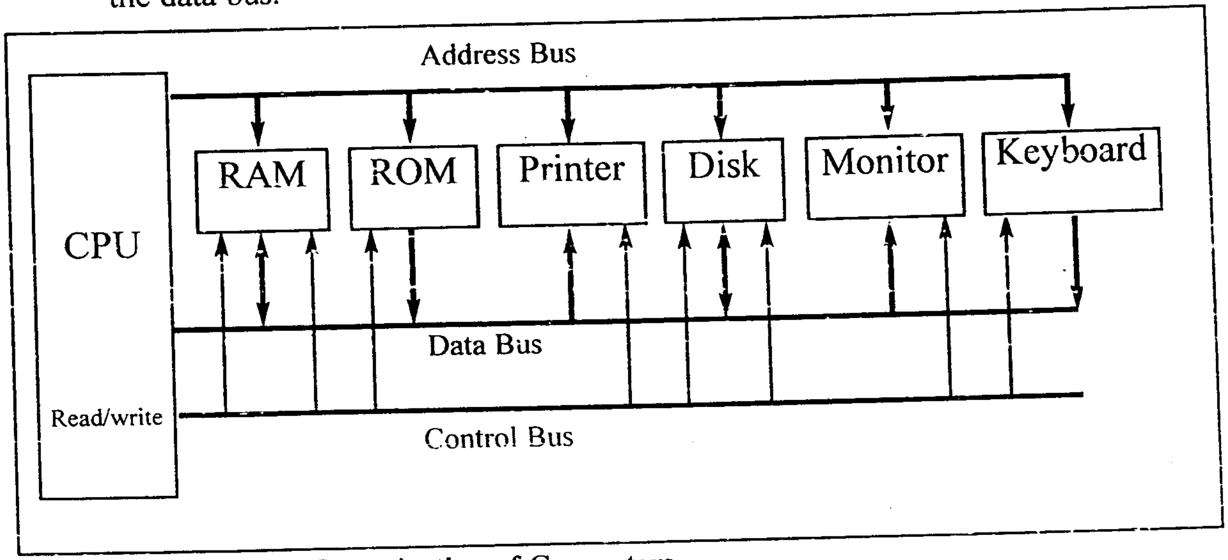


Figure 0-10: Internal Organization of Computers

CPU and its relation to RAM and ROM

For the CPU to process information, the data must be stored in RAM or ROM. The function of ROM in computers is to provide information that is fixed and permanent. This is information such as tables for character patterns to be displayed on the video monitor, or programs that are essential to the working of the computer, such as programs for testing and finding the total amount of RAM installed on the system, or programs to display information on the video monitor. In contrast, RAM is used to store information that is not permanent and can change with time, such as various versions of the operating system and application packages such as word processing or tax calculation packages. These programs are loaded into RAM to be processed by the CPU. The CPU cannot get the information from the disk directly since the disk is too slow. In other words, the CPU gets the information to be processed, first from RAM (or ROM). Only if it is not there does the CPU seek it from a mass storage device such as a disk, and then it transfers the information to RAM. For this reason, RAM and ROM are sometimes referred to as *primary memory* and disks are called *secondary memory*. Figure 0-11 shows a block diagram of the internal organization of the PC.

Inside CPUs

A program stored in memory provides instructions to the CPU to perform an action. The action can simply be adding data such as payroll data or controlling a machine such as a robot. It is the function of the CPU to fetch these instructions from memory and execute them. To perform the actions of fetch and execute, all CPUs are equipped with resources such as the following:

1. Foremost among the resources at the disposal of the CPU are a number of *registers*. The CPU uses registers to store information temporarily. The information could be two values to be processed, or the address of the value needed to be fetched from memory. Registers inside the CPU can be 8-bit, 16-bit, 32-bit, or even 64-bit registers, depending on the CPU. In general, the more and bigger the registers, the better the CPU. The disadvantage of more and bigger registers is the increased cost of such a CPU.
2. The CPU also has what is called the *ALU* (arithmetic/logic unit). The ALU section of the CPU is responsible for performing arithmetic functions such as add, subtract, multiply, and divide, and logic functions such as AND, OR, and NOT.
3. Every CPU has what is called a *program counter*. The function of the program counter is to point to the address of the next instruction to be executed. As each instruction is executed, the program counter is incremented to point to the address of the next instruction to be executed. It is the contents of the program counter that are placed on the address bus to find and fetch the desired instruction. In the IBM PC, the program counter is a register called IP, or the instruction pointer.
4. The function of the *instruction decoder* is to interpret the instruction fetched into the CPU. One can think of the instruction decoder as a kind of dictionary, storing the meaning of each instruction and what steps the CPU should take upon receiving a given instruction. Just as a dictionary requires more pages the more words it defines, a CPU capable of understanding more instructions requires more transistors to design.

Internal working of computers

To demonstrate some of the concepts discussed above, a step-by-step analysis of the process a CPU would go through to add three numbers is given next. Assume that an imaginary CPU has registers called A, B, C, and D. It has an 8-bit data bus and a 16-bit address bus. Therefore, the CPU can access memory from addresses 0000 to FFFFH (for a total of 10000H locations). The action to be performed by the CPU is to put hexadecimal value 21 into register A, and then add to register A values 42H and 12H. Assume that the code for the CPU to move a value to register A is 1011 0000 (B0H) and the code for adding a value to register A is 0000 0100 (04H). The necessary steps and code to perform them are as follows.

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

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

The actions performed by the CPU to run the program above would be as follows:

1. The CPU's program counter can have a value between 0000 and FFFFH. The program counter must be set to the value 1400H, indicating the address of the first instruction code to be executed. After the program counter has been loaded with the address of the first instruction, the CPU is ready to execute.
2. The CPU puts 1400H on the address bus and sends it out. The memory circuitry finds the location while the CPU activates the READ signal, indicating to memory that it wants the byte at location 1400H. This causes the contents of memory location 1400H, which is B0, to be put on the data bus and brought into the CPU.
3. The CPU decodes the instruction B0 with the help of its instruction decoder dictionary. When it finds the definition for that instruction it knows it must bring into register A of the CPU the byte in the next memory location. Therefore, it commands its controller circuitry to do exactly that. When it brings in value 21H from memory location 1401, it makes sure that the doors of all registers are closed except register A. Therefore, when value 21H comes into the CPU it will go directly into register A. After completing one instruction, the program counter points to the address of the next instruction to be executed, which in this case is 1402H. Address 1402 is sent out on the address bus to fetch the next instruction.
4. From memory location 1402H it fetches code 04H. After decoding, the CPU knows that it must add to the contents of register A the byte sitting at the next address (1403). After it brings the value (in this case 42H) into the CPU, it provides the contents of register A along with this value to the ALU to perform the addition. It then takes the result of the addition from the ALU's output and

puts it in register A. Meanwhile the program counter becomes 1404, the address of the next instruction.

5. Address 1404H is put on the address bus and the code is fetched into the CPU, decoded, and executed. This code is again adding a value to register A. The program counter is updated to 1406H.
6. Finally, the contents of address 1406 are fetched in and executed. This HALT instruction tells the CPU to stop incrementing the program counter and asking for the next instruction. In the absence of the HALT, the CPU would continue updating the program counter and fetching instructions.

Now suppose that address 1403H contained value 04 instead of 42H. How would the CPU distinguish between data 04 to be added and code 04? Remember that code 04 for this CPU means move the next value into register A. Therefore, the CPU will not try to decode the next value. It simply moves the contents of the following memory location into register A, regardless of its value.

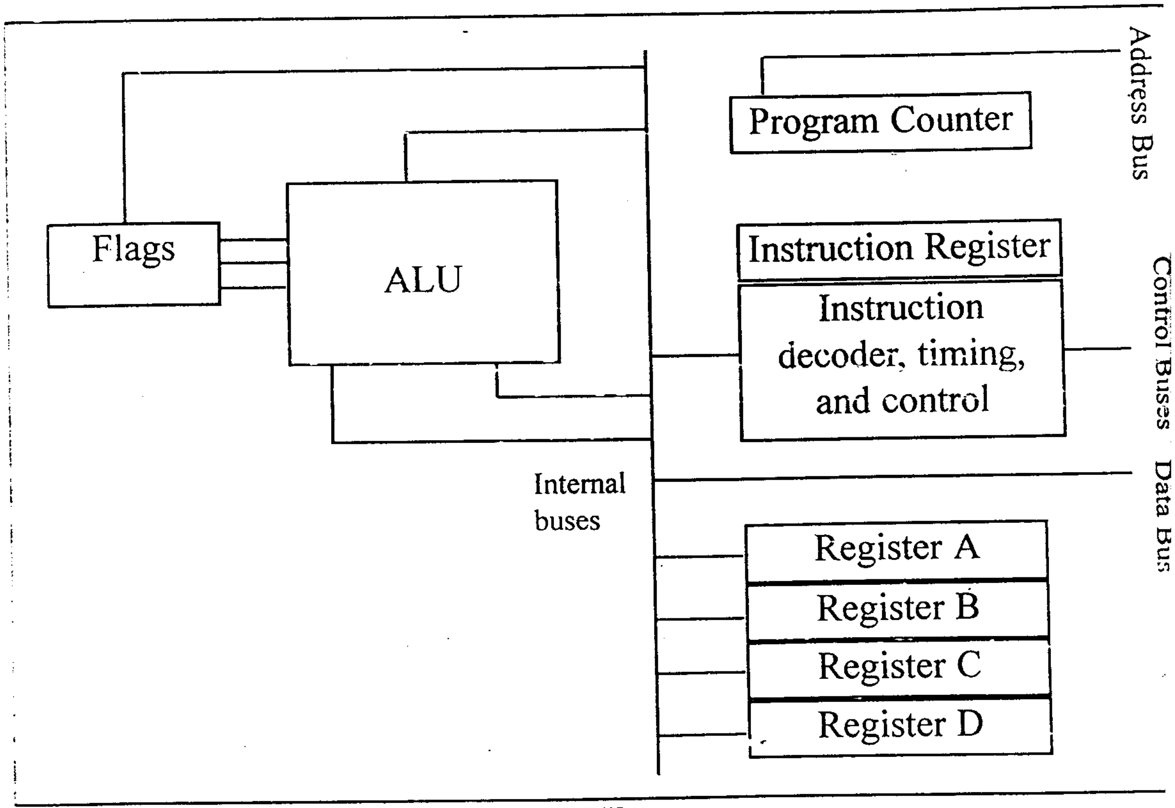


Figure 0-11: Internal Block Diagram of a CPU

Review Questions

1. How many bytes is 24 kilobytes?
2. What does "RAM" stand for? How is it used in computer systems?
3. What does "ROM" stand for? How is it used in computer systems?
4. Why is RAM called volatile memory?
5. List the three major components of a computer system.
6. What does "CPU" stand for? Explain its function in a computer.

7. List the three types of buses found in computer systems and state briefly the purpose of each type of bus.
8. State which of the following is unidirectional and which is bidirectional.
   (a) data bus (b) address bus
9. If an address bus for a given computer has 16 lines, what is the maximum amount of memory it can access?
10. What does "ALU" stand for? What is its purpose?
11. How are registers used in computer systems?
12. What is the purpose of the program counter?
13. What is the purpose of the instruction decoder?

SUMMARY

The binary number system represents all numbers with a combination of the two binary digits, 0 and 1. The use of binary systems is necessary in digital computers because only two states can be represented: on or off. Any binary number can be coded directly into its hexadecimal equivalent for the convenience of humans. Converting from binary/hex to decimal, and vice versa, is a straightforward process that becomes easy with practice. The ASCII code is a binary code used to represent alphanumeric data internally in the computer. It is frequently used in peripheral devices for input and/or output.

The logic gates AND, OR, and Inverter are the basic building blocks of simple circuits. NAND, NOR, and XOR gates are also used to implement circuit design. Diagrams of half-adders and full-adders were given as examples of the use of logic gates for circuit design. Decoders are used to detect certain addresses. Flip-flops are used to latch in data until other circuits are ready for it.

The major components of any computer system are the CPU, memory, and I/O devices. "Memory" refers to temporary or permanent storage of data. In most systems; memory can be accessed as bytes or words. The terms *kilobyte*, *megabyte*, *gigabyte*, and *terabyte* are used to refer to large numbers of bytes. There are two main types of memory in computer systems: RAM and ROM. RAM (random access memory) is used for temporary storage of programs and data. ROM (read-only memory) is used for permanent storage of programs and data that the computer system must have in order to function. All components of the computer system are under the control of the CPU. Peripheral devices such as I/O (input/output) devices allow the CPU to communicate with humans or other computer systems. There are three types of buses in computers: address, control, and data. Control buses are used by the CPU to direct other devices. The address bus is used by the CPU to locate a device or a memory location. Data buses are used to send information back and forth between the CPU and other devices.

Finally, this chapter gave an overview of digital logic.

PROBLEMS

SECTION 0.1: NUMBERING AND CODING SYSTEMS

1. Convert the following decimal numbers to binary.
   (a) 12 (b) 123 (c) 63 (d) 128 (e) 1000
2. Convert the following binary numbers to decimal.
   (a) 100100 (b) 1000001 (c) 11101 (d) 1010 (e) 00100010
3. Convert the values in Problem 2 to hexadecimal.
4. Convert the following hex numbers to binary and decimal.
   (a) 2B9H (b) F44H (c) 912H (d) 2BH (e) FFFFH
5. Convert the values in Problem 1 to hex.
6. Find the 2's complement of the following binary numbers.
   (a) 1001010 (b) 111001 (c) 10000010 (d) 111110001
7. Add the following hex values.
   (a) 2CH+3FH (b) F34H+5D6H (c) 20000H+12FFH (d) FFFFH+2222H
8. Perform hex subtraction for the following.
   (a) 24FH-129H (b) FE9H-5CCH (c) 2FFFFH-FFFFFH (d) 9FF25H-4DD99H
9. Show the ASCII codes for numbers 0, 1, 2, 3, ..., 9 in both hex and binary.
10. Show the ASCII code (in hex) for the following string:
    "U.S.A. is a country" CR,LF
    "in North America" CR,LF
    CR is carriage return
    LF is line feed

SECTION 0.2: DIGITAL PRIMER

11. Draw a 3-input OR gate using a 2-input OR gate.
12. Show the truth table for a 3-input OR gate.
13. Draw a 3-input AND gate using a 2-input AND gate.
14. Show the truth table for a 3-input AND gate.
15. Design a 3-input XOR gate with a 2-input XOR gate. Show the truth table for a 3-input XOR.
16. List the truth table for a 3-input NAND.
17. List the truth table for a 3-input NOR.
18. Show the decoder for binary 1100.
19. Show the decoder for binary 11011.
20. List the truth table for a D-FF.

SECTION 0.3: INSIDE THE COMPUTER

21. Answer the following:
    (a) How many nibbles are 16 bits?
    (b) How many bytes are 32 bits?
    (c) If a word is defined as 16 bits, how many words is a 64-bit data item?
    (d) What is the exact value (in decimal) of 1 meg?

(e) How many K is 1 meg?
(f) What is the exact value (in decimal) of i giga?
(g) How many K is 1 giga?
(h) How many meg is 1 giga?
(i) If a given computer has a total of 8 megabytes of memory, how many bytes (in decimal) is this? How many kilobytes is this?

22. A given mass storage device such as a hard disk can store 2 gigabytes of information. Assuming that each page of text has 25 rows and each row has 80 columns of ASCII characters (each character = 1 byte), approximately how many pages of information can this disk store?

23. In a given byte-addressable computer, memory locations 10000H to 9FFFFH are available for user programs. The first location is 10000H and the last location is 9FFFFH. Calculate the following:
(a) The total number of bytes available (in decimal)
(b) The total number of kilobytes (in decimal)

24. A given computer has a 32-bit data bus. What is the largest number that can be carried into the CPU at a time?

25. Below are listed several computers with their data bus widths. For each computer, list the maximum value that can be brought into the CPU at a time (in both hex and decimal).
(a) Apple 2 with an 8-bit data bus
(b) IBM PS/2 with a 16-bit data bus
(c) IBM PS/2 model 80 with a 32-bit data bus
(d) CRAY supercomputer with a 64-bit data bus

26. Find the total amount of memory, in the units requested, for each of the following CPUs, given the size of the address buses.
(a) 16-bit address bus (in K)
(b) 24-bit address bus (in meg)
(c) 32-bit address bus (in megabytes and gigabytes)
(d) 48-bit address bus (in megabytes, gigabytes, and terabytes)

27. Regarding the data bus and address bus, which is uniairectional and which is bidirectional?

28. Which register of the CPU holds the address of the instruction to be fetched?

29. Which section of the CPU is responsible for performing addition?

30. List the three bus types present in every CPU.

ANSWERS TO REVIEW QUESTIONS

SECTION 0.1: NUMBERING AND CODING SYSTEMS

1. Computers use the binary system because each bit can have one of two voltage levels: on and off.
2. $34_{10} = 100010_2 = 22_{16}$
3. $110101_2 = 35_{16} = 53_{10}$
4. 1110001
5. 010100
6. 461
7. 275
8. 38 30 78 38 36 20 43 50 55 73

SECTION 0.2: DIGITAL PRIMER

1. AND   2. OR
3. XOR   4. Buffer
5. Storing data
6. Decoder

SECTION 0.3: INSIDE THE COMPUTER

1. 24,576
2. Random access memory; it is used for temporary storage of programs that the CPU is running, such as the operating system, word processing programs, etc.
3. Read-only memory; it is used for permanent programs such as those that control the keyboard, etc.
4. The contents of RAM are lost when the computer is powered off.
5. The CPU, memory, and I/O devices
6. Central processing unit; it can be considered the "brain" of the computer; it executes the programs and controls all other devices in the computer.
7. The address bus carries the location (address) needed by the CPU; the data bus carries information in and out of the CPU; the control bus is used by the CPU to send signals controlling I/O devices.
8. (a) bidirectional (b) unidirectional
9. 64K, or 65,536 bytes
10. Arithmetic/logic unit; it performs all arithmetic and logic operations.
11. It is for temporary storage of information.
12. It holds the address of the next instruction to be executed.
13. It tells the CPU what steps to perform for each instruction.