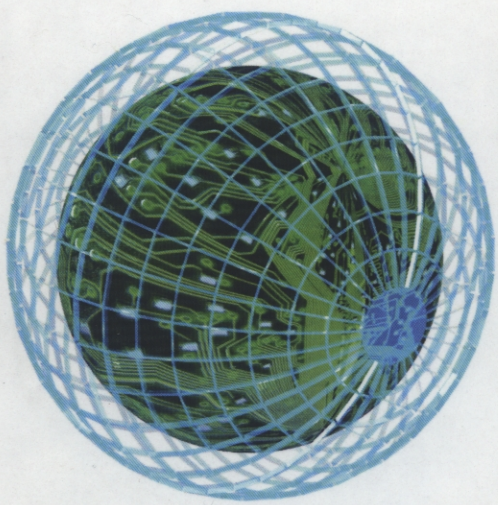
the avr microcontroller and embedded system

using assembly and c



MUHAMMAD ALI MAZIDI
SARMAD NAIMI
SEPEHR NAIMI

the avr microcontroller and embedded systems
using assembly and c

MUHAMMAD ALI MAZIDI, SARMAD NAIMI, AND SEPEHR NAIMI

The AVR microcontroller from Atmel is one of the most widely used 8-bit microcontrollers in the world. In this book the authors use a step-by-step and systematic approach to show the programming of the AVR chip. Examples in both Assembly language and C show how to program many of the AVR features, such as timers, serial communication, ADC, SPI, I2C, and PWM. The text is organized into two parts:

- The first seven chapters use Assembly language programming to examine the internal architecture of the AVR.
- Chapters 7–18 use both Assembly and C to show the AVR peripherals and I/O interfacing to real-world devices such as LCDs, motors, and sensors.

The AVR Microcontroller and Embedded Systems is the latest volume in the series of textbooks by Mazidi et al. This series of texts is widely used around the world by both industry and academics and has been translated into many languages. The other titles in the series are:

The x86 PC (5th ed.)
The 8051 Microcontroller and Embedded Systems (2nd ed.)
The PIC Microcontroller and Embedded Systems
The HCS12 Microcontroller and Embedded Systems

Titles to come include:

The ARM Microcontroller and Embedded Systems

CHAPTERS

ISBN-13: 978-0-13-800331-9
ISBN-10: 0-13-800331-9

Prentice Hall
is an imprint of

PEARSON

www.pearsonhighered.com

THE AVR MICROCONTROLLER AND EMBEDDED SYSTEMS

Using Assembly and C

Muhammad Ali Mazidi
Sarmad Naimi
Sepehr Naimi

Prentice Hall

Boston Columbus Indianapolis New York San Francisco Upper Saddle River
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto
Delhi Mexico City Sao Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Editor in Chief: Vernon Anthony
Acquisitions Editor: Wyatt Morris
Editorial Assistant: Chris Reed
Director of Marketing: David Gesell
Marketing Manager: Kara Clark
Senior Managing Coordinator: Alicia Wozniak
Marketing Assistant: Les Roberts
Senior Managing Editor: JoEllen Gohr
Project Manager: Rex Davidson
Senior Operations Supervisor: Pat Tonneman
Operations Specialist: Laura Weaver
Art Director: Dianne Ernsberger
Cover Designer: Jeff Vanik
Cover Art: Antonis Papantoniou, Fotolia.com
Printer/Binder: Courier/Kendallville
Cover Printer: Demand Production Center
Text Font: Times Roman

Library of Congress Cataloging in Publication Data

Mazidi, Muhammad Ali.
The AVR microcontroller and embedded systems: using Assembly and C / Muhammad Ali Mazidi, Sarmad Naimi, Sepehr Naimi.
p. cm.
ISBN-13: 978-0-13-800331-9 (alk. paper)
ISBN-10: 0-13-800331-9 (alk. paper)
1. Atmel AVR microcontroller. 2. Embedded computer systems. 3. Assembler language (Computer program language) 4. C (Computer program language) I. Naimi, Sarmad. II. Naimi, Sepehr. III. Title.
TJ223.P76M378136 2009
004.16--dc22
2009039790

10 9 8 7 6 5 4 3 2 1
Prentice Hall
is an imprint of

www.pearsonhighered.com

ISBN 10: 0-13-800331-9
ISBN 13: 978-0-13-800331-9

This book is dedicated
to the memory of Dr. Kamal Bakhtavar
for all his sacrifices.
– Muhammad Ali Mazidi

This book is dedicated
to the memory of Dr. P. Javid
for his inspiring example of dedication to the education of young people.
– Sarmad Naimi

This book is dedicated to
Yaran.
– Sepehr Naimi

Regard man as a mine rich in gems of inestimable value. Education can, alone, cause it to reveal its treasures, and enable mankind to benefit therefrom.

Baha'u'llah

BRIEF CONTENTS

CHAPTERS

APPENDICES

CONTENTS

PREFACE

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 x86 PC: Assembly Language, Design, and Interfacing*, published by Prentice Hall. In the second category of applications, performance is secondary; issues of cost, 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 AVR is a widely used microcontroller. 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 AVR interfacing for engineering students. From this background, the design and interfacing of microcontroller-based embedded systems can be explored. This book can also be used by practicing technicians, hardware engineers, computer scientists, and hobbyists.

Prerequisites

Readers should have had an introductory digital course. Knowledge of Assembly language would be helpful, but is not necessary. Although this 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 AVR architecture very rapidly and start on their projects right away. For the AVR C programming sections of the book, a basic knowledge of C programming is required. We use the AVR Studio compiler IDE from Atmel throughout the book. The AVR Studio compiler is available for free from the Atmel website (www.atmel.com). We encourage you to use the AVR Studio or some other IDE to simulate and run the programs in this book.

Overview

A systematic, step-by-step approach is used to cover various aspects of AVR C and 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 memory. This chapter 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 the history of the AVR and features of the members such as ATmega32. It also provides a list of various members of the AVR family.

Chapter 2 discusses the internal architecture of the AVR and explains the use of a AVR assembler to create ready-to-run programs. It also explores the program counter 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 AVR I/O interfacing and start the hardware project as soon as possible.

Chapter 5 is dedicated to arithmetic, logic instructions, and programs.

Chapter 6 covers the AVR advanced addressing modes and explains how to access the data stored in the look-up table, as well as how to use EEPROM to store data and how to do macros.

The C programming of the AVR is covered in Chapter 7. We use the WinAVR compiler for this and throughout the book. The WinAVR is available for free from the winavr.sourceforge.net website.

In Chapter 8 we discuss the hardware connection of the AVR chip.

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

Chapter 10 provides a detailed discussion of AVR interrupts with many examples on how to write interrupt handler programs.

Chapter 11 is dedicated to serial data communication of the AVR and its interfacing to the RS232. It also shows AVR communication with COM ports of the x86 IBM PC and compatible computers.

Chapter 12 shows AVR interfacing with real-world devices such as LCDs and keyboards.

Chapter 13 shows AVR interfacing with real-world devices such as DAC chips, ADC chips, and sensors.

Chapter 14 covers the basic interfacing of the AVR chip to relays, optoisolators, and stepper motors.

In Chapter 15 we cover how to use AVR timers to generate waves and explain how to capture waves to measure period and duty cycle.

Chapter 16 shows PWM and basic interfacing to DC motors.

Chapter 17 covers the SPI bus protocol and describes how to interface 7-segment displays using MAX7221.

Finally, Chapter 18 shows how to connect and program the DS1307 real-time clock chip using the TWI (I2C) bus protocol.

The appendices have been designed to provide all reference material required for the topics covered in the book. Appendix A describes each AVR instruction in detail, with examples. Appendix A also provides the clock count for instructions and AVR I/O registers. Appendix B describes the basics of wire wrapping. Appendix C examines IC interfacing and logic families, as well as AVR I/O port interfacing and fan-out. Make sure you study this section before connecting the AVR to an external device. In Appendix D, the use of flowcharts and pseudocode is explored. Appendix E is for students familiar with 8051 architectures who need to make a rapid transition to AVR architecture. Appendix F provides the table of ASCII characters. Appendix G lists resources for assembler shareware, AVR trainers, and electronics parts. Appendix H contains data sheets for the AVR chip.

Lab Manual

The lab manual covers some very basic labs and can be found at the **www.MicroDigitalEd.com** website. The more advanced and rigorous lab assign-

ments are left up to the instructors depending on the course objectives, class level, and whether the course is graduate or undergraduate. The support materials for this text and other books by the authors can be found on this website, too.

Solutions Manual/PowerPoint® Slides

The end-of-chapter problems cover some very basic concepts. The more challenging and rigorous homework assignments are left up to the instructors depending on the course objectives, class level, and whether the course is graduate or undergraduate. The solutions manual and PowerPoint® slides for the drawings are available online for instructors only.

Online Instructor Resources

To access supplementary materials online, instructors need to request an instructor access code. Go to **www.prenhall.com**, click the **Instructor Resource Center** link, and then click **Register Today** for an instructor access code. Within 48 hours after registering you will receive a confirming e-mail including an instructor access code. Once you have received your code, go to the site and log on for full instructions on downloading the materials you wish to use.

Acknowledgments

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

Thanks to the reviewers of this edition:
Orod Haghighi Ara, BIHE University;
Arona Kosari, BIHE University;
Anahita Omidvar, BIHE University;
Vahid Mokhtari, BIHE University;
Farshid Hoori, BIHE University;
Navid HajatDoost, BIHE University;
Hootan Rahmanian, BIHE University;
Farzad Sabeti, BIHE University;
Moshtagh Samandari, BIHE University.

Numerous students found errors or made suggestions in improving this book. We would like to thank all of them for their enthusiasm and support. Those students are: Arash Noori, Soroush Taefi, Golriz Nourani, Mozhdeh Amiri, Negar Ziaee Nasrabadi, and Maryam NouhNezhad, all from the computer engineering department of BIHE.

Finally, we would like to thank the people at Prentice Hall, in particular our editor, Wyatt Morris, who continues to support and encourage our writing, and our project manager, Rex Davidson, who made the book a reality. We were lucky to get the best copy editors in the world, Janice Mazidi and Bret Workman. Thank you both for your fantastic job, as usual.

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/Compilers

The AVR Studio can be downloaded from the following website:
http://www.Atmel.com

The WinAVR C compiler for AVR can be downloaded from the following website:
http://winavr.sourceforge.net

The tutorials for all the above assemblers/compilers and AVR Trainer boards can be found on the following website:
http://www.MicroDigitalEd.com

Trademark Information and Acknowledgments

ABOUT THE AUTHORS

Muhammad Ali Mazidi went to Tabriz University and holds Master's degrees from both Southern Methodist University and the University of Texas at Dallas. He is currently a.b.d. on his Ph.D. in the Electrical Engineering Department of Southern Methodist University. He is co-author of some widely used textbooks, including *The x86 PC, The 8051 Microcontroller and Embedded Systems, The PIC Microcontroller and Embedded Systems,* and *The HCS12 Microcontroller and Embedded Systems,* also available from Prentice Hall. He teaches microprocessor-based system design at DeVry University in Dallas, Texas. He is the founder of MicroDigitalEd.com.

Sarmad Naimi graduated from the Computer Engineering department of BIHE university and is currently working on his Master's degree. His areas of interest include FPGA, RTOS, and real-time embedded systems.

Sepehr Naimi graduated from the Computer Engineering department of BIHE university and is currently working on his Master's degree. His areas of interest include high-performance microcontrollers, RTOS, and real-time embedded systems.

The authors can be contacted at the following e-mail addresses if you have any comments or suggestions, or if you find any errors.

mdebooks@yahoo.com
mmazidi@microdigitaled.com
SarmadNaimi@gmail.com
Sepehr.Naimi@gmail.com

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**
- **>> Describe the logical operations AND, OR, NOT, XOR, NAND, and 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*, *gigabyte*, and *terabyte***
- **>> Describe the purpose of the major components of a computer system**
- **>> Contrast and compare various types of semiconductor memories in terms of their capacity, organization, and access time**
- **>> Describe the relationship between the number of memory locations on a chip, the number of data pins, and the chip's memory capacity**
- **>> Contrast and compare PROM, EPROM, UV-EPROM, EEPROM, Flash memory EPROM, and mask ROM memories**
- **>> Contrast and compare SRAM, NV-RAM, and DRAM memories**
- **>> List the steps a CPU follows in memory address decoding**
- **>> 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**
- **>> Understand the RISC and Harvard architectures**

To understand the software and hardware of a microcontroller-based system, one must first master some very basic concepts underlying computer architecture. In this chapter (which in the tradition of digital computers is called Chapter 0), the fundamentals of numbering and coding systems are presented in Section 0.1. In Section 0.2, an overview of logic gates is given. The semiconductor memory and memory interfacing are discussed in Section 0.3. In Section 0.4, CPUs and Harvard and von Neumann architectures are discussed. Finally, in the last section we give a brief history of RISC architecture. Although some readers may have an adequate background in many of the topics of this chapter, it is recommended that the material be reviewed, 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 of 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	=	700000
		740683

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

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, or 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 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

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
29B   =   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	First, convert to binary.
1	0	1	1	0	1	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.

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.

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

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.

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
```

ASCII code

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

The discussion so far has revolved around the representation of number systems. Because 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 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 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 and lowercase is as simple as changing bit 5 of the ASCII code.

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.

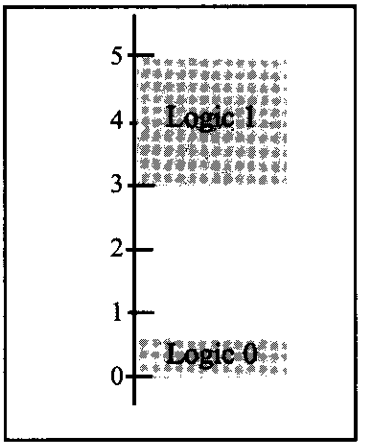


Figure 0-2. Binary Signals

Logic gates

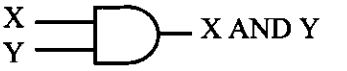
Binary logic gates are simple circuits that take one or more input signals and send out one output signal. Several of these gates are defined below.

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 0.

Logical AND Function

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

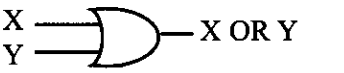


OR gate

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

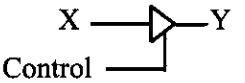
Inputs	Output
X Y	X OR Y
0 0	0
0 1	1
1 0	1
1 1	1



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

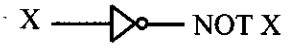


Inverter

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

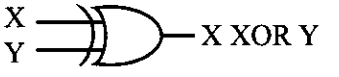


XOR gate

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 0. Likewise, if both operands are 1, the output is also 0. Notice from the XOR truth table, that whenever the two inputs are the same, the output is 0. 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

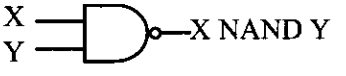


NAND and NOR gates

The NAND gate functions like an AND gate with an inverter on the output. It produces a 0 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 0, the output is 1. Notice in NOR, that if any input is 1, the output is 0.

Logical NAND Function

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

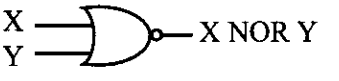


Logic design using gates

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

Logical NOR Function

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



	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 and inverters.

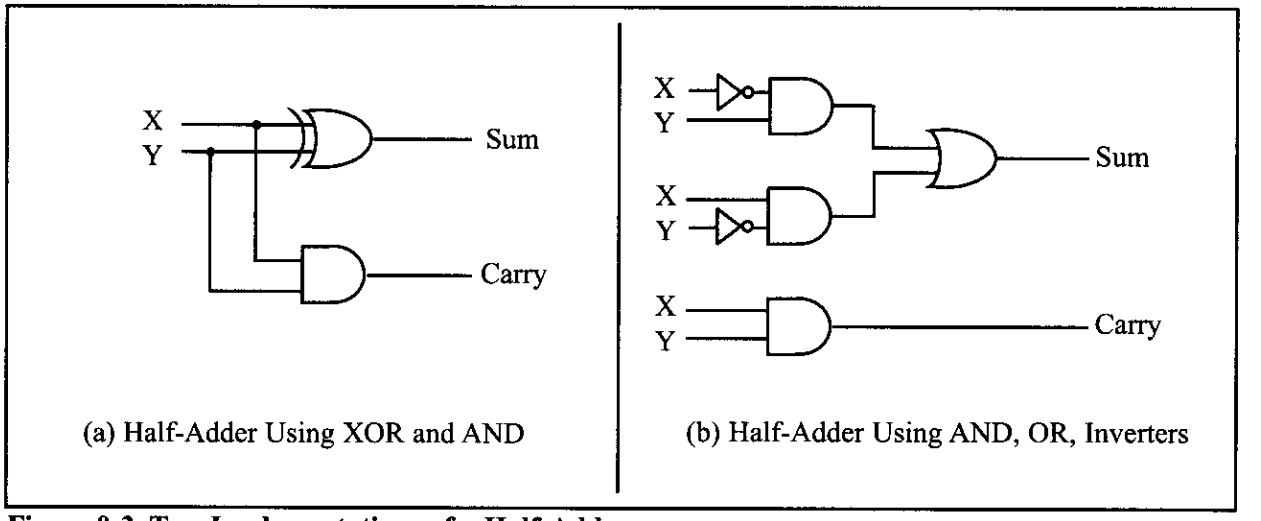


Figure 0-3. Two Implementations of a Half-Adder

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 that masks the details of the circuit. Figure 0-6 shows a 3-bit adder using three full-adders.

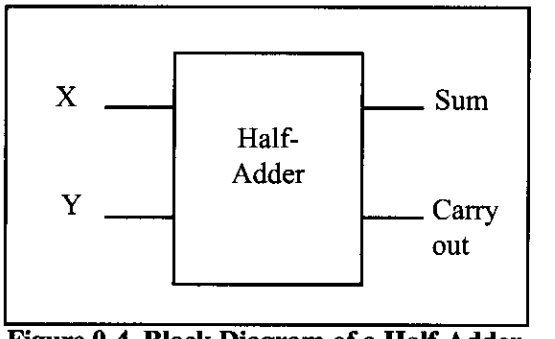


Figure 0-4. Block Diagram of a Half-Adder

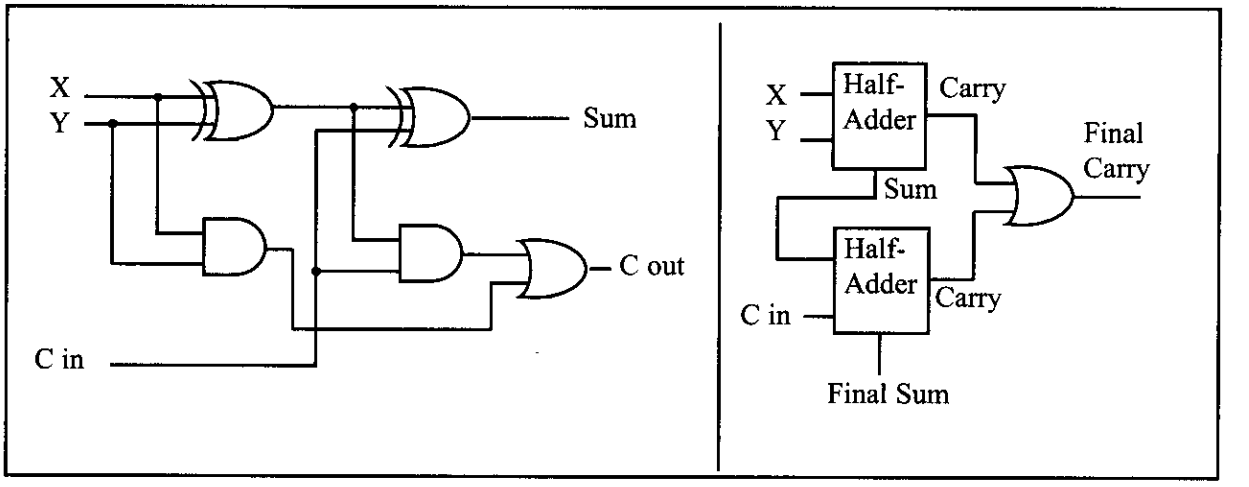


Figure 0-5. Full-Adder Built from a Half-Adder

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.

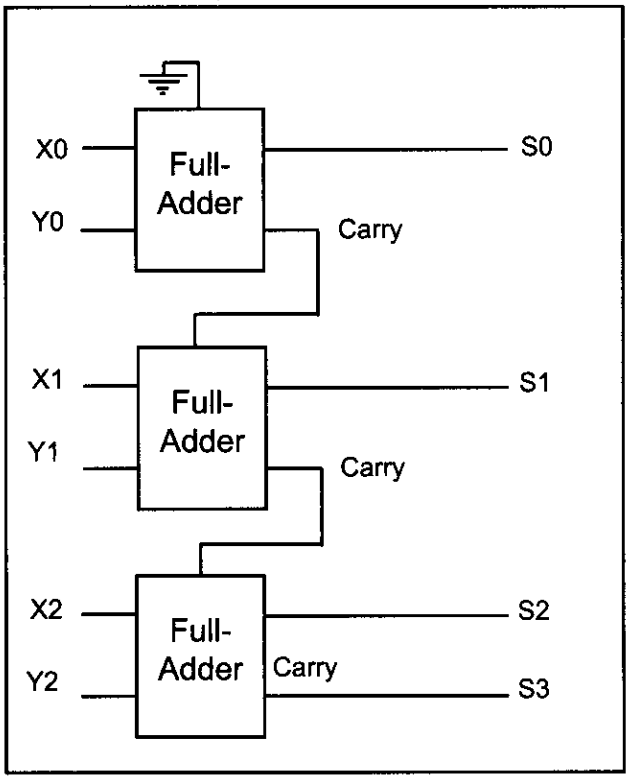
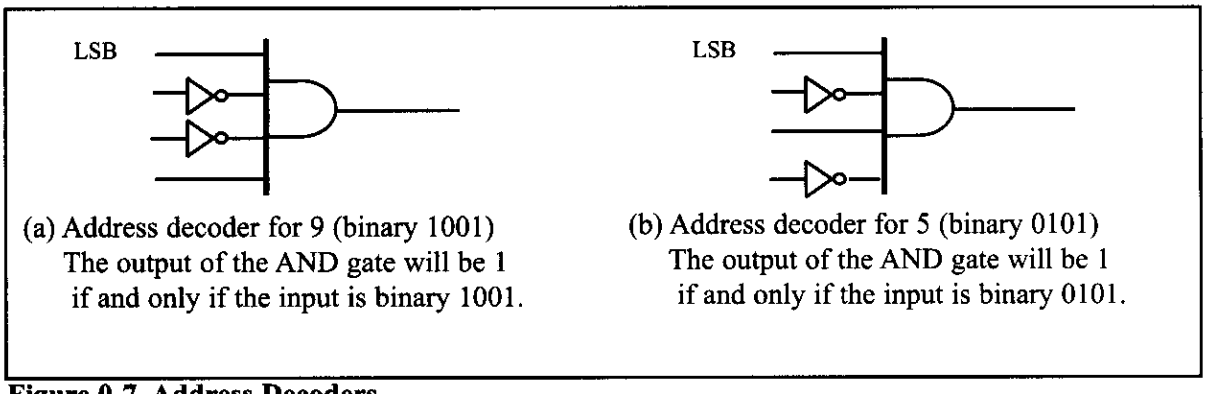


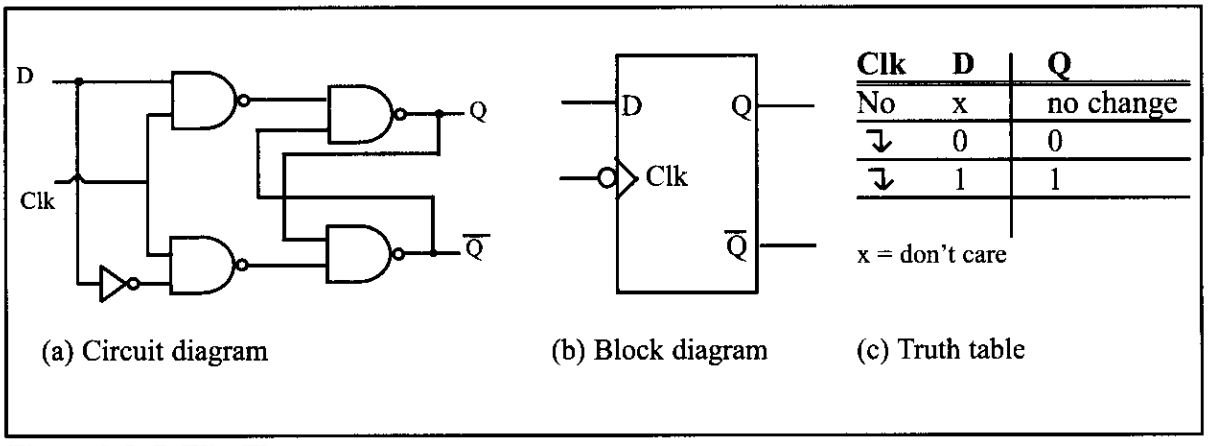
Figure 0-6. 3-Bit Adder Using Three Full-Adders



(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 one or more of its inputs is 1.
3. The logical operation _____ is often used to compare two inputs to determine whether they 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 predetermined binary address.

SECTION 0.3: SEMICONDUCTOR MEMORY

In this section we discuss various types of semiconductor memories and their characteristics such as capacity, organization, and access time. We will also show how the memory is connected to CPU. Before we embark on the subject of memory, it will be helpful to give an overview of computer organization and review some widely used terminology in computer literature.

Some important terminology

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

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.

A *kilobyte* is 2^{10} bytes, which is 1024 bytes. The abbreviation K is often used to represent kilobytes. 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. Figure 0-9 shows a block diagram of the internal organization of a computer.

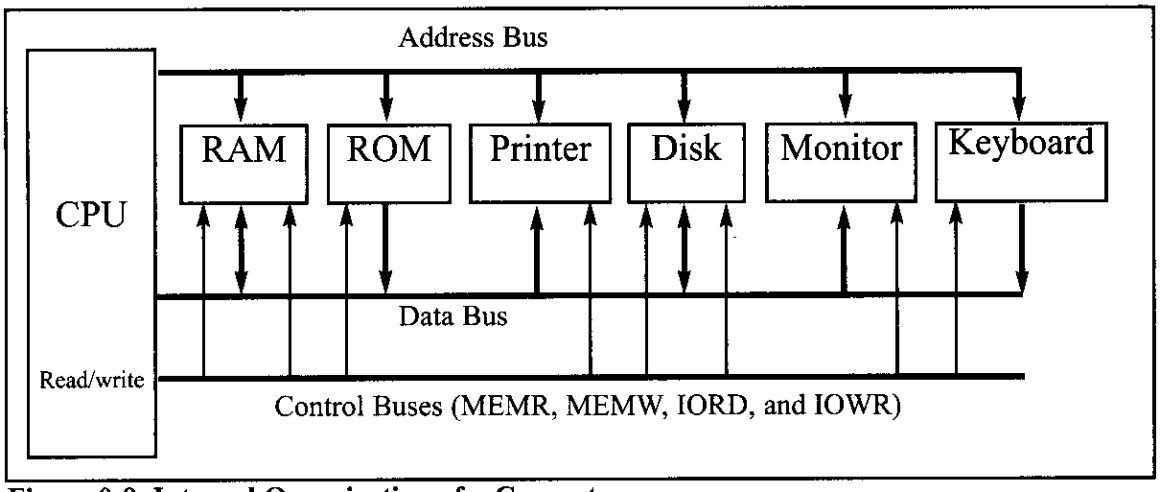


Figure 0-9. Internal Organization of a Computer

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 allows carrying information from place to place just as a street allows cars to carry 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 (in binary, of course) 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 information. Of the three buses, the address bus and data bus determine the capability of a given CPU.

More about the data bus

Because 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 0-9). 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 bits. Early personal computers such as Apple 2 used an 8-bit data bus, while supercomputers such as Cray used a 64-bit data bus. Data buses are bidirectional, because 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, because 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

Because 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 of addressable memory. Each location can have a maximum of 1 byte of data. This is because 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. Thus, the total accessible memory is 16 megabytes (2^{24} = 16 megabytes). In this example there would be 2^{24} locations, and because 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.

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 for displaying information on the video monitor. In contrast, RAM stores temporary information that 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 from the hard drive into RAM to be processed by the CPU. The CPU cannot get the information from the disk directly because the disk is too slow. In other words, the CPU first seeks the information to be processed from RAM (or ROM). Only if the data 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*. Next, we discuss various types of semiconductor memories and their characteristics such as capacity, organization, and access time.

Memory capacity

The number of bits that a semiconductor memory chip can store is called chip *capacity*. It can be in units of Kbits (kilobits), Mbits (megabits), and so on. This must be distinguished from the storage capacity of computer systems. While the memory capacity of a memory IC chip is always given in bits, the memory capacity of a computer system is given in bytes. For example, an article in a technical journal may state that the 128M chip has become popular. In that case, it is understood, although it is not mentioned, that 128M means 128 megabits since the article is referring to an IC memory chip. However, if an advertisement states that a computer comes with 128M memory, it is understood that 128M means 128 megabytes since it is referring to a computer system.

Memory organization

Memory chips are organized into a number of locations within the IC. Each location can hold 1 bit, 4 bits, 8 bits, or even 16 bits, depending on how it is designed internally. The number of bits that each location within the memory chip can hold is always equal to the number of data pins on the chip. How many locations exist inside a memory chip? That depends on the number of address pins. The number of locations within a memory IC always equals 2 to the power of the number of address pins. Therefore, the total number of bits that a memory chip can store is equal to the number of locations times the number of data bits per location. To summarize:

1. A memory chip contains 2^x locations, where x is the number of address pins.
2. Each location contains y bits, where y is the number of data pins on the chip.
3. The entire chip will contain $2^x \times y$ bits, where x is the number of address pins and y is the number of data pins on the chip.

Table 0-4: Powers of 2

x	2^x
10	1K
11	2K
12	4K
13	8K
14	16K
15	32K
16	64K
17	128K
18	256K
19	512K
20	1M
21	2M
22	4M
23	8M
24	16M
25	32M
26	64M
27	128M

Speed

One of the most important characteristics of a memory chip is the speed at which its data can be accessed. To access the data, the address is presented to the address pins, the READ pin is activated, and after a certain amount of time has elapsed, the data shows up at the data pins. The shorter this elapsed time, the better, and consequently, the more expensive the memory chip. The speed of the memory chip is commonly referred to as its *access time*. The access time of memory chips varies from a few nanoseconds to hundreds of nanoseconds, depending on the IC technology used in the design and fabrication process.

The three important memory characteristics of capacity, organization, and access time will be explored extensively in this chapter. Table 0-4 serves as a reference for the calculation of memory organization. Examples 0-12 and 0-13 demonstrate these concepts.

ROM (read-only memory)

ROM is a type of memory that does not lose its contents when the power is turned off. For this reason, ROM is also called *nonvolatile* memory. There are different types of read-only memory, such as PROM, EPROM, EEPROM, Flash EPROM, and mask ROM. Each is explained next.

PROM (programmable ROM) and OTP

PROM refers to the kind of ROM that the user can burn information into. In other words, PROM is a user-programmable memory. For every bit of the PROM, there exists a fuse. PROM is programmed by blowing the fuses. If the information burned into PROM is wrong, that PROM must be discarded since its internal fuses are blown permanently. For this reason, PROM is also referred to as

Example 0-12

A given memory chip has 12 address pins and 4 data pins. Find:
(a) the organization, and (b) the capacity.

Solution:

(a) This memory chip has 4,096 locations (2^{12} = 4,096), and each location can hold 4 bits of data. This gives an organization of 4,096 × 4, often represented as 4K × 4.
(b) The capacity is equal to 16K bits since there is a total of 4K locations and each location can hold 4 bits of data.

Example 0-13

A 512K memory chip has 8 pins for data. Find:
(a) the organization, and (b) the number of address pins for this memory chip.

Solution:

(a) A memory chip with 8 data pins means that each location within the chip can hold 8 bits of data. To find the number of locations within this memory chip, divide the capacity by the number of data pins. 512K/8 = 64K; therefore, the organization for this memory chip is 64K × 8.
(b) The chip has 16 address lines since 2^{16} = 64K.

OTP (one-time programmable). Programming ROM, also called *burning* ROM, requires special equipment called a ROM burner or ROM programmer.

EPROM (erasable programmable ROM) and UV-EPROM

EPROM was invented to allow making changes in the contents of PROM after it is burned. In EPROM, one can program the memory chip and erase it thousands of times. This is especially necessary during development of the prototype of a microprocessor-based project. A widely used EPROM is called UV-EPROM, where UV stands for ultraviolet. The only problem with UV-EPROM is that erasing its contents can take up to 20 minutes. All UV-EPROM chips have a window through which the programmer can shine ultraviolet (UV) radiation to erase the chip's contents. For this reason, EPROM is also referred to as UV-erasable EPROM or simply UV-EPROM. Figure 0-10 shows the pins for UV-EPROM chips.

To program a UV-EPROM chip, the following steps must be taken:

1. Its contents must be erased. To erase a chip, remove it from its socket on the system board and place it in EPROM erasure equipment to expose it to UV radiation for 15–20 minutes.
2. Program the chip. To program a UV-EPROM chip, place it in the ROM burner (programmer). To burn code or data into EPROM, the ROM burner uses 12.5 volts or higher, depending on the EPROM type. This voltage is referred

to as V_{PP} in the UV-EPROM data sheet.

3. Place the chip back into its socket on the system board.

As can be seen from the above steps, not only is there an EPROM programmer (burner), but there is also separate EPROM erasure equipment. The main problem, and indeed the major disadvantage of UV-EPROM, is that it cannot be erased and programmed while it is in the system board. To provide a solution to this problem, EEPROM was invented.

Notice the patterns of the IC numbers in Table 0-5. For example, part number 27128-25 refers to UV-EPROM that has a capacity of 128K bits and access time of 250 nanoseconds. The capacity of the memory chip is indicated in the part number and the access time is given with a zero dropped. See Example 0-14. In part numbers, C refers to CMOS technology. Notice that 27XX always refers to UV-EPROM chips. For a comprehensive list of available memory chips see the JAMECO (jameco.com) or JDR (jdr.com) catalogs.

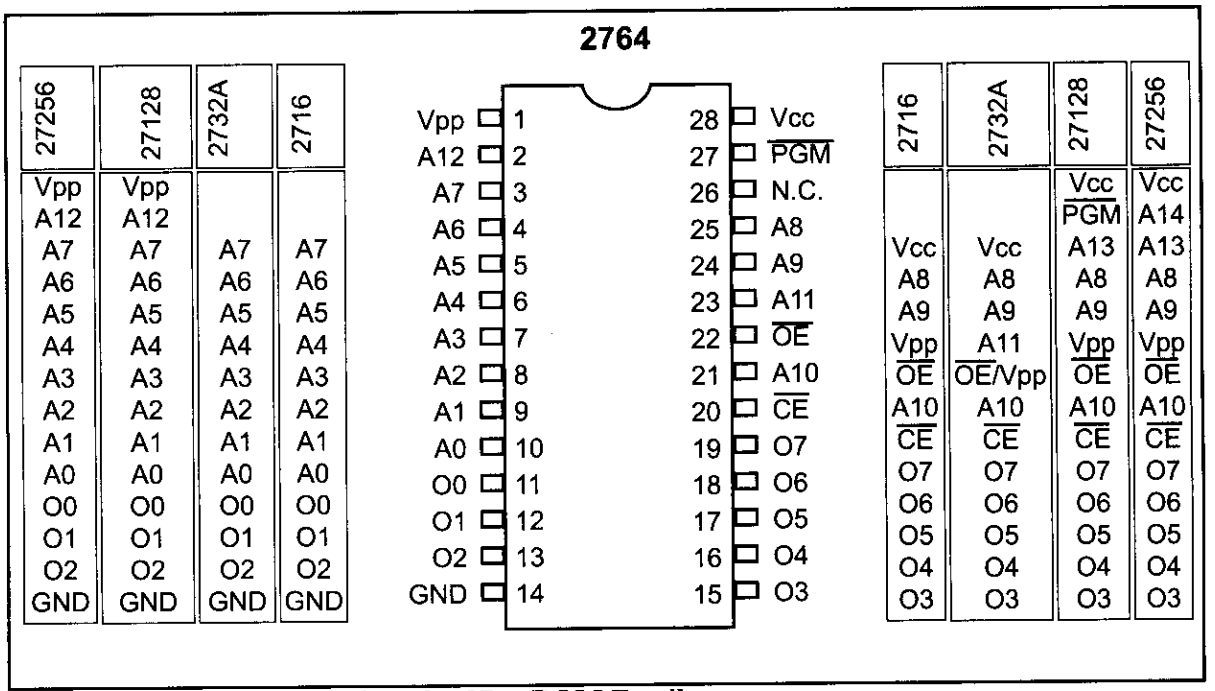


Figure 0-10. Pin Configurations for 27xx ROM Family

Example 0-14

For ROM chip 27128, find the number of data and address pins.

Solution:

The 27128 has a capacity of 128K bits. It has 16K × 8 organization (all ROMs have 8 data pins), which indicates that there are 8 pins for data and 14 pins for address (2^{14} = 16K).

Table 0-5: Some UV-EPROM Chips

Part #	Capacity	Org.	Access	Pins	V_{PP}
2716	16K	2K × 8	450 ns	24	25 V
2732	32K	4K × 8	450 ns	24	25 V
2732A-20	32K	4K × 8	200 ns	24	21 V
27C32-1	32K	4K × 8	450 ns	24	12.5 V CMOS
2764-20	64K	8K × 8	200 ns	28	21 V
2764A-20	64K	8K × 8	200 ns	28	12.5 V
27C64-12	64K	8K × 8	120 ns	28	12.5 V CMOS
27128-25	128K	16K × 8	250 ns	28	21 V
27C128-12	128K	16K × 8	120 ns	28	12.5 V CMOS
27256-25	256K	32K × 8	250 ns	28	12.5 V
27C256-15	256K	32K × 8	150 ns	28	12.5 V CMOS
27512-25	512K	64K × 8	250 ns	28	12.5 V
27C512-15	512K	64K × 8	150 ns	28	12.5 V CMOS
27C010-15	1024K	128K × 8	150 ns	32	12.5 V CMOS
27C020-15	2048K	256K × 8	150 ns	32	12.5 V CMOS
27C040-15	4096K	512K × 8	150 ns	32	12.5 V CMOS

EEPROM (electrically erasable programmable ROM)

EEPROM has several advantages over EPROM, such as the fact that its method of erasure is electrical and therefore instant, as opposed to the 20-minute erasure time required for UV-EPROM. In addition, in EEPROM one can select which byte to be erased, in contrast to UV-EPROM, in which the entire contents of ROM are erased. However, the main advantage of EEPROM is that one can program and erase its contents while it is still in the system board. It does not require physical removal of the memory chip from its socket. In other words, unlike UV-EPROM, EEPROM does not require an external erasure and programming device. To utilize EEPROM fully, the designer must incorporate the circuitry to program the EEPROM into the system board. In general, the cost per bit for EEPROM is much higher than for UV-EPROM.

Flash memory EPROM

Since the early 1990s, Flash EPROM has become a popular user-programmable memory chip, and for good reasons. First, the erasure of the entire contents takes less than a second, or one might say in a flash, hence its name, Flash memory. In addition, the erasure method is electrical, and for this reason it is sometimes referred to as Flash EEPROM. To avoid confusion, it is commonly called Flash memory. The major difference between EEPROM and Flash memory is that when Flash memory's contents are erased, the entire device is erased, in contrast to EEPROM, where one can erase a desired byte. Although in many Flash memories recently made available the contents are divided into blocks and the erasure can be done block by block, unlike EEPROM, Flash memory has no byte erasure option. Because Flash memory can be programmed while it is in its socket on the system board, it is widely used to upgrade the BIOS ROM of the PC. Some designers believe that Flash memory will replace the hard disk as a mass storage medium.

Table 0-6: Some EEPROM and Flash Chips

EEPROMs

Part No.	Capacity	Org.	Speed	Pins	V_{PP}
2816A-25	16K	2K × 8	250 ns	24	5 V
2864A	64K	8K × 8	250 ns	28	5 V
28C64A-25	64K	8K × 8	250 ns	28	5 V CMOS
28C256-15	256K	32K × 8	150 ns	28	5 V
28C256-25	256K	32K × 8	250 ns	28	5 V CMOS

Flash

Part No.	Capacity	Org.	Speed	Pins	V_{PP}
28F256-20	256K	32K × 8	200 ns	32	12 V CMOS
28F010-15	1024K	128K × 8	150 ns	32	12 V CMOS
28F020-15	2048K	256K × 8	150 ns	32	12 V CMOS

This would increase the performance of the computer tremendously, since Flash memory is semiconductor memory with access time in the range of 100 ns compared with disk access time in the range of tens of milliseconds. For this to happen, Flash memory's program/erase cycles must become infinite, just like hard disks. Program/erase cycle refers to the number of times that a chip can be erased and reprogrammed before it becomes unusable. At this time, the program/erase cycle is 100,000 for Flash and EEPROM, 1000 for UV-EPROM, and infinite for RAM and disks. See Table 0-6 for some sample chips.

Mask ROM

Mask ROM refers to a kind of ROM in which the contents are programmed by the IC manufacturer. In other words, it is not a user-programmable ROM. The term *mask* is used in IC fabrication. Since the process is costly, mask ROM is used when the needed volume is high (hundreds of thousands) and it is absolutely certain that the contents will not change. It is common practice to use UV-EPROM or Flash for the development phase of a project, and only after the code/data have been finalized is the mask version of the product ordered. The main advantage of mask ROM is its cost, since it is significantly cheaper than other kinds of ROM, but if an error is found in the data/code, the entire batch must be thrown away. It must be noted that all ROM memories have 8 bits for data pins; therefore, the organization is ×8.

RAM (random access memory)

RAM memory is called *volatile* memory since cutting off the power to the IC results in the loss of data. Sometimes RAM is also referred to as RAWM (read and write memory), in contrast to ROM, which cannot be written to. There are three types of RAM: static RAM (SRAM), NV-RAM (nonvolatile RAM), and dynamic RAM (DRAM). Each is explained separately.

SRAM (static RAM)

Storage cells in static RAM memory are made of flip-flops and therefore do not require refreshing in order to keep their data. This is in contrast to DRAM, discussed below. The problem with the use of flip-flops for storage cells is that each cell requires at least 6 transistors to build, and the cell holds only 1 bit of data. In recent years, the cells have been made of 4 transistors, which still is too many. The use of 4-transistor cells plus the use of CMOS technology has given birth to a high-capacity SRAM, but its capacity is far below DRAM. Figure 0-11 shows the pin diagram for an SRAM chip.

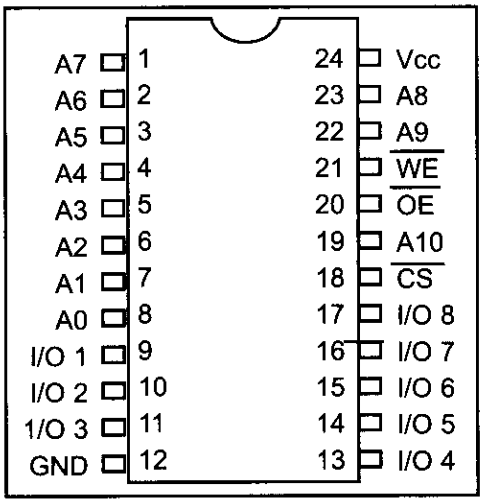


Figure 0-11. 2K × 8 SRAM Pins

The following is a description of the 6116 SRAM pins.

A0–A10 are for address inputs, where 11 address lines gives $2^{11} = 2K$.
WE (write enable) is for writing data into SRAM (active low).
OE (output enable) is for reading data out of SRAM (active low)
CS (chip select) is used to select the memory chip.
I/O0–I/O7 are for data I/O, where 8-bit data lines give an organization of 2K × 8.

The functional diagram for the 6116 SRAM is given in Figure 0-12.

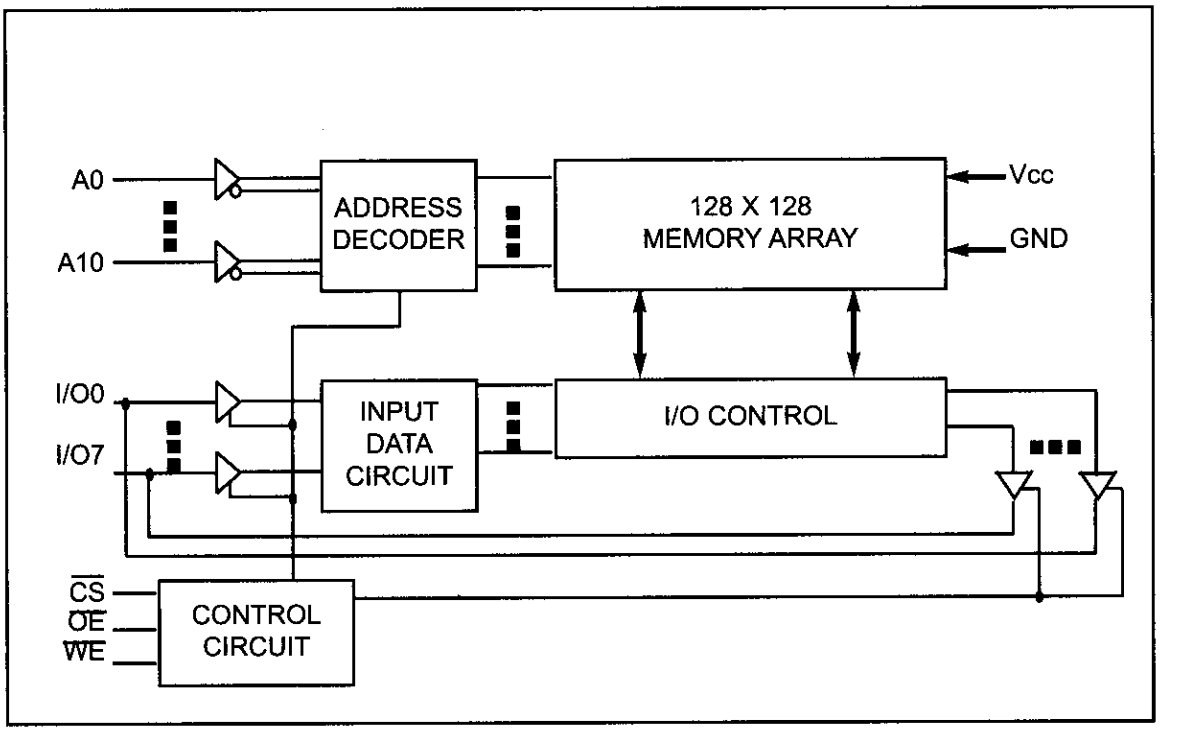


Figure 0-12. Functional Block Diagram for 6116 SRAM

Figure 0-13 shows the following steps to write data into SRAM.

1. Provide the addresses to pins A0–A10.
2. Activate the CS pin.
3. Make WE = 0 while RD = 1.
4. Provide the data to pins I/O0–I/O7.
5. Make WE = 1 and data will be written into SRAM on the positive edge of the WE signal.

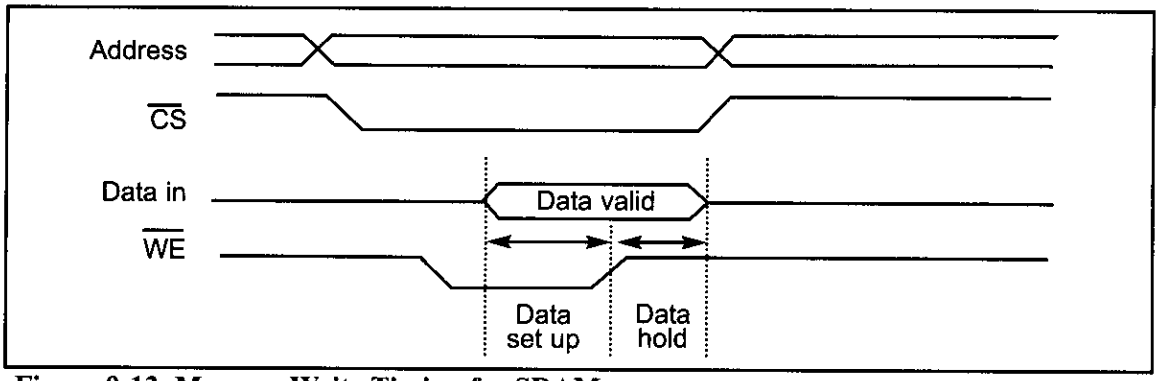


Figure 0-13. Memory Write Timing for SRAM

The following are steps to read data from SRAM. See Figure 0-14.

1. Provide the addresses to pins A0–A10. This is the start of the access time (t_{AA}).
2. Activate the CS pin.
3. While WE = 1, a high-to-low pulse on the OE pin will read the data out of the chip.

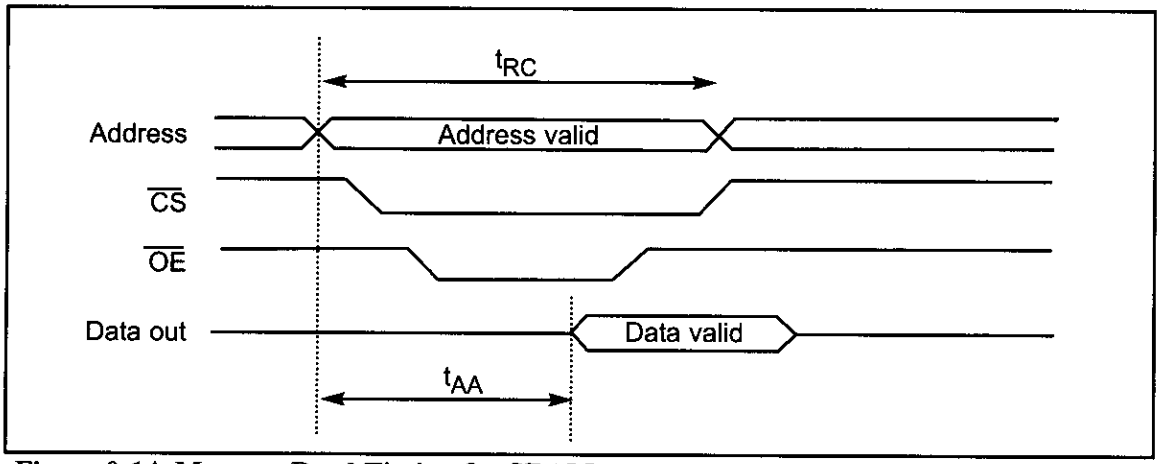


Figure 0-14. Memory Read Timing for SRAM

NV-RAM (nonvolatile RAM)

Whereas SRAM is volatile, there is a new type of nonvolatile RAM called NV-RAM. Like other RAMs, it allows the CPU to read and write to it, but when the power is turned off the contents are not lost. NV-RAM combines the best of RAM and ROM: the read and write ability of RAM, plus the nonvolatility of ROM. To retain its contents, every NV-RAM chip internally is made of the following components:

1. It uses extremely power-efficient (very low-power consumption) SRAM cells built out of CMOS.

Table 0-7: Some SRAM and NV-RAM Chips

SRAM

Part No.	Capacity	Org.	Speed	Pins	V_{PP}
6116P-1	16K	2K × 8	100 ns	24	CMOS
6116P-2	16K	2K × 8	120 ns	24	CMOS
6116P-3	16K	2K × 8	150 ns	24	CMOS
6116LP-1	16K	2K × 8	100 ns	24	Low-power CMOS
6116LP-2	16K	2K × 8	120 ns	24	Low-power CMOS
6116LP-3	16K	2K × 8	150 ns	24	Low-power CMOS
6264P-10	64K	8K × 8	100 ns	28	CMOS
6264LP-70	64K	8K × 8	70 ns	28	Low-power CMOS
6264LP-12	64K	8K × 8	120 ns	28	Low-power CMOS
62256LP-10	256K	32K × 8	100 ns	28	Low-power CMOS
62256LP-12	256K	32K × 8	120 ns	28	Low-power CMOS

NV-RAM from Dallas Semiconductor

Part No.	Capacity	Org.	Speed	Pins	V_{PP}
DS1220Y-150	16K	2K × 8	150 ns	24	
DS1225AB-150	64K	8K × 8	150 ns	28	
DS1230Y-85	256K	32K × 8	85 ns	28	

2. It uses an internal lithium battery as a backup energy source.
3. It uses an intelligent control circuitry. The main job of this control circuitry is to monitor the V_{CC} pin constantly to detect loss of the external power supply. If the power to the V_{CC} pin falls below out-of-tolerance conditions, the control circuitry switches automatically to its internal power source, the lithium battery. The internal lithium power source is used to retain the NV-RAM contents only when the external power source is off.

It must be emphasized that all three of the components above are incorporated into a single IC chip, and for this reason nonvolatile RAM is a very expensive type of RAM as far as cost per bit is concerned. Offsetting the cost, however, is the fact that it can retain its contents up to ten years after the power has been turned off and allows one to read and write in exactly the same way as SRAM. Table 0-7 shows some examples of SRAM and NV-RAM parts.

DRAM (dynamic RAM)

Since the early days of the computer, the need for huge, inexpensive read/write memory has been a major preoccupation of computer designers. In 1970, Intel Corporation introduced the first dynamic RAM (random access memory). Its density (capacity) was 1024 bits and it used a capacitor to store each bit. Using a capacitor to store data cuts down the number of transistors needed to build the cell; however, it requires constant refreshing due to leakage. This is in contrast to SRAM (static RAM), whose individual cells are made of flip-flops. Since each bit in SRAM uses a single flip-flop, and each flip-flop requires six transistors,

SRAM has much larger memory cells and consequently lower density. The use of capacitors as storage cells in DRAM results in much smaller net memory cell size.

The advantages and disadvantages of DRAM memory can be summarized as follows. The major advantages are high density (capacity), cheaper cost per bit, and lower power consumption per bit. The disadvantage is that it must be refreshed periodically because the capacitor cell loses its charge; furthermore, while DRAM is being refreshed, the data cannot be accessed. This is in contrast to SRAM's flip-flops, which retain data as long as the power is on, do not need to be refreshed, and whose contents can be accessed at any time. Since 1970, the capacity of DRAM has exploded. After the 1K-bit (1024) chip came the 4K-bit in 1973, and then the 16K chip in 1976. The 1980s saw the introduction of 64K, 256K, and finally 1M and 4M memory chips. The 1990s saw 16M, 64M, 256M, and the beginning of 1G-bit DRAM chips. In the 2000s, 2G-bit chips are standard, and as the fabrication process gets smaller, larger memory chips will be rolling off the manufacturing line. Keep in mind that when talking about IC memory chips, the capacity is always assumed to be in bits. Therefore, a 1M chip means a 1-megabit chip and a 256K chip means a 256K-bit memory chip. However, when talking about the memory of a computer system, it is always assumed to be in bytes.

Packaging issue in DRAM

In DRAM there is a problem of packing a large number of cells into a single chip with the normal number of pins assigned to addresses. For example, a 64K-bit chip (64K × 1) must have 16 address lines and 1 data line, requiring 16 pins to send in the address if the conventional method is used. This is in addition to V_{CC} power, ground, and read/write control pins. Using the conventional method of data access, the large number of pins defeats the purpose of high density and small packaging, so dearly cherished by IC designers. Therefore, to reduce the number of pins needed for addresses, multiplexing/demultiplexing is used. The method used is to split the address in half and send in each half of the address through the same pins, thereby requiring fewer address pins. Internally, the DRAM structure is divided into a square of rows and columns. The first half of the address is called the row and the second half is called the column. For example, in the case of DRAM of 64K × 1 organization, the first half of the address is sent in through the 8 pins A0–A7, and by activating RAS (row address strobe), the internal latches inside DRAM grab the first half of the address. After that, the second half of the address is sent in through the same pins, and by activating CAS (column address strobe), the internal latches inside DRAM latch the second half of the address. This results in using 8 pins for addresses plus RAS and CAS, for a total of 10 pins, instead of the 16 pins that would be required without multiplexing. To access a bit of data from DRAM, both row and column addresses must be provided. For this concept to work, there must be a 2-by-1 multiplexer outside the DRAM circuitry and a demultiplexer inside every DRAM chip. Due to the complexities associated with DRAM interfacing (RAS, CAS, the need for multiplexer and refreshing circuitry), some DRAM controllers are designed to make DRAM interfacing much easier. However, many small microcontroller-based projects that do not require much RAM (usually less than 64K bytes) use SRAM of types EEPROM and NV-RAM, instead of DRAM.

DRAM organization

In the discussion of ROM, we noted that all of these chips have 8 pins for data. This is not the case for DRAM memory chips, which can have ×1, ×4, ×8, or ×16 organizations. See Example 0-15 and Table 0-8.

In memory chips, the data pins are also called I/O. In some DRAMs there are separate D_{in} and D_{out} pins. Figure 0-15 shows a 256K × 1 DRAM chip with pins A0–A8 for address, RAS and CAS, WE (write enable), and data in and data out, as well as power and ground.

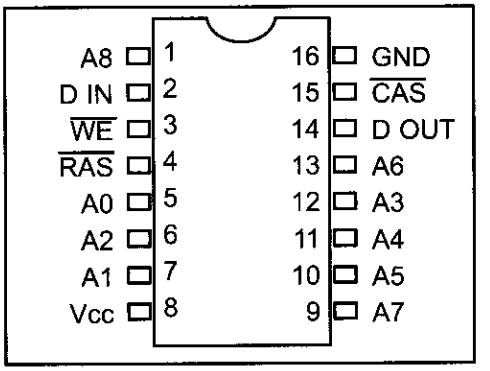


Figure 0-15. 256K × 1 DRAM

Example 0-15

Discuss the number of pins set aside for addresses in each of the following memory chips. (a) 16K × 4 DRAM (b) 16K × 4 SRAM

Solution:

Since 2^{14} = 16K:

(a) For DRAM we have 7 pins (A0–A6) for the address pins and 2 pins for RAS and CAS.

(b) For SRAM we have 14 pins for address and no pins for RAS and CAS since they are associated only with DRAM. In both cases we have 4 pins for the data bus.

Table 0-8: Some DRAMs

Part No.	Speed	Capacity	Org.	Pins
4164-15	150 ns	64K	64K × 1	16
41464-8	80 ns	256K	64K × 4	18
41256-15	150 ns	256K	256K × 1	16
41256-6	60 ns	256K	256K × 1	16
414256-10	100 ns	1M	256K × 4	20
511000P-8	80 ns	1M	1M × 1	18
514100-7	70 ns	4M	4M × 1	20

Memory address decoding

Next we discuss address decoding. The CPU provides the address of the data desired, but it is the job of the decoding circuitry to locate the selected memory block. To explore the concept of decoding circuitry, we look at various methods used in decoding the addresses. In this discussion we use SRAM or ROM for the sake of simplicity.

Memory chips have one or more pins called CS (chip select), which must be activated for the memory's contents to be accessed. Sometimes the chip select is also referred to as chip enable (CE). In connecting a memory chip to the CPU,

note the following points.

1. The data bus of the CPU is connected directly to the data pins of the memory chip.
2. Control signals RD (read) and WR (memory write) from the CPU are connected to the OE (output enable) and WE (write enable) pins of the memory chip, respectively.
3. In the case of the address buses, while the lower bits of the addresses from the CPU go directly to the memory chip address pins, the upper ones are used to activate the CS pin of the memory chip. It is the CS pin that along with RD/WR allows the flow of data in or out of the memory chip. No data can be written into or read from the memory chip unless CS is activated.

As can be seen from the data sheets of SRAM and ROM, the CS input of a memory chip is normally active low and is activated by the output of the memory decoder. Normally memories are divided into blocks, and the output of the decoder selects a given memory block. There are three ways to generate a memory block selector: (a) using simple logic gates, (b) using the 74LS138, or (c) using programmable logics such as CPLD and FPGA. Each method is described below.

Simple logic gate address decoder

The simplest method of constructing decoding circuitry is the use of a NAND gate. The output of a NAND gate is active low, and the CS pin is also active low, which makes them a perfect match. In cases where the CS input is active high, an AND gate must be used. Using a combination of NAND gates and inverters, one can decode any address range. An example of this is shown in Figure 0-16, which shows that A15–A12 must be 0011 in order to select the chip. This results in the assignment of addresses 3000H to 3FFFH to this memory chip.

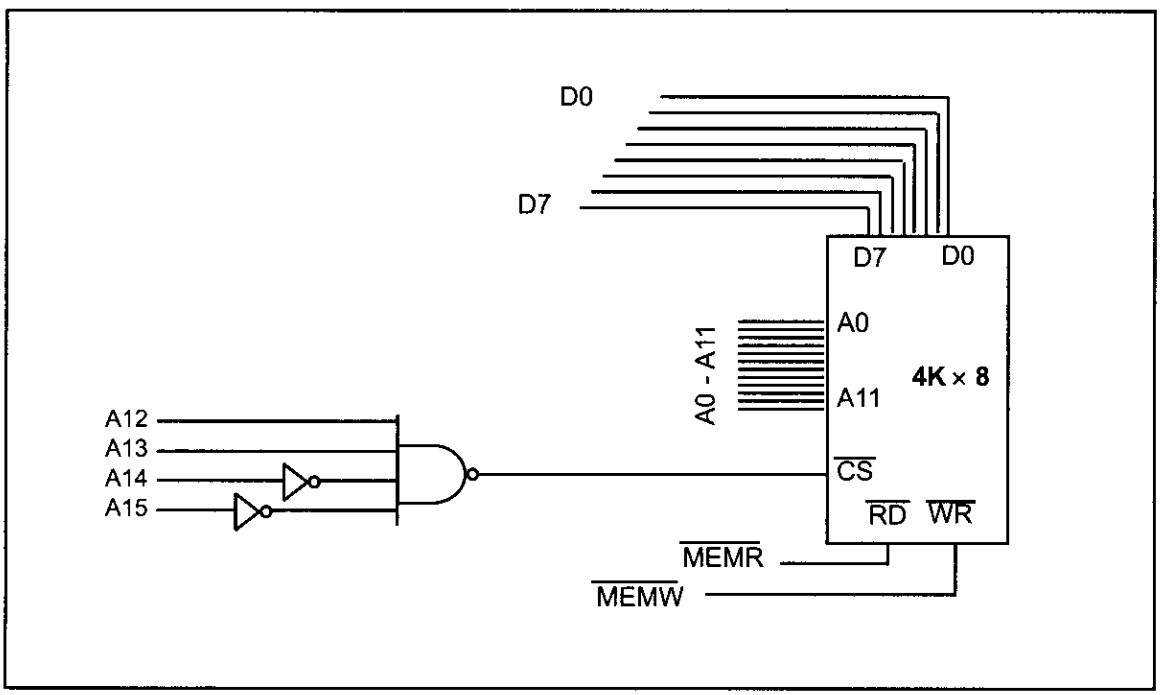
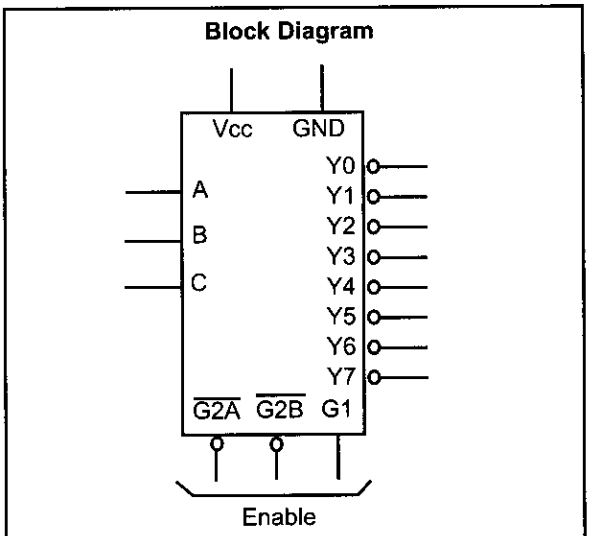


Figure 0-16. Logic Gate as Decoder

Using the 74LS138 3-8 decoder

This used to be one of the most widely used address decoders. The 3 inputs A, B, and C generate 8 active-low outputs Y0–Y7. See Figure 0-17. Each Y output is connected to CS of a memory chip, allowing control of 8 memory blocks by a single 74LS138. In the 74LS138, where A, B, and C select which output is activated, there are three additional inputs, G2A, G2B, and G1. G2A and G2B are both active low, and G1 is active high. If any one of the inputs G1, G2A, or G2B is not connected to an address signal (sometimes they are connected to a control signal), they must be activated permanently by either V_{CC} or ground, depending on the activation level. Example 0-16 shows the design and the address range calculation for the 74LS138 decoder.



Function Table

Inputs			Outputs
Enable		Select	
G1	G2	C B A	Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7
X	H	X X X	H H H H H H H H
L	X	X X X	H H H H H H H H
H	L	L L L	L H H H H H H H
H	L	L L H	H L H H H H H H
H	L	L H L	H H L H H H H H
H	L	L H H	H H H L H H H H
H	L	H L L	H H H H L H H H
H	L	H L H	H H H H H L H H
H	L	H H L	H H H H H H L H
H	L	H H H	H H H H H H H L

Figure 0-17. 74LS138 Decoder

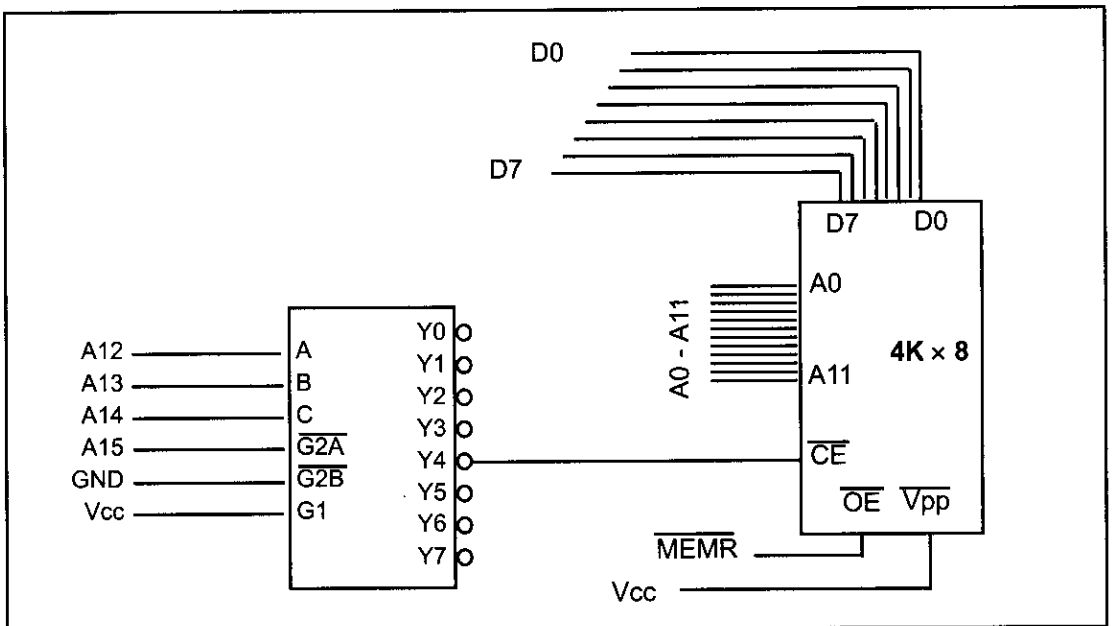


Figure 0-18. Using 74LS138 as Decoder

Example 0-16

Looking at the design in Figure 0-18, find the address range for the following:
(a) Y4, (b) Y2, and (c) Y7.

Solution:

(a) The address range for Y4 is calculated as follows.

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1

The above shows that the range for Y4 is 4000H to 4FFFH. In Figure 0-18, notice that A15 must be 0 for the decoder to be activated. Y4 will be selected when A14 A13 A12 = 100 (4 in binary). The remaining A11–A0 will be 0 for the lowest address and 1 for the highest address.

(b) The address range for Y2 is 2000H to 2FFFH.

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1

(c) The address range for Y7 is 7000H to 7FFFH.

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Using programmable logic as an address decoder

Other widely used decoders are programmable logic chips such as PAL, GAL, and FPGA chips. One disadvantage of these chips is that they require PAL/GAL/FPGA software and a burner (programmer), whereas the 74LS138 needs neither of these. The advantage of these chips is that they can be programmed for any combination of address ranges, and so are much more versatile. This plus the fact that PAL/GAL/FPGA chips have 10 or more inputs (in contrast to 6 in the 74138) means that they can accommodate more address inputs.

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. The speed of semiconductor memory is in the range of
    (a) microseconds (b) milliseconds
    (c) nanoseconds (d) picoseconds
11. Find the organization and chip capacity for each ROM with the indicated number of address and data pins.
    (a) 14 address, 8 data (b) 16 address, 8 data (c) 12 address, 8 data
12. Find the organization and chip capacity for each RAM with the indicated number of address and data pins.
    (a) 11 address, 1 data SRAM (b) 13 address, 4 data SRAM
    (c) 17 address, 8 data SRAM (d) 8 address, 4 data DRAM
    (e) 9 address, 1 data DRAM (f) 9 address, 4 data DRAM
13. Find the capacity and number of pins set aside for address and data for memory chips with the following organizations.
    (a) 16K × 4 SRAM (b) 32K × 8 EPROM (c) 1M × 1 DRAM
    (d) 256K × 4 SRAM (e) 64K × 8 EEPROM (f) 1M × 4 DRAM
14. Which of the following is (are) volatile memory?
    (a) EEPROM (b) SRAM (c) DRAM (d) NV-RAM
15. A given memory block uses addresses 4000H–7FFFH. How many kilobytes is this memory block?
16. The 74138 is a(n) _____ by _____ decoder.
17. In the 74138 give the status of G2A and G2B for the chip to be enabled.
18. In the 74138 give the status of G1 for the chip to be enabled.
19. In Example 0-16, what is the range of addresses assigned to Y5?

SECTION 0.4: CPU ARCHITECTURE

In this section we will examine the inside of a CPU. Then, we will compare the Harvard and von Neumann architectures.

Inside CPU

A program stored in memory provides instructions to the CPU to perform an action. See Figure 0-19. The action can simply be adding data such as payroll data or controlling a machine such as a robot. The function of the CPU is 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,

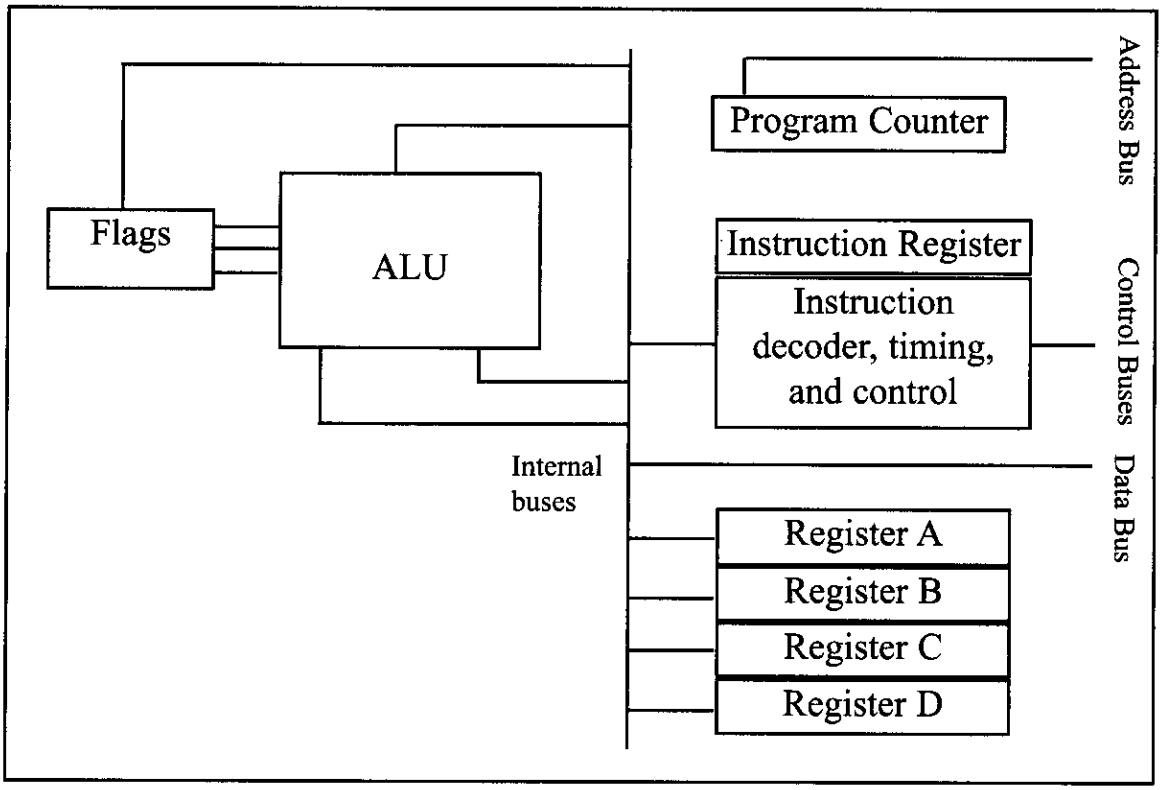


Figure 0-19. Internal Block Diagram of a CPU

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. The contents of the program counter 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 CPUs

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 the 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 these opera-

tions 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 the byte in the next memory location into register A of the CPU. 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 the CPU fetches code 04H. After decoding, the CPU knows that it must add the byte sitting at the next address (1403) to the contents of register A. After the CPU brings the value (in this case, 42H) into register A, 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 into 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 again is 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. Without 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.

Harvard and von Neumann architectures

Every microprocessor must have memory space to store program (code) and data. While code provides instructions to the CPU, the data provides the information to be processed. The CPU uses buses (wire traces) to access the code ROM and data RAM memory spaces. The early computers used the same bus for accessing both the code and data. Such an architecture is commonly referred to as *von Neumann (Princeton) architecture*. That means for von Neumann computers, the process of accessing the code or data could cause them to get in each other's way and slow down the processing speed of the CPU, because each had to wait for the other to finish fetching. To speed up the process of program execution, some CPUs use what is called *Harvard architecture*. In Harvard architecture, we have separate buses for the code and data memory. See Figure 0-20. That means that we need four sets of buses: (1) a set of data buses for carrying data into and out of the CPU, (2) a set of address buses for accessing the data, (3) a set of data buses for carrying code into the CPU, and (4) an address bus for accessing the code. See Figure 0-20. This is easy to implement inside an IC chip such as a microcontroller where both ROM code and data RAM are internal (on-chip) and distances are on the micron and millimeter scale. But implementing Harvard architecture for systems such as x86 IBM PC-type computers is very expensive because the RAM and ROM that hold code and data are external to the CPU. Separate wire traces for data and code on the motherboard will make the board large and expensive. For example, for a Pentium microprocessor with a 64-bit data bus and a 32-bit address bus we will need about 100 wire traces on the motherboard if it is von Neumann architecture (96 for address and data, plus a few others for control signals of read and write and so on). But the number of wire traces will double to 200 if we use Harvard architecture. Harvard architecture will also necessitate a large number of pins coming out of the microprocessor itself. For this reason you do not see Harvard architecture implemented in the world of PCs and workstations. This is also the reason that microcontrollers such as AVR use Harvard architecture internally, but they still use von Neumann architecture if they need external memory for code and data space. The von Neumann architecture was developed at Princeton University, while the Harvard architecture was the work of Harvard University.

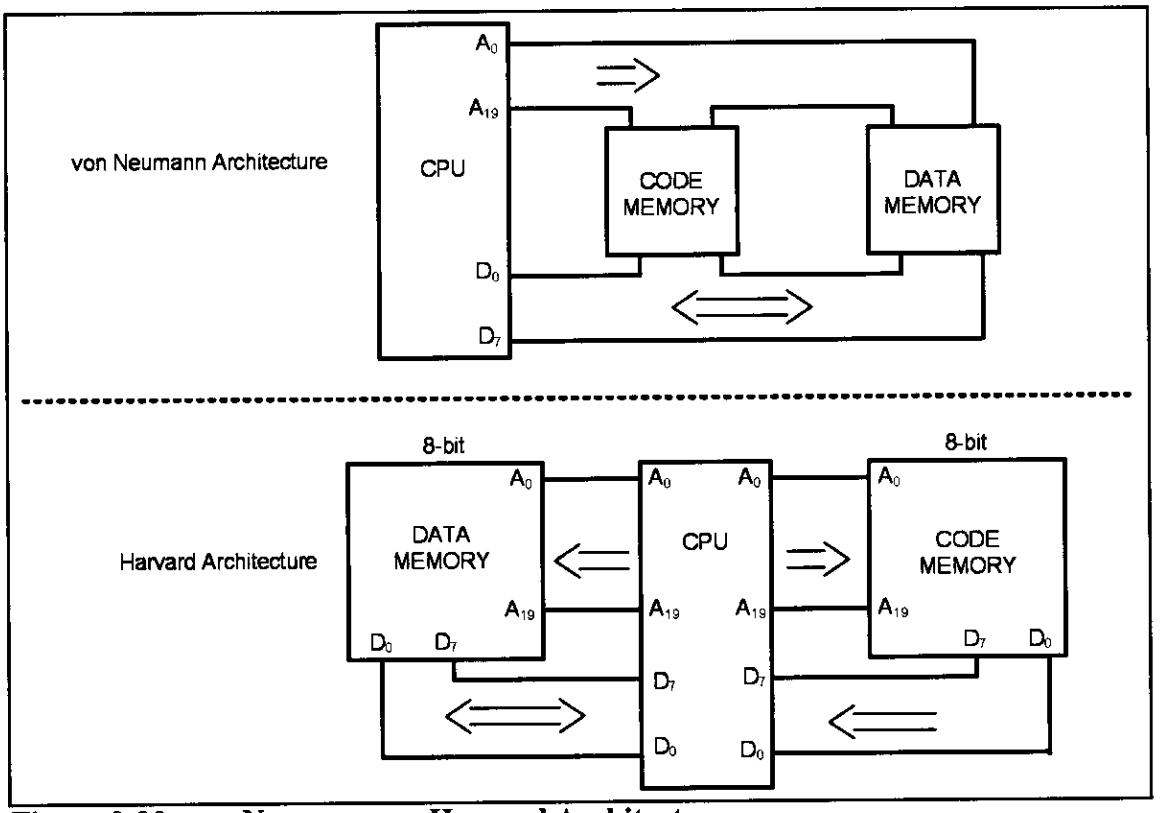


Figure 0-20. von Neumann vs. Harvard Architecture

Review Questions

1. What does "ALU" stand for? What is its purpose?
2. How are registers used in computer systems?
3. What is the purpose of the program counter?
4. What is the purpose of the instruction decoder?
5. True or false. Harvard architecture uses the same address and data buses to fetch both code and data.

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. 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 AND, OR, and inverter logic gates 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.

This chapter provided an overview of semiconductor memories. Types of memories were compared in terms of their capacity, organization, and access time. ROM (read-only memory) is nonvolatile memory typically used to store programs in embedded systems. The relative advantages of various types of ROM were described, including PROM, EPROM, UV-EPROM, EEPROM, Flash memory EPROM, and mask ROM.

Address decoding techniques using simple logic gates, decoders, and programmable logic were covered.

The computer organization and the internals of the CPU were also covered.

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 strings:
    "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: SEMICONDUCTOR MEMORY

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 kilobytes is 1 meg?
    (f) What is the exact value (in decimal) of 1 gigabyte?
    (g) How many kilobytes is 1 gigabyte?
    (h) How many megs is 1 gigabyte?
    (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) x86 PC with a 16-bit data bus
    (c) x86 PC 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 megs)
(c) 32-bit address bus (in megabytes and gigabytes)
(d) 48-bit address bus (in megabytes, gigabytes, and terabytes)

27. Of the data bus and address bus, which is unidirectional and which is bidirectional?
28. What is the difference in capacity between a 4M memory chip and 4M of computer memory?
29. True or false. The more address pins, the more memory locations are inside the chip. (Assume that the number of data pins is fixed.)
30. True or false. The more data pins, the more each location inside the chip will hold.
31. True or false. The more data pins, the higher the capacity of the memory chip.
32. True or false. The more data pins and address pins, the greater the capacity of the memory chip.
33. The speed of a memory chip is referred to as its _____.
34. True or false. The price of memory chips varies according to capacity and speed.
35. The main advantage of EEPROM over UV-EPROM is _____.
36. True or false. SRAM has a larger cell size than DRAM.
37. Which of the following, EPROM, DRAM, or SRAM, must be refreshed periodically?
38. Which memory is used for PC cache?
39. Which of the following, SRAM, UV-EPROM, NV-RAM, or DRAM, is volatile memory?
40. RAS and CAS are associated with which type of memory?
    (a) EPROM (b) SRAM (c) DRAM (d) all of the above
41. Which type of memory needs an external multiplexer?
    (a) EPROM (b) SRAM (c) DRAM (d) all of the above
42. Find the organization and capacity of memory chips with the following pins.
    (a) EEPROM A0–A14, D0–D7
    (b) UV-EPROM A0–A12, D0–D7
    (c) SRAM A0–A11, D0–D7
    (d) SRAM A0–A12, D0–D7
    (e) DRAM A0–A10, D0
    (f) SRAM A0–A12, D0
    (g) EEPROM A0–A11, D0–D7
    (h) UV-EPROM A0–A10, D0–D7
    (i) DRAM A0–A8, D0–D3
    (j) DRAM A0–A7, D0–D7
43. Find the capacity, address, and data pins for the following memory organizations.
    (a) 16K × 8 ROM
    (b) 32K × 8 ROM
    (c) 64K × 8 SRAM
    (d) 256K × 8 EEPROM
    (e) 64K × 8 ROM
    (f) 64K × 4 DRAM
    (g) 1M × 8 SRAM
    (h) 4M × 4 DRAM
    (i) 64K × 8 NV-RAM
44. Find the address range of the memory design in the diagram.
45. Using NAND gates and inverters, design decoding circuitry for the address range 2000H–2FFFH.
46. Find the address range for Y0, Y3, and Y6 of the 74LS138 for the diagrammed

design.

47. Using the 74138, design the memory decoding circuitry in which the memory block controlled by Y0 is in the range 0000H to 1FFFH. Indicate the size of the memory block controlled by each Y.
48. Find the address range for Y3, Y6, and Y7 in Problem 47.

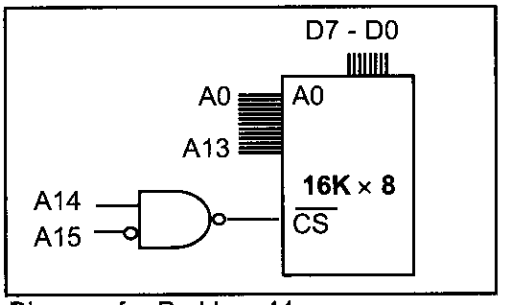


Diagram for Problem 44

49. Using the 74138, design memory decoding circuitry in which the memory block controlled by Y0 is in the 0000H to 3FFFH space. Indicate the size of the memory block controlled by each Y.
50. Find the address range for Y1, Y2, and Y3 in Problem 49.

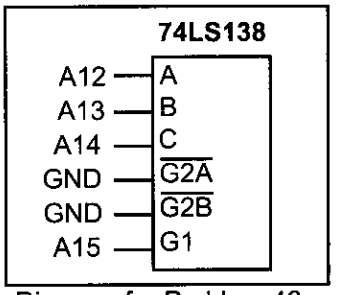


Diagram for Problem 46

SECTION 0.4: CPU AND HARVARD ARCHITECTURE

51. Which register of the CPU holds the address of the instruction to be fetched?
52. Which section of the CPU is responsible for performing addition?
53. 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: SEMICONDUCTOR MEMORY

1. 24,576
2. Random access memory; it is used for temporary storage of programs that the CPU is run-

ning, 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. c
11. (a) 16K × 8, 128K bits (b) 64K × 8, 512K (c) 4K × 8, 32K
12. (a) 2K × 1, 2K bits (b) 8K × 4, 32K (c) 128K × 8, 1M
    (d) 64K × 4, 256K (e) 256K × 1, 256K (f) 256K × 4, 1M
13. (a) 64K bits, 14 address, and 4 data (b) 256K, 15 address, and 8 data
    (c) 1M, 10 address, and 1 data (d) 1M, 18 address, and 4 data
    (e) 512K, 16 address, and 8 data (f) 4M, 10 address, and 4 data
14. b, c
15. 16K bytes
16. 3, 8
17. Both must be low.
18. G1 must be high.
19. 5000H–5FFFH

SECTION 0.4: CPU ARCHITECTURE

1. Arithmetic/logic unit; it performs all arithmetic and logic operations.
2. They are used for temporary storage of information.
3. It holds the address of the next instruction to be executed.
4. It tells the CPU what actions to perform for each instruction.
5. False