PC ARCHITECTURE from ASSEMBLY LANGUAGE to C

DAVID HERGERT
NANCY THIBEAULT

PC ARCHITECTURE FROM ASSEMBLY LANGUAGE TO C

DAVID HERGERT
Miami University

NANCY THIBEAULT
Miami University

Prentice Hall

Upper Saddle River, New Jersey *Columbus, Ohio*

Library of Congress Cataloging-in-Publication Data

Hergert, David.
PC architecture from assembly language to C / David Hergert, Nancy Thibeault.
p. cm.
Includes index.
ISBN 0-13-653775-8
1. Microcomputers. 2. Microprocessors. 3. Computer architecture. 4. Assembler language (Computer program language) 5. C (Computer program language) I. Thibeault, Nancy. II. Title.
QA76.5.H4447 1998
004.16--DC21 97-20688
CIP

Cover art: © 1997 Photo Disc, Inc.
Editor: Charles E. Stewart, Jr.
Production Editor: Alexandrina Benedicto Wolf
Cover Design Coordinator: Karrie M. Converse
Cover Designer: Russ Maselli
Production Manager: Pamela D. Bennett
Marketing Manager: Debbie Yarnell
Editorial/Production Supervision: Custom Editorial Productions, Inc.

This book was set in Times Roman by Custom Editorial Productions, Inc. and was printed and bound by R.R. Donnelley and Sons Company. The cover was printed by Phoenix Color Corp.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN: 0-13-653775-8

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall of Canada, Inc., *Toronto*
Prentice-Hall Hispanoamericana, S. A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

PC ARCHITECTURE FROM ASSEMBLY LANGUAGE TO C

PREFACE

Over the past decade, there have been many books on PC assembly language written for computer science and engineering technology programs. Some texts place a heavy emphasis on microprocessor hardware, and others have a stronger focus on assembly language and operating systems. This book, while placing a slightly stronger focus on assembly language, seeks to provide a practical balance between hardware and software. Timing diagrams, 80x86 bus configurations, PC bus configurations, and a set of interfacing experiments make up the hardware components of the text. A complete coverage of assembly and how it relates to the PC hardware, the BIOS, and C make up the software component of the book. Numerous examples are given to supplement topics in the text.

This book can be used as a text or as a reference manual. As a text, it presents the history of the microcomputer. It starts with a crude microprocessor unit (the Easy4), evolves to .COM files (with structure derived from the CPM operating system), .EXE files (which allow memory segmentation), and ends with C files. As a reference, the appendixes contain valuable information on BIOS and DOS interrupts (Appendixes A and B), PC memory and port maps (Appendixes C and D), a summary of 8088-based instructions (Appendix E), an ASCII table (Appendix F), installing the Easy4 software (Appendix G), and a set of interfacing experiments (Appendix H). The Easy4 software and the library file ASY.LIB are included with the text. ASY.LIB is on Disk 2 and can be copied to the MASM directory.

The PC operating system (whether it is DOS, Windows® 3.1, or Windows 95) has a common core of BIOS and DOS interrupts. These interrupts are explained with numerous examples, along with an introduction to 80x86 assembly language and the PC memory map. This book emphasizes programming as a means to teach PC architecture. The text is designed for instructors desiring to give their students a solid understanding of how the PC operating system works. Students who have had an introductory course in the C language and want to broaden their programming knowledge by learning how C relates to assembly will find this book useful. The prerequisite for this book is, at minimum, a course in the C language. Although this is not absolutely necessary, it would also be helpful if the student had or is taking a course in digital systems.

The text begins at an elementary level and advances to coverage of interrupt service routines, terminate and stay resident programs, and DOS memory configurations. A simple processor called the Easy4 is used to help introduce basic architecture without relying on advanced 80x86 topics like segmentation. At Miami University, a separate course is used to teach such hardware topics as bus architecture, address decoding, DMA, and other related concepts. The authors envision this book being used in two- or four-year computer or electrical engineering technology programs. It may also be useful in computer technology or computer science programs.

Learning assembly language on 80x86 processors can be a daunting task for students. Not only do they have to learn the CPU registers and assembly language mnemonics, but they must also understand the segmented architecture of the 80x86 and the structure of .EXE files on the PC. To this end, the book gradually introduces these topics. Chapters 1 through 10 include topics normally covered in a course on PC assembly language, and can be covered in a single semester. Chapters 11 through 14 introduce advanced topics, and can be included in a second course or as selected extensions to a first course.

Chapter 1 introduces students to bits and bytes, base conversions, and basic microprocessor architecture. Chapter 2 covers a pseudocomputer, the Easy4. It allows students to "see" the relationship between registers and memory by graphically displaying them on the screen. A simple stack is also introduced in this chapter, as well as instruction fetch, decode, and execute cycles.

Segments are introduced in Chapter 3 and are fully explained in Chapter 5. Since .COM files are easier to understand, they are introduced before .EXE files. This gives the student time to learn the 80x86 assembly language mnemonics before studying segmentation and the .EXE structure. This is a departure from most assembly language texts, which tend to introduce .EXE files when they are covering assembly instructions.

Chapter 6 introduces the ASY.LIB library provided on disk with the book. This library contains many common I/O routines for the keyboard, screen, and printer. The 80x86 stack is also introduced in this chapter. Chapter 7 provides some useful PC examples using the ASY.LIB library, including the clock timer, toggle keys, keyboard buffer, and writing directly to screen memory and printer ports. Chapter 8 covers basic string operations.

Chapter 9 gives some basic floating point operations, including transcendental functions. This chapter gives examples that engineering technology students may find useful. Chapter 10 covers interrupts and interrupt service routines. Examples are given using the screen and mouse.

Chapter 11 provides unique coverage of an assembly text. Using Microsoft C as an example, it disassembles basic C programs to show the equivalent 80x86 assembly language version of the .EXE file. This helps students understand how C is converted into assembly when compiled. There is also coverage of intermingling assembly and C code, including passing variables from C to assembly and back. Terminate and stay resident programs are also covered in this chapter, using combined C and assembly routines.

Chapter 12 covers the PC boot process and the memory map. Programs used to view the memory map are also covered. Chapter 13 covers advanced C memory and I/O routines, mainly oriented toward tips and techniques not covered in a standard introductory course in C. Chapter 14 covers mnemonics from the 80286, 80386, and 80486. Finally,

Appendix H contains a set of interfacing experiments including I/O ports, analog measurements that include maximum, minimum, DC, and RMS voltage readings, and frequency measurements. Also included is an experiment on creating an ISR for interrupt driven I/O.

Many hours of work have gone into the writing of this text. Thanks to all of the students at Miami University who have patiently tolerated the many revisions. We would also like to thank the following reviewers for their invaluable feedback: Mike Awwad, DeVry Institute; Boris Kovalchuk, Central Washington University; Michael A. Miller, DeVry Institute; and Gregory S. Romine, Indiana University-Purdue University at Indianapolis.

David Hergert
Nancy Thibeault

BRIEF CONTENTS

CONTENTS

CHAPTER 1

Microprocessor Architecture

THE STRUCTURE OF A COMPUTER

In twenty years' time, computers have shrunk from large mainframes that took up a whole room, to laptops that weigh less than six pounds. At the same time, microcomputers have advanced from a small box with no keyboard or monitor to multimedia machines capable of editing motion pictures. This book is about PC-based (or IBM®-compatible) microcomputer architecture. In a microcomputer, the electronic hardware that makes up the computer is contained in a variety of integrated circuits including the microprocessor, clock, memory, and port chips (Figure 1.1).

The **microprocessor** chip includes all of the circuitry needed to implement instructions coming from the operating system or programming language. In mainframe computers, the processor is contained in many separate circuits. In microcomputers, the processor has been squeezed onto one chip. Program instructions and data are stored in **memory** chips. Today's microcomputers have far more memory than a mainframe of thirty years ago had. Instructions are stored in one section of memory, and data in another. In order to interpret instructions stored in memory, a microprocessor must first fetch each instruction from memory before interpreting it.

Port chips control access to outside devices such as the keyboard, monitor, mouse, and printer. Think of a computer port chip as being similar to a shipping port. When a boat laden with goods comes from overseas, the boat must first stop at an assigned port before the goods can enter the country. The same is true for a computer port. When data comes from an outside source, it must be routed to a port in order for the microprocessor to have access to it. The microprocessor has instructions that allow it to send information to or receive information from a port.

All computers have a basic operational structure as shown in Figure 1.2. The **Operating System** and high-level ***Programming Language*** control the ***Electronic Hardware*** that comprises the physical layer (Layer 1) of the computer. The programming language could be BASIC, Pascal, C, FORTRAN, or any high-level language. When the computer is turned on, a core routine loads the operating system in first (this is called a system boot),

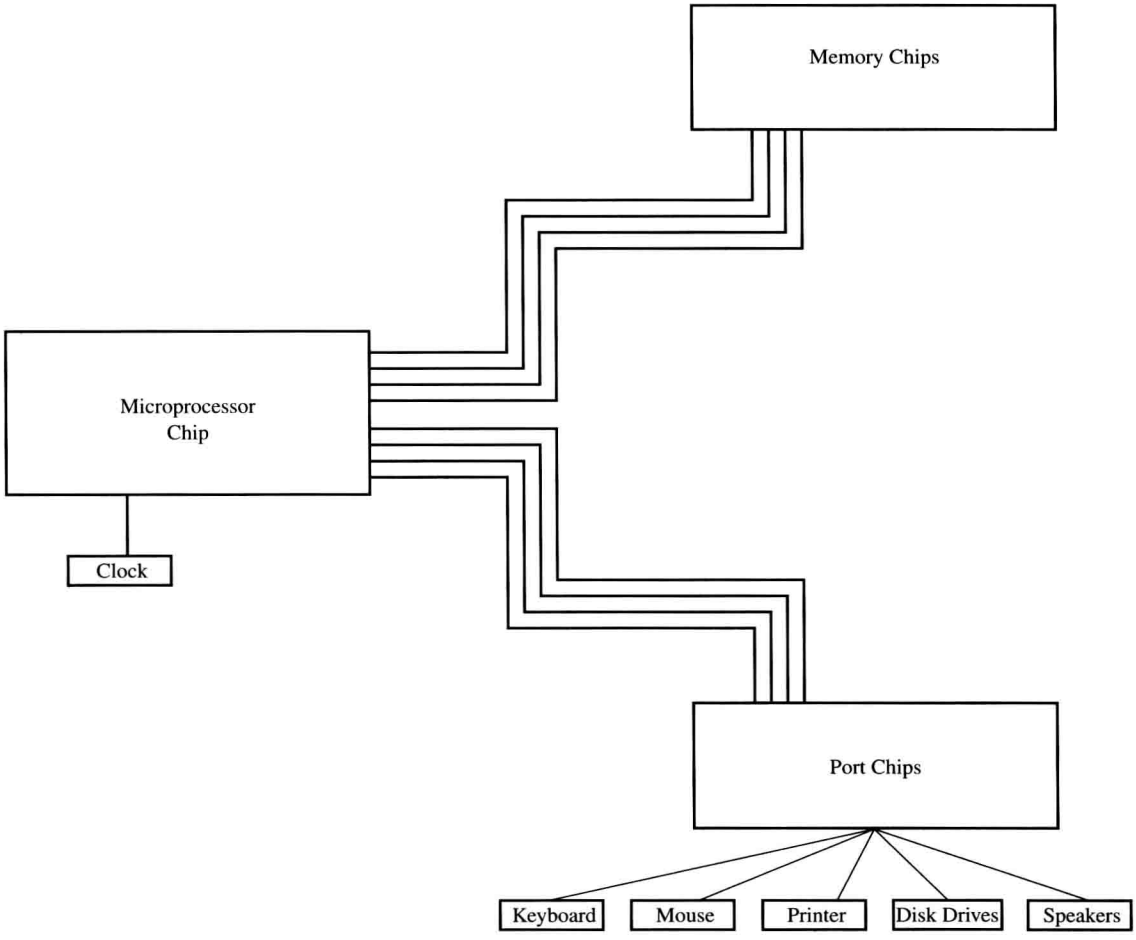

FIGURE 1.1 Microcomputer System

and the programming language is then loaded by the operating system. At higher layers, the computer is easier to use and program.

The ***Application Program,*** Layer 4, is a program that is frequently used, such as a spreadsheet, CAD program, or word processor. It is written in a programming language *(Layer 3).* Most modern software is written in either the C or C++ language or a combination of Assembly and C. The operating system is also written in a combination of Assembly and C. The programming language communicates with the operating system *(Layer 2)* to control the electronic hardware *(Layer 1).* Note the arrows connecting the four layers in Figure 1.2. All four are intertwined. This book slowly unravels the connection between them. You will study the underlying glue that holds them together. This "glue" is called machine language. Applications programs, programming languages, and the operating system must all be converted to a series of instructions that the microprocessor