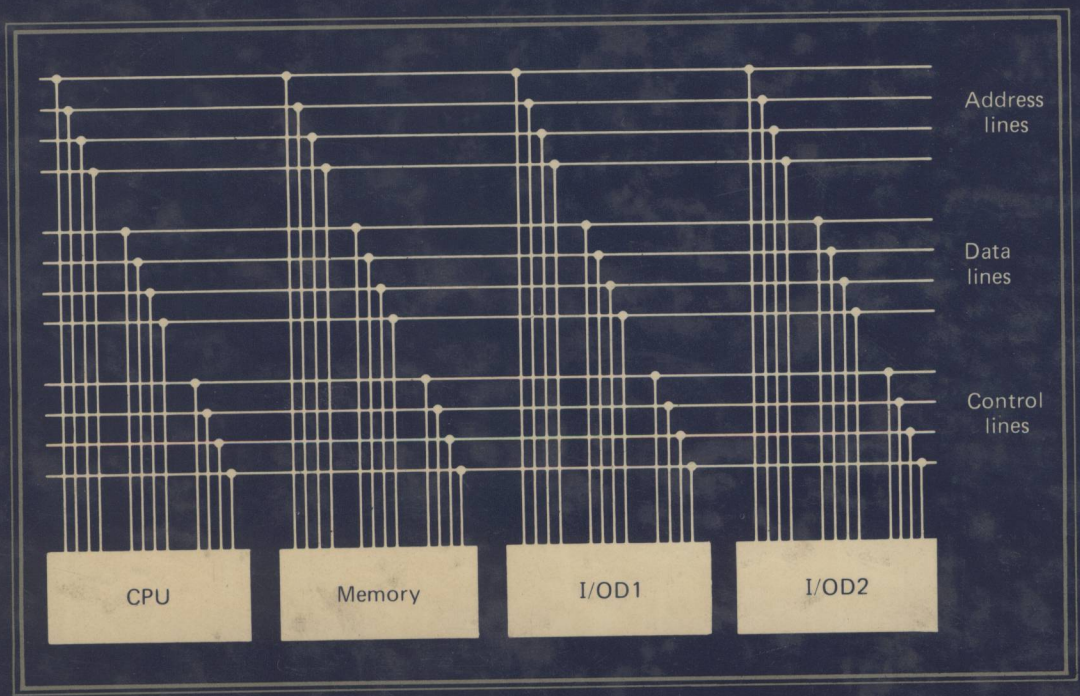
68000 ASSEMBLY LANGUAGE PROGRAMMING

A STRUCTURED APPROACH

J. Michael Bennett



68000 ASSEMBLY LANGUAGE PROGRAMMING
A Structured Approach

J. Michael Bennett

PRENTICE-HALL, INC.
Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging-in-Publication Data

BENNETT, J. MICHAEL.
68000 assembly language programming.

Bibliography: p.
Includes index.
1. Motorola 68000 (Microprocessor)—Programming.
2. Assembler language (Computer program language)
3. Structured programming. I. Title. II. Title:
Sixty-eight thousand assembly language programming.
QA76.8.M67B46 1987 005.265 86-9393
ISBN 0-13-811381-5

Editorial/production supervision and
interior design: *Carol L. Atkins*
Cover design: *20/20 Services, Inc.*
Manufacturing buyer: *Ed O'Dougherty*

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-811381-5 025

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

Preface

The purpose of this book is to make you intimately familiar with the programming environment of the Motorola MC68000 family of microprocessors. Here we cover the architectural details of the basic MC68000, the MC68008, the MC68010, the MC68012, and the MC68020. A very large part of this book is concerned with teaching you how to become an effective Assembly Language programmer. We have chosen the MC68000 as our typical example of a modern 16/32-bit microprocessor. However, the ideas advanced can be easily transferred to other popular machines, such as the Intel 80286, the National Semiconductor 32016, and the DEC MicroVax. We could just list a set of MC68000 operation codes of course. But that provides no understanding of the *how* and the *why* of the opcodes. Therefore, we constantly try to illuminate the philosophy of why particular design choices were made. We illustrate the important architectural features of the machine. Even if you never return to Assembler after reading this text, you will always retain a feeling for the structure of the MC68000 series. You will be able to constructively compare it against other micros.

Most books in this area tend to be written from the viewpoint of electrical engineering. This text approaches the topic from the software side and describes hardware in software engineering terms. It is *not* a hardware text but will still give the ordinary programmer an opportunity to learn about the hardware side of things, especially in how it can affect the operation of programs that he or she might write.

The pedagogic approach we apply is to use Pascal as our starting point. We assume that you have an understanding of the language (C or FORTRAN-77 would do as well). Using the constructs of that language and results from recent advances in research into high-level programming language techniques, we teach Assembly

Language from a structured and disciplined point of view. One of the major differences between programming in Assembler compared with programming in Pascal, for example, is that, since you have full access to the machine's resources, you have the power to inadvertently do great damage to your programs, too. Therefore, we stress the use of modern programming techniques such as structuring and data classing to underline the importance of discipline in the writing of good Assembler programs. Following our techniques will go a long way in helping you write clear, readable, modifiable, yet efficient MC68000 Assembly Language programs. To this end, we introduce a new structuring concept called PASWEB. Based on the work of Knuth [3], it greatly improves the readability of assembler programs.

We also examine interfacing techniques from the programmer's point of view. We examine how I/O devices, from simple switches to hard disks, are interfaced into the MC68000 and the ramifications to the software. Implicit in this is a detailed examination of the problem of bus structures connecting CPUs to memory and I/O devices. We also examine exactly *when* you should use Assembler and when you should remain with a high-level language. Finally, we discuss various hardware assists that have been provided to aid in efficient implementations of operating systems and related systems support software.

Our journey is long and arduous. But we will illustrate points along the way with meaningful, real examples of actual MC68000 code. Throughout the text, we try to illustrate design trade-offs made by the Motorola designers. In this manner, the text also provides useful insights into the realm of computer architecture.

The reader is assumed to have had a general introduction (formal or practical) into computer science and to be familiar with at least one high-level language, preferably Pascal. Second, a previous introduction into some Assembly Language programming would be helpful but not mandatory. The main thing you need is an inquiring spirit and the ability not to get discouraged if you do not understand things immediately. One prerequisite that is *not* expected is any knowledge of electrical engineering.

The book is structured with the view of getting you writing code as soon as possible, yet consistent with understanding the architectural details. Chapter 2 describes the programming model, the various addressing forms, and when you should use each and why. Chapter 3 describes individual components of the instruction set and gives examples of their use. Chapter 4, using Pascal as the model, shows how to convert structured control statements into equivalent MC68000 code. Data classing is also covered. Chapter 5 is concerned with simple I/O. If you are interested in getting a simple program running in the minimum amount of reading and/or teaching time, follow the sequence 2.1, 2.2, 2.3, 2.4.1, 2.4.2, 3.1, 3.2, 3.3.1, 3.3.2, 4.2.1, 4.2.2, and 5.3. That is enough to handle a subset that will allow you to write a simple program. You may return later to handle more complex forms.

Chapter 6 stresses the need to approach Assembly Language programming with a plan, and we give one. Chapter 7 covers the real story on I/O, starting at the very lowest level and building up, in an organized fashion, to programming UARTs and PI/T chips (serial and parallel interface chips), concluding with a look

at interrupts, traps, and DMA operations. Chapter 8 examines the Pascal/Assembly Interface, addresses interfacing the two, and discusses when to use what. The book concludes with an examination of the whole family, including the CPUs available, the MC68020, and other support chips.

Why should you read this book? The most important benefit for doing so is that you will gain a clear and comprehensive understanding of the development of one family of computers. You will learn a very popular 16-bit computer's Assembly Language in an organized and structured manner. This experience can be easily extended to any other assembler on currently available machines. You will learn when Assembly Language can be used for great effectiveness in conjunction with high-level languages such as Pascal. This knowledge will provide you with insight into the architecture of the MC68000.

You will also learn how I/O devices are connected to micros, why bus structures are important, and the effects of these features on the design of the I/O service routines [I/O serrs][1] that do the software interfacing. You will also learn about the writing of I/O serrs, how to handle interrupts, and where and what to do with them. Finally, you will gain familiarity with the jargon and buzz words of the field, enabling you to converse intelligently with hardware interface people. This familiarity with the MC68000's architectural details and the knowledge of the language of the area will also make it easier for you to read and understand Motorola's (and other microprocessor manufacturers') literature and technical articles.

Last, but certainly not least, the MC68000 family provides several hardware assists that aid operating system designers in implementing better software. Reading this text will show you how current advances in computer architecture are shrinking the "semantic gap"; that gap between the hardware constructs provided by the machine and the software constructs needed by modern, high-level software. The MC68000 family is one of the most advanced series of micros available. Thus, it is fitting to contrast the various members of the family and show where each member, the MC68008, the MC68010, the MC68012, and the MC68020, would be used.

Finally, we constantly stress the design aspect of the MC68000's architecture, stressing why certain choices were and were not made. This will enable you to fairly accurately predict what advances are likely to be forthcoming in the next few years. The knowledge thus gained is useful with Motorola or other popular manufacturers.

I would like to acknowledge the special assistance given by Motorola, Inc. in the putting together of this book. The material in Appendixes A–G has been reprinted by permission of Motorola, Inc.

Most importantly—to Kirsten: TAK for ALT!

J. Michael Bennett

[1]Of necessity, this book is full of acronyms and short forms. We flag any such defined terms with their acronyms in square brackets [SB] like this. Please try to note them as you read.

Contents

CHAPTER 1

The World of the Microprocessor

In this chapter, our purpose is to briefly cover the fascinating history of the development of microprocessors and to locate the MC68000 family within that development. We will also discuss some of the differences (and difficulties) in programming on a micro as opposed to a large mainframe. Finally, we look at the scenario in which you may find yourself in trying to learn MC68000 Assembler.

1.1 THE MC68000'S PLACE IN THE SCHEME OF THINGS

The historical development of the tool that we now call the computer stretches back over a very long time. Even the fundamental conceptual ideas of how a computer should function were worked out in the beginning of the last century by the Englishman, Charles Babbage. However, the realization of those ideas required a suitable physical medium for their implementation. That technology only became available during the chaotic, yet immensely fruitful, explosion of intellectual energy unleashed by World War II. The first real digital computer, as we know it, was developed by an Austrian, Konrad Zuse, in the early 1940s. Unfortunately, not much was known of his work until recently (Zuse unwittingly also presented the world with the first example of computer obsolescence; the Z-4 was bombed into that state during an RAF air raid on Munich in 1944), and we generally credit the Americans (Eckert and Mauchly) and the English (Wilkes and others) with developing the first computers. At any rate, both UNIVAC and IBM had commercial models available in the early 1950s.

Architecturally, these computers were classified as von Neumann machines.

The remarkable thing is that virtually all our computers to date can be so characterized. If biologists were to categorize computers like primates, all our machines would fit into a subphylum similar to "homo sapiens intelligensi." To be sure, speeds differ, capacities span orders of magnitude (even the colors are different), but, topologically, the world's largest computer is structurally similar to the tiny micro driving your calculator.

What *has* changed drastically since von Neumann's time is the technological bedrock in which the various von Neumann architectures have been implemented. And what a change! Gone are the leviathans of the 1950s, those million-dollar computer dinosaurs that consumed vast amounts of energy and real estate, failed with distressing regularity, yet performed at the capacity of a pocket calculator. It is not surprising that even John von Neumann thought that a handful of these digital monsters would be enough to service the civilized world until the millenium. We clearly could not afford many.

In parallel with the development of vacuum tube computers, physicists were developing solid-state equivalents to the large, power-hungry, and unreliable tubes. For our computers depend on Boolean algebraic logic, and the vacuum tube is but *one* implementation of that. We could equally well use any device that has a binary characteristic; a transistor, a water level, or even an Irish leprechaun waving an orange flag (though they tell me their reliability is open to question). Thus, once the solid-state technology became mature and cost effective, computers were reimplemented using those smaller, more reliable devices.

The consequence of all of this was that computers became easier to make and thus cheaper. No longer were they the sole preserve of research institutions, military complexes, or large companies. Now they could be owned by much more modest social organizations. This proliferation also had an important side effect. The democratization of the computer led to a very serious software problem. Thus, this period also saw the development of high-level computer languages, operating systems, and similar systems software.

The next generation was spawned by more complicated solid-state developments. Physicists succeeded in putting together several transistors into the same space formerly occupied by one. Such small-scale integration [SSI] accelerated this already established trend toward miniaturization and reliability. As the construction of computers became manifestly easier, the designers had an option of two directions into which they could evolve. One was toward more baroque architectures providing physically smaller, but more powerful machines for the same dollar. The other was to provide far simpler machines for far less money.

Almost all the established computer companies chose the former route. One chose the latter, and the decision of Digital Equipment Corporation [DEC] to introduce the PDP-8 in 1965 dates the beginning of the minicomputer revolution. The details of that fascinating technological development need not detain us here. But the salient points to be noted are that the developments in solid-state technology made possible a reincarnation of the first-generation machines in a far cheaper medium. For a PDP-8 is the architectural kid brother of the UNIVAC-1. Like those

first tube relics, the PDP-8 (initially) had about as much software. Yet they were seized by the thousands by scientists on a budget and placed to work controlling complex equipment, gathering and reducing data and eventually front-ending onto mainframes. Nor does the story end here. For minicomputers went down the same evolutionary road as their bigger mainframe siblings. Now we identify three distinct minicomputer generations, both in a software and hardware sense. Indeed, a mature minicomputer product line (such as the PDP-11) can now boast a very impressive range of software and hardware. So successful was this development that the world, in a short span of seven years, had more minicomputers than mainframes.

Our story now has as many parallel threads as a good novel. For while mainframes were struggling to fend off predatory attacks by marauding minis, developments in solid-state technology marched on at an accelerated pace. SSI gave way to medium-scale integration [MSI], and we were now seeing a density of 100 gates per chip. By the end of the 1960s, chip manufacturers were producing products that had densities approaching a thousand.

At the turn of the decade, a now-defunct Japanese calculator manufacturer approached a fledgling Silicon Valley chip manufacturer about the possibility of producing an unusual set of calculator chips. The only problem was that their design required a chip density in excess of 2000. The company responded and two years later began marketing a 4-bit microprocessor. The production of the Intel 4004 microprocessor signalled the beginning of the microcomputer revolution.

Again, plus ca change, plus c'est la meme chose.[1] For the Intel 4004 architecture was reminiscent of the UNIVAC I or the PDP-8. The era of LSI, large-scale integration, had arrived, and the 4004 was quickly followed up with an 8-bit version, the Intel 8008. While the 4004 was really only designed to drive a calculator, the 8008 was a real computer processor. However, like UNIVAC and DEC before them, Intel ignored the holistic approach to complete computer systems. Their products were software-naked and, worse, required extensive interfacing (or electronic "glue") to attach them to outside-world devices. In 1974, Motorola announced the MC6800; a solid 8-bit design with a family of compatible I/O chips, usable software, and a microcomputer development system. So successful was this approach that it was duplicated shortly by Intel, Zilog, and many others. Again, the cycle repeated. Within 3 years of the introduction of the first real micro, the volume of microprocessors exceeded the combined total of minicomputers and mainframes. Software also became more commonly available, and micros even joined in the generation game.

Useful as the 8-bit micros were and are, they have severe arithmetic and addressing limitations. Thus, as the three streams of computer technologies advanced, so did basic semiconductor research. At the turn of the 1980s, densities were approaching 100,000 transistor-equivalents per chip and the first of the true 16-bit micros were delivered. The first four available were the Texas Instruments TI9990, the MC68000, the Zilog Z-8000, and the Intel 8086/88 (of IBM PC fame).

[1]The more things change, the more similar they become.