Assembly Language Programming and Machine Organization

PDP-11

Ed Wishart

ASSEMBLY LANGUAGE PROGRAMMING AND MACHINE ORGANIZATION

PDP-11

Ed Wishart
University of Nevada—Reno

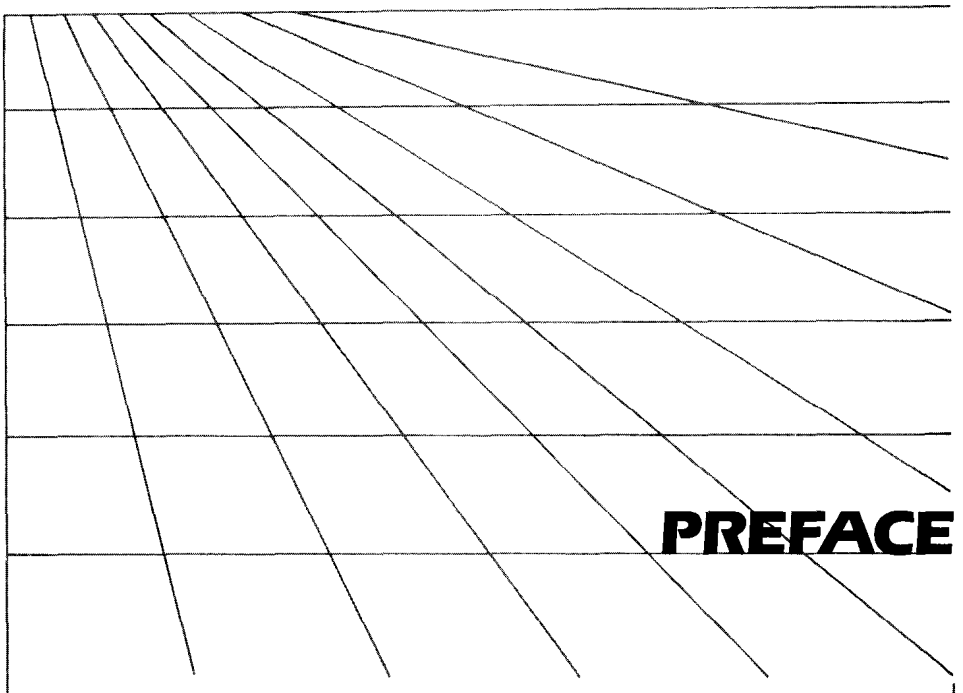
Allyn and Bacon, Inc.
Boston London Sydney Toronto

Series Editor: John Sulzycki
Production Coordinator: Sue Freese
Editorial/Production Services: Linda Zuk, WordCrafters, Inc.
Cover Coordinator: Linda K. Dickinson
Cover Designer: Susan Hamant

Library of Congress Cataloging-in-Publication Data
Wishart, Ed.
Assembly language programming and machine organization.
Includes index
1. PDP-11 (Computer)—Programming. 2. Assembler language (Computer program language) I. Title.
QA76.8.P2W57 1987 005.2'45 86-20553
ISBN 0-205-10477-0

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1 90 89 88 87 86



PREFACE

This book is an introduction to assembly language programming and computer architecture done in the context of two computer architectures: SIMPLE and Digital's PDP-11. Memory organization, registers, instruction sets, addressing modes, I/O systems, interrupt structures, and bus communication are some of the major topics covered. In addition to these hardware concepts, software concepts such as subroutine linking, parameter passing, conditional branching, macros, floating point arithmetic, recursion, and modular program design are important topics in this book. In short, this is a text for CS-3, the third course of ACM's Curriculum '78. CS-1 and CS-2 are courses in programming and algorithms in the context of high level languages and should be considered prerequisites for a course based on this text.

Program execution, input, output, and termination require the assistance of the host operating system. The method by which this assistance is obtained is illustrated for two operating systems used on PDP-11 computers: UNIX and RT-11 (and the RT-11 emulation within RSTS/E). This assistance is requested by means of *system calls*, which function similarly in every operating system, but differ in syntax. Therefore, this text can be used profitably with any operating system running on the PDP-11 that supports MACRO-11; one need only consult software manuals or local wisdom about the form of the system calls once it is known what assistance is needed.

PLAN FOR THE TEXT

Chapters 1 through 8 make up the core of the book, and it should be possible to cover them in depth in a one-semester course and still have time to survey the material in the rest of the text. Chapters 9 through 11 cover more advanced material such as floating point arithmetic, recursion, bus communication, virtual computers, and an overview of CISC versus RISC. Some of these topics are not normally part of an introductory assembly language course—they are included for the sake of the curious and to provide a reference for future studies. Chapter 12 is meant to prepare students for the term programs of Chapter 13 by leading them through the step-by-step design and coding of a 500-line MACRO-11 program—a SIMPLE emulator for Chapter 1.

Chapter 1 introduces many machine and assembly language concepts by means of a simulated decimal computer that we call SIMPLE. Its 1950s architecture and its ability to function with no supporting software (editors, assemblers) enables students to begin programming almost immediately. Of course, to get the full benefit of this chapter, students should execute their programs on a SIMPLE machine. A SIMPLE emulator written in MACRO-11 is developed in Chapter 12 and listed in Appendix G. Other emulators written in C and Pascal are available from the author. Advanced students or those wishing to learn MACRO-11 sooner may skip Chapter 1 and refer to it only as needed. SIMPLE is a source of an occasional exercise throughout the book, the most notable being a cross assembler outlined in Chapter 13.

Chapter 2 covers data representation, with particular attention paid to changing the representation of integers from one base to another. Chapter 3 introduces the MACRO-11 assembly language in the context of writing and reading character data to and from the user's terminal. This is done almost entirely with high-level language concepts such as assembler directives and programmed requests or system calls. Chapter 4 introduces the PDP-11 architecture by explaining 16 instructions and four addressing modes, as well as the PDP-11's register and memory organization. Chapter 5 explains the system stack and its use in subroutine linking, as well as seven more instructions. Chapter 6 completes the explanation of the basic PDP-11 architecture by covering the four remaining addressing modes, the details of conditional branching, the logical instructions, and the shift/rotate instructions. In addition, Chapter 6 illustrates subroutine communication utilizing addressing modes 3, 5, 6, and 7. Chapter 7 explains macro definition, reference, and expansion with several examples of macros to permit programming at higher levels of abstraction. Chapter 8 explains traps and interrupts, and contains examples of code to decode traps and respond to I/O interrupts.

Chapter 9 explains floating-point number formats and the floating-point instruction set, as well as recursion in the assembly language setting. Chapter 10 explains how communication takes place on two computer buses: a hypothetical MINIBUS and Digital's UNIBUS. Normally this topic is not part of CS-3, but it is covered because the UNIBUS and Q-bus have played such a major role in the evolution of the PDP-11, and the cost in terms of bus time for instruction and operand fetch is such a large part of instruction execution time. In addition, memory mapping and context switching is covered. This chapter contains a fair amount of detailed material that the author surveys in one or two lectures. Chapter 10 provides a good lead into Chapter 11, since it exposes the subterfuge of operations carried out in memory such as "ADD A, B". Chapter 11 looks at zero- and three-address instruction formats, and virtual computers built with microcode, and compares complex instruction set computers (CISC) with reduced instruction set computers (RISC).

Chapter 12 and 13 go together. Chapter 12 explains a program design methodology directed towards making the term programs of Chapter 13 more enjoyable by reducing frustration and increasing the probability of success. This incremental design methodology is illustrated by building a SIMPLE emulator in MACRO-11. This chapter also illustrates several programming techniques not covered elsewhere in the text: file access, jump tables, and the catching of keyboard interrupts. Chapter 13 contains detailed outlines for five term programs that require 300 to 600 lines of code to complete. Students at this stage in their computer science studies need experience in managing software projects—these programs are meant to give them this experience. The author allows two students to work together on a program; this usually improves the result and makes the project more fun, as well as providing experience with team programming.

The 95 pages of appendices are meant to make the text as self-contained as possible by listing all instructions, programmed requests/system calls, directives, and error codes in systematic fashion for ease of reference. A Glossary and Index provide additional aid to the students.

In addition to being a book about bits and bytes and the role they play in computing, this text emphasizes *structured programming*. Like all assembly languages, MACRO-11's primary control structure is the conditional branch (GOTO). However, by using the abstractions provided by subroutines, macros, and defined constants, one can write modular programs. If these modules are well commented and have narrow and well-defined interfaces with each other and cause no side effects, then the resulting program should be understandable and hence maintainable. This captures the essence of structured programming much better than the simple avoidance of GOTOs.

A word about the exercises. The best way to deepen and verify understanding of the concepts presented is to do the exercises. Exercises that involve the writing of code are either a complete program or a subroutine, and thus can be tested by execution. There are no "dry labs." For many of us, mastery of computer science, like writing and mathematics, requires active participation—it is not a spectator sport.

ACKNOWLEDGMENTS

I owe thanks to many for help with this book; without them it would not have been completed. First, there are the fond memories of a first class of 10 students and a little LSI-11 that introduced us to the beauty of the PDP-11 architecture many years ago. Second, there are the hundreds of students who responded to my enthusiasm about assembly language and endured the early stages of the manuscript.

Third, heartfelt thanks are due to Ted Sarbin, head of Bally Systems in Reno, but formerly of Digital Equipment Corporation, for sharing his immense and intimate knowledge of PDP-11 hardware and software with me. In several places, his criticism made for a much clearer presentation.

Fourth, thanks are due the staff at Allyn and Bacon: John Sulzycki for his enthusiastic support, Sue Freese for giving the book a coherent and crisp appearance, Linda Zuk for her expert editing that made it read so much better. Thanks also to Mary Jeffreson and Richard Stewart for constructing the index. Fifth, thanks to the reviewers for their thoughtful and informative input: Steven Stepanek and Fred Gruenberger of California State University at Northridge; Robert Trenary of Western Michigan University; Joseph G. Tront of Virginia Polytechnic Institute and State University; Maarten Van Sway of Kansas State University; and Walter Piotrowski of SUNY-Binghampton.

Finally, I thank my family for bearing with me during the seemingly endless hours and late nights that this project took to complete. Thanks to Kathy, for being a good sport about the late nights, to Eric for his enthusiasm for life, and to Michael for following his Dad into the exciting world of computer science; to him this book is dedicated.

E. W.

CONTENTS

PREFACE xi

1. COMPUTER ORGANIZATION VIA SIMPLE 1

A simulated four-digit decimal computer gives a gentle introduction to machine organization and machine language programming and allows us to start programming immediately.

1.1. Fundamental Computer Concepts, 1

1.2. SIMPLE, 7

1.2.1. Memory, 9

1.2.2. Input/Output, 10

1.2.3. Central Processing Unit, 11

1.2.4. Instruction Set, 13

1.3. Assembly Language, 17

1.4. The Console, 30

Summary, 34

Exercises, 35

2. DATA REPRESENTATION – All About Bits and Bytes 39

An understanding of the representation of data and the differences between the internal and external representation is necessary to exploit the power of assembly language.

2.1. Positional Notation for Positive Integers, 39

2.2. Changing the Representation of Numbers from One Base to Another, 43

2.2.1. Arithmetic in the Notation of the Target Base, 44

2.2.2. Arithmetic in the Notation of the Original Base, 47

2.2.3. Arithmetic in the Notation of a Third Base, 50

2.2.4. Conversion Among Binary, Octal, and Hexadecimal, 51

2.3. Negative Integers, 53

2.3.1. Ones' Complement Notation, 53

2.3.2. Two's Complement Notation, 56

2.4. Coding, 58

2.4.1. ASCII, 59

2.4.2. Parity, 60

2.4.3. Examples of Data Representations, 61

2.5. BCD, 63

2.6. Data Communication, 64

Summary, 67

Exercises, 67

3. GETTING STARTED IN MACRO-11 – Communicating with Your Terminal 73

We take our first steps in MACRO-11 by writing and reading character data to and from our terminal.

3.1. MACRO-11, 74

3.1.1. Statement Format, 74

3.1.2. Types of MACRO-11 Statements, 75

3.1.3. Labels, 77

3.2. A First Program: HELLO WORLD! 79

3.2.1. Creating Text in MACRO-11 Programs, 80
3.2.2. Sending Text to the Terminal, 81
3.3. Documenting Your Programs, 86
3.4. A Second Program: ECHO, 87
3.4.1. Reading with RT-11, 88
3.4.2. Reading with UNIX, 89
3.5. Assembly and Execution of MACRO-11 Programs, 90
3.5.1. Assembly and Execution under RT-11, 92
3.5.2. Assembly and Execution under UNIX, 93
3.5.3. Error Messages During Assembly, 94
Summary, 94
Exercises, 95

4. INTRODUCING THE PDP-11 – CPU, Memory, 16 Instructions, Four Addressing Modes 97

The complexity of the PDP-11 is uncovered a little at a time. Knowledge of one-fourth of the instructions and half of the addressing modes let us start writing useful programs.

4.1. Memory, 99
4.2. The Central Processor, 102
4.3. Sixteen Instructions, 105
4.4. Addresses, Values, and Expressions in MACRO-11, 110
4.5. Addressing Modes, 114
4.6. Machine Language Representation of Some Instructions, 117
Summary, 123
Exercises, 124

5. STACKS, SUBROUTINES, AND SEVEN MORE INSTRUCTIONS 129

Subroutines are the most important programming tools available to the assembly language programmer. They provide the abstraction and modularity that are necessary to design, write, and maintain large programs.

5.1. Stacks, 130

5.2. Subroutines, 132
5.2.1. Getting There and Getting Back, 133
5.2.2. Linking the Modules Together, 136
5.2.3. Parameter Passing, 140
5.2.4. Saving Registers, 143
5.3. How Big is Small? 147
5.4. Seven More Instructions, 150
Summary, 155
Exercises, 156

6. THE PDP-11 CONTINUED – More Addressing Modes, Branching, Logical Instructions 161

Except for a few miscellaneous instructions, this chapter completes the description of the PDP-11 as viewed by the applications programmer. The remaining instructions belong in the domain of the systems programmer.

6.1. Double Indirect and Indexed Addressing, 161
6.2. The Condition Codes: Carry and Overflow, 178
6.3. Signed and Unsigned Conditional Branches, 182
6.4. Instructions That Operate on Bits, 187
6.4.1. Logical Instructions, 187
6.4.2. Shift and Rotate Instructions, 190
Summary, 192
Exercises, 193

7. MACROS 199

Macros provide a second method for abstracting details of program design and managing complexity. Since they are "called" at assembly time, parameters are passed lexically rather than by value or location; this method has surprising power.

7.1. Parameters, 202
7.2. Labels within Macros, 207
7.3. Conditional Assembly, 209
7.4. Repetition, 213

7.5. Placement of Macro Definitions, 215

7.6. Listing the Macro Expansion, 217

Summary, 218

Exercises, 220

8. TRAPS AND INTERRUPTS 223

Traps and interrupts provide a method of transfer of control with a new dimension: changing the contents of the processor status register.

8.1. Traps, 226

8.2. Interrupts, 231

8.3. Role of the PS in Traps and Interrupts, 238

Summary, 239

Exercises, 240

9. ADVANCED TOPICS – Floating Point Arithmetic and Recursion 241

Floating point arithmetic lets the PDP-11 do scientific computations. Recursion gives us a new method of problem solving; implementing it in assembly language strips away the mystery of recursion in high level languages.

9.1. Floating Point, 241

9.1.1. Representation of Floating Point Numbers, 242

9.1.2. Floating Point Operations, 246

9.2. Recursion, 253

9.2.1. Examples of Recursive Definitions, 255

9.2.2. Implementing Recursion, 260

9.2.3. Recursion in MACRO-11, 262

Summary, 269

Exercises, 270

10. THE UNIBUS AND MEMORY MANAGEMENT 273

The UNIBUS is the glue that ties the components of a PDP-11 computer system together by letting them communicate with each other. The memory

management unit is the hardware device that separates virtual memory from a larger physical memory and enables the computer to support a multiuser environment.

10.1. Buses, 274

10.2. Communication Via the Minibus, 277

10.3. Communication Via the UNIBUS, 282

10.3.1. A Bus Read Cycle, CPU as Bus Master, 284

10.3.2. Nonprocessor Request (DMA) Bus Cycle, 286

10.3.3. Bus Interrupt Cycle, 288

10.4. Memory Management, 291

Summary, 300

Exercises, 301

11. POTPOURRI 303

Chapter 11 takes a brief look at other computer organizations with regard to addressing, the implementation of virtual computers via microcode, and complex instruction set computers versus reduced instruction set computers.

11.1. Addressing Methods, 303

11.2. Virtual Computers and Microprograms, 305

11.3. CISC versus RISC, 307

12. INCREMENTAL PROGRAM DESIGN VIA AN EXAMPLE – A SIMPLE Emulator 311

To prepare you to handle the term programs in Chapter 13, we illustrate a design methodology that allows the design, coding, and testing to proceed in unison. By this means we can keep the amount of code that is being designed, written, and debugged to about a page, for example about the size of ADTOB from the exercises of Chapter 6.

12.1. Incremental Program Design, 311

12.2. An Example of Incremental Design: A SIMPLE Emulator, 313

12.2.1. The Display Command, 318

12.2.2. The Load Command, 324

12.2.3. The Run Module, 326

12.2.4. The Peek/Poke Command, 332

12.2.5. Reading a SIMPLE Program from a File, 334
12.2.6. Making it User-Friendly, 339

13. TERM PROGRAMS 349

The suggestions for term programs in this chapter let you experience the deep sense of creativity that programming affords, as well as sharpening your programming skills.

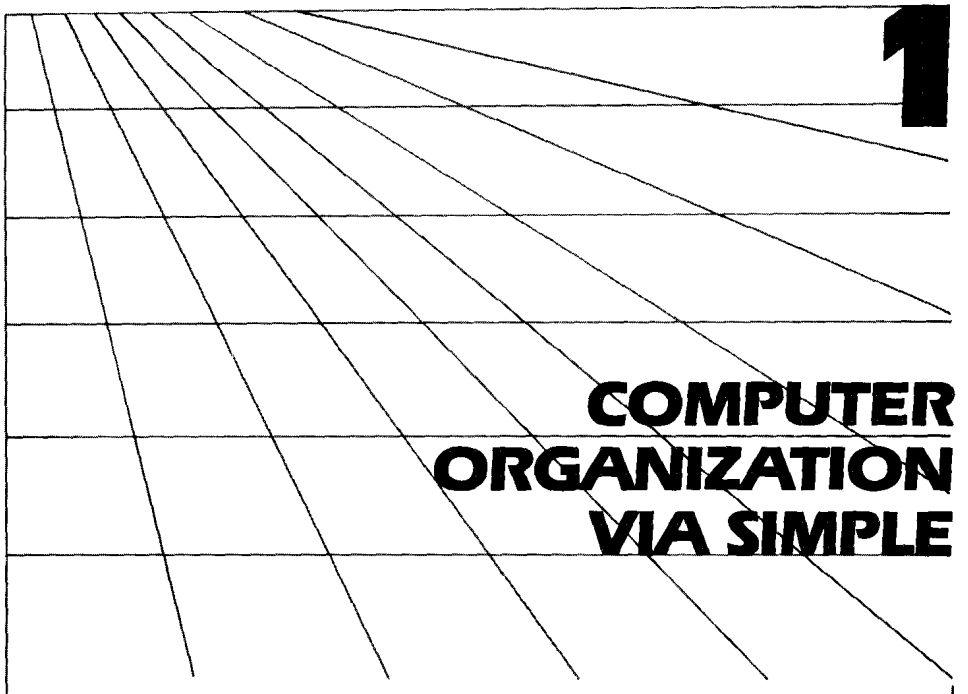
13.1. SIMPLE Assembler Program, SAP, 349
13.2. Four-Dollar Calculator, FDC, 354
13.3. Double-Precision Integer Arithmetic, 359
13.4. Clock, 362
13.5. Tadpole, 368
13.6. Term Program Documentation, 374

APPENDICES

A – The ASCII Code, 377
B – PDP-11 Instructions, 379
C – Selected Programmed Requests and System Calls, 403
D – Selected MACRO-11 Assembler Directives, 423
E – MACRO-11 Error Codes and their Meanings, 433
F – Dynamic Debugging: ODT and ADB, 437
G – SIMPLE Emulator, 449

GLOSSARY, 463

INDEX, 471



1 COMPUTER ORGANIZATION VIA SIMPLE

1.1. FUNDAMENTAL COMPUTER CONCEPTS

We begin our study of computer organization and assembly language programming by introducing basic machine and assembly language concepts. We do this with a simulated decimal computer that, for want of a better name, we call SIMPLE. SIMPLE allows us to avoid the complexities of present-day computers, provides a friendly environment in which to work, and enables us to introduce many fundamental hardware and software concepts much earlier than would otherwise be possible. In this chapter we introduce and define these terms:

accumulator	effective address	mnemonic code
address	indexing	program counter
assembly language	input/output	register
memory cell	loop	self modifying code
central processing unit	machine language	stored program concept
conditional branching	memory	word

Before describing SIMPLE, we introduce some general concepts about computer organization at the machine language level with examples from everyday life. All computers have four fundamental components:

- ☐ Central Processing Unit (CPU)
- ☐ Memory
- ☐ Input Device
- ☐ Output Device

These components can even be identified in an inexpensive hand calculator with memory: The calculator's display is the output device; the keys are the input device; the memory is accessed with the *STOR, M+, M-,* keys; and the CPU is the device that performs the arithmetic computations. However, such a calculator is not a *computer* if it is not programmable; it can perform only the arithmetic and mathematical functions listed on its keyboard.

On the other hand, a computer is a very general purpose symbol manipulator with the capability of solving any problem whose solution can be expressed in terms of elementary operations and conditional branching (within the limitations of its memory and its speed). Therefore, the concept of a **program**—a list of instructions that direct its operation—is fundamental to the concept of a computer. A more accurate model of the computer concept is shown in Figure 1.1. Here we have a person seated at a desk solving problems using manuals and a calculator. An unlimited supply of scratch paper is available and the pencil has an eraser that never wears out. The person and the calculator form the *CPU* of this computer; the paper and manuals, the *memory*; and the in and out boxes, the *input* and *output* devices, respectively. Note that our model gives no credit to the human memory.

This model also illustrates that the central processing unit, or CPU, may be separated into two subcomponents: the **control unit** and the **arithmetic unit**. The control unit is the person; he or she reads instructions from the input box, consults manuals, performs calculations using the calculator, and makes decisions regarding what to do next. In other words, the person directs the flow of the calculations based on the program. The program, or list of instructions, was provided to the control unit through the input box, and parts of it may also be in manuals.

The arithmetic unit is the four function calculator. It is sometimes called an **arithmetic logic unit,** or ALU. In this unit data is transformed by various, but trivial, operations. The CPU is capable of only the simplest operations: performing the four arithmetic operations, performing the logical operations of *and, or,* and *not*; shifting numbers left or right; making simple comparisons—things that most grade-school pupils can do without hesitation.[1] The arithmetic unit cannot, as a single operation,

[1] Some ALUs can only add and subtract.

Figure 1.1
A realization of a computer having both organic and electronic components