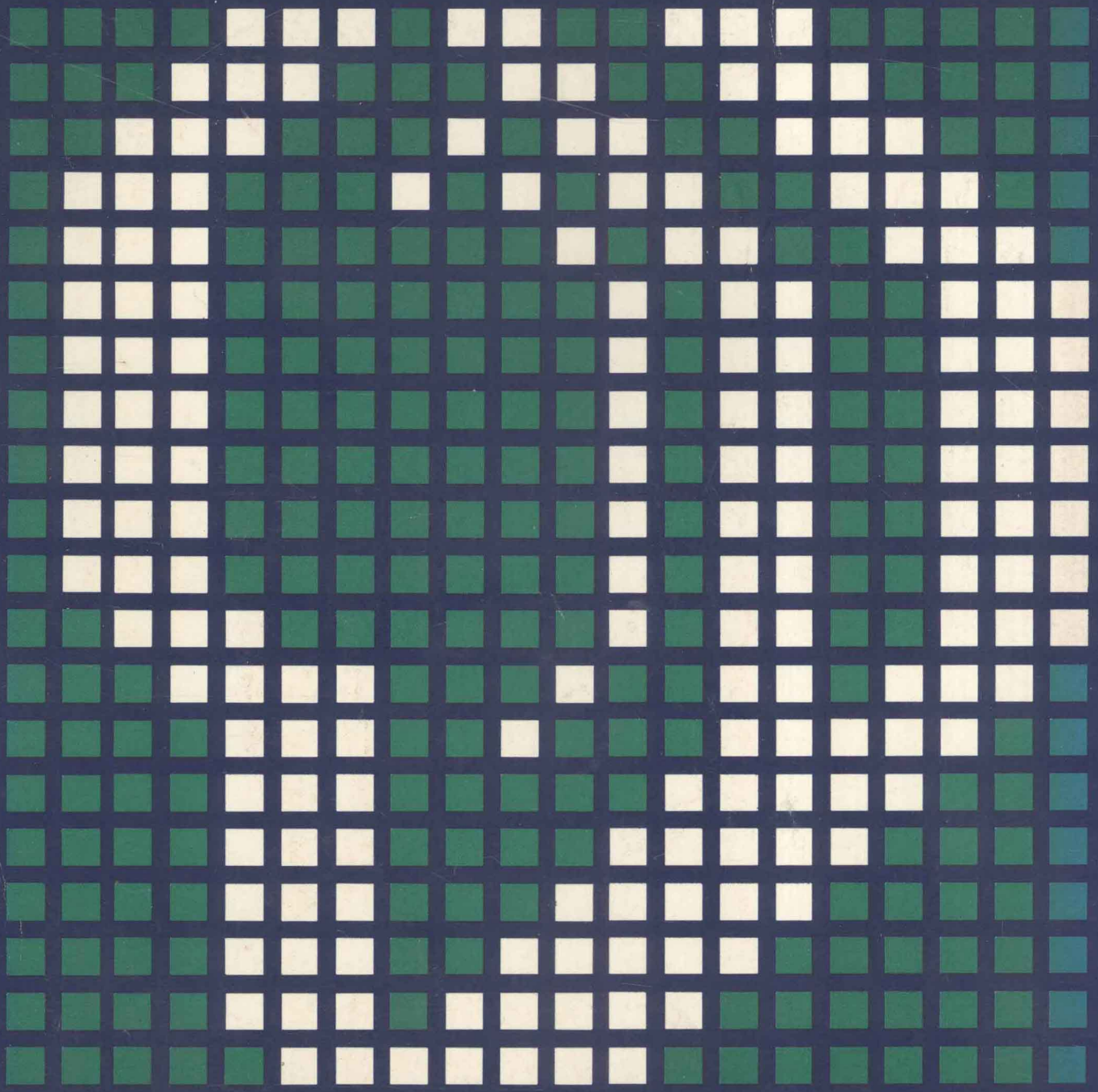
Assembly Language Programming for PDP 11 and LSI 11 Computers

an introduction to computer organization

Edouard J. Desautels



Assembly Language Programming for PDP 11 and LSI 11 Computers

an introduction to computer organization

Edouard J. Desautels
University of Wisconsin / Madison

wcb
Wm. C. Brown Company Publishers
Dubuque, Iowa

wcb group

Wm. C. Brown Chairman of the Board
Mark C. Falb Executive Vice President

Book Team

Alan R. Apt Editor
Julie A. Kennedy Production Editor
James M. McNeil Designer
Kevin J. Pruessner Design Layout Assistant
Faye Schilling Visual Research Editor
Mavis M. Oeth Permissions Editor

wcb

Wm. C. Brown Company Publishers, College Division

Lawrence E. Cremer President
David Wm. Smith Vice President/Marketing
David A. Corona Assistant Vice President/Production Development and Design
Marcia H. Stout Marketing Manager
Janis M. Machala Director of Marketing Research
William A. Moss Production Editorial Manager
Marilyn A. Phelps Manager of Design
Mary Heller Visual Research Manager

Library of Congress Catalog Card Number: 81–70686

ISBN: 0–697–08164–8

Printed in the United States of America

Assembly Language Programming for PDP 11 and LSI 11 Computers

an introduction to computer organization

This book is dedicated to my wife, Jeannine, and my children, Francine, Nicole, and Philip, for their help and encouragement.

preface

The goal of this book is to help readers who have some experience using computers to develop a deeper understanding of how they compute. The very popular PDP-11 computer (and its look-alike, the LSI-11) are used to illustrate the concepts involved.

This is a very substantial book. Don't let it overwhelm you. Among other virtues, it is essentially self-contained. This means you do not have to keep looking through the PDP-11 MACRO-11 reference manual (138 pages plus appendixes), or the PDP-11 Processor Handbook (468 pages plus appendixes), or the PDP-11 Peripherals Handbook (435 pages), or the PDP-11 Terminals and Communications Handbook (344 pages), to understand the topic at hand. One of the major advantages of having all this material integrated in one volume is that it can all be extensively cross-referenced in the index. The index is one of the most important parts of this book. The index is presented in four parts. The first two parts provide a quick reference to the PDP-11 instructions and the MACRO-11 directives. The third part consists of special character references, since these often have a critical role in the writing of computer programs. The fourth part consists of a standard but very comprehensive index. The instructions and directives are repeated in the index for the benefit of the readers may not yet appreciate these distinctions.

The material is sequenced so that the key topics can be studied even in the shortest typical quarter. Each instructor or reader can then select from among the many other topics for semester courses, honors courses, and independent (self-study) courses. Some examples of course organization are as follows.

1. Short course covering only fundamentals:
   chapters 1–9; 10 from beginning through Hard-Copy Terminals; 11; 12 (macros only); selected topics from other chapters, as desired.
2. Course for those familiar with some other machine language:
   chapters 1 (first two sections); 3–19.
3. Course emphasizing software concepts:
   chapters 1–9; 10 and 11 (selected topics); 12; 14 and 15 (selected topics); 18–19.
4. Course emphasizing hardware concepts:
   chapters 1–11; 12 and 13 (selected topics); 14; 15 (selected topics); 16–19.

Once the fundamentals have been covered, the instructor or reader can pick and choose among many essentially independent topics. Thus, in chapter 12 assembly-time conditional directives are treated independently of macros, and vice versa. Chapter 14 first discusses high speed input and output; the later sections on spooling, performance, analog data, processor traps, etc., can be included or not, as desired. Chapter 15, "Selected Topics" has a similar structure. Any of the topics may be included in a course or not, as desired.

The programming examples in the text are supplemented by programming examples in the solutions to the over 280 end-of-chapter exercises; the selected solutions, at the end of the book, cover questions that range from simple review exercises to thought-provoking problems.

We hope readers will find the bibliographic essay in the last chapter more interesting and useful than a conventional annotated bibliography.

The goal of this text is not to make readers experts at PDP-11 programming, but to lay the foundation that is essential for any one who wishes to pursue the study of computer organization, computer architecture, the design of compilers for high level languages, and many other facets of computing. This foundation will be very helpful for those who have personal computers of any kind and want to find out what makes them tick.

We assume that readers have programmed some computer in some high level language. This experience should include having used two-dimensional arrays and procedures or subroutines. Readers are therefore assumed to be familiar with the program development cycle: problem definition, data definition, algorithm selection, implementation, documentation, optimization, testing, and enhancement.

Readers must have access to a DEC PDP-11 or an LSI-11 or some system which uses either CPU (any model will do) and which provides some version of the MACRO-11 assembler. Some of the systems which could be used, besides those sold directly by DEC, include the Heath/Zenith H11 and the Terak 8510. We are convinced that merely reading about using machine language and assembly language is not a substitute for writing and running programs at this language level.

The exercises at the end of most chapters are important. Readers should take the time to at least think through an approach to solving the more demanding problems if time does not permit writing out and possibly programming the solutions. Some of the exercises introduce new ideas, new terms, new tools, and useful algorithms.

Material is presented in a carefully thought-out sequence. Readers are presented with relevant factual information and background, but not so much as to overwhelm them. Learning the details of a computer's instruction set can be as deadly as reading a dictionary if the presentation does not take human nature into account. We have partitioned topics into manageable units, interspersed with other topics, and we take advantage of opportunities to use simpler concepts before proceeding to more advanced ones. The presentation is also enlivened by newspaper and magazine articles on computer-based system malfunctions whose cause is very likely to be related directly to the topic at hand. Some of the articles discuss broader issues such as the legal challenges facing the marriage of computing and telecommunications services.

It is important to note that we first examine machine language before we approach assembly language. We recognize that the assembler is a very useful tool, and make extensive use of it in the text. But in the beginning, the assembler and the assembly process obscure what is going on at the machine level just as much as do high level languages. After a brief but illuminating venture into absolute coding in machine language, we can then appreciate and exploit symbolic programming as a means to gaining more insight into computing.

We do not advocate using machine language or assembly language as a general purpose problem-solving tool. These low level languages are like the ancient Latin and Greek that people were supposed to study as a means of gaining insight into the structure of contemporary English. For instance, the text often uses octal memory dumps, and we expect readers to be able to interpret these. These are just a means to an end—to understand what the computer was doing, stripped of all layers of software. Anyone who concludes that octal memory dumps are an efficient tool for debugging programs written in a high level language has missed the point.

Learning about computing can be enjoyable and addictive, particularly if you have sufficiently ready access to a computer so you can try all the "what if" situations that should be popping through your mind as you proceed.

Notes to the Instructor

Our practice has been to assign computer exercises beginning with the first week of class and to have one assignment immediately follow another continuously throughout the semester. Depending on the class size, the computing resources available, the magnitude of the assigned work, and the assistance provided in grading the programs in a timely fashion, from four to eight programming assignments are made in a 13–16 week semester.

Typically, the first assignment involves finding the laboratory, verifying that one can log in, that one can create and modify a small file using the on-line editor. This course has been conducted in batch-processing mode (using a self-service PDP-11); it has also been run with RT-11, RSTS, and UNIX. In those cases it is advisable to also have a bare PDP-11 for one-at-a-time hands-on use, if the multi-user system cannot be taken off the air.

A typical second assignment has students design and implement a small machine-language program, coded in absolute octal. The students use the instruction subset of chapter 2 and installation-specific instructions on loading and running this program. If a hands-on machine is available, then it can be keyed in at the console and run. For many students this may be the only chance they will ever have to completely control a computer, and this experience gives them a great deal of confidence.

The next assignment has students use a few of the PDP-11 addressing modes while writing their first MACRO-11 program. They are generally given a file with instructions on assembling, linking, and executing it. Then they are told to replace the innards with their own code and repeat the process. This is a painless way to get them introduced to seeing and

using tools such as a contingency post-mortem dump, a memory dump, and a register dump, with the setup required for their use (e.g., .MCALLs, .GLOBLs, invocations, etc.).

The next assignments can involve number representation transformations—say, in implementing a very simple two-function decimal calculator. Students may be asked to implement it using ASCII string-to-binary conversion or programmed BCD arithmetic. The more ambitious students might be asked to write the machine-language loading program (in assembly language) and to propose and implement extensions to it. Similarly, all of the other tools used in the class could themselves be the subject of assignments: students take pride in constructing their own dump programs. Many students have a mental block regarding the transformations from bits in a memory to octal digits on paper. Having them write a dump routine, perhaps adding ASCII interpretations to it, is very salutary. It further develops self-confidence and a sense of knowing what is going on, knowing that you could build up a usable system even if you had nothing but a bare machine. If hands-on access is possible, it is very instructive for students to estimate (by hand) what their program execution time should be, then devise an experiment to actually measure this time.

For the assignment just described, students would need to have access to the appropriate PDP-11 or LSI-11 Processor Handbook, or at least to the relevant instruction timing information. This textbook is otherwise sufficiently self-contained that it should not be necessary for students to buy either the Processor Handbook or the MACRO-11 Reference Manual. However, copies should be made available as references. Many of the problems given at the end of each chapter could be used as the basis for programming assignments.

Instructors sometimes regard teaching a course such as this as a way to give students an opportunity to work on very large projects. We think it is entirely inappropriate to assign large projects to be programmed in assembly language. That is the kind of experience students should be getting when using high level languages. The point of writing programs in machine and assembly language in this course should be to develop understanding of some fundamental ideas: indexing, indirection, manipulation of a stack, mastering the overflow bit, fielding an interrupt, etc. These objectives are more likely to be realized by assignment of smaller, more narrowly focussed problems, and more of them.

Careful readers will appreciate that by the time they reach the end of the text they will know how to design and implement each of the basic tools they have been using from the beginning. This should help dispel much of the mystery that shrouds computing.

I would like to thank the following reviewers for their constructive suggestions: William Bregar, Oregon State University; Linda Eshleman, Western Maryland College; Gordon Fish, Bucks County Community College; Bryan Hansche, Arizona State University; Alex Nichols, Cleveland State Community College; Michael Schneider, University of Minnesota; Abraham Silbershatz, University of Texas at Austin; and Larry Symes, University of Regina.

Edouard J. Desautels

contents