



# PROGRAMMING ASSEMBLER LANGUAGE

**Peter Abel**

*British Columbia Institute of Technology*



**RESTON PUBLISHING COMPANY, INC.**

**Reston, Virginia**

*A Prentice-Hall Company*

**Library of Congress Cataloging in Publication Data**

Abel, Peter

Programming assembler language.

Includes index.

1. Assembler language (Computer program language)
2. IBM 360 (Computer)—Programming. 3. IBM 370 (Computer)—Programming. I. Title.

QA76.73.A8A23 001.6'424 79-255  
ISBN 0-8359-5658-X

© 1979 by  
Reston Publishing Company, Inc.  
*A Prentice-Hall Company*  
Reston, Virginia 22090

All rights reserved. No part of this book may be reproduced in any way, or by any means, without permission in writing from the publisher.

10 9 8 7 6 5 4 3 2

Printed in the United States of America.

# **PROGRAMMING ASSEMBLER LANGUAGE**

# PREFACE

Assembler language is the fundamental “low-level” language of the IBM 360 and 370 computers. As such, it is directly translatable into machine language; thus, one Assembler instruction typically generates one machine code instruction. “High-level” languages like COBOL and PL/I are easier to learn. Why then an emphasis on learning Assembler language? An understanding of Assembler language can help the programmer in a number of ways:

- A knowledge of Assembler can facilitate learning of any other language, including “high-level” languages and other assembly languages. And with a background in Assembler, the user can more clearly understand what the computer is doing.
- A knowledge of Assembler can help the programmer become more efficient. High-level languages like COBOL and PL/I can be deceptive, and appear to execute in some mysterious fashion. A programmer familiar with Assembler can code high-level languages with an understanding of what machine code they generate, and what is the more efficient technique. For example, why in COBOL does the use of COMPUTATIONAL, COMPUTATIONAL-3, and SYNC have considerable effect on the program’s efficiency? What is the significance in PL/I of Decimal Fixed, Aligned, and Defined? With knowledge of Assembler, a programmer can examine the generated code to determine more efficient ways to write certain routines.
- Although most high-level languages provide extensive debugging aids, there are times when the programmer needs to delve into the generated machine code or examine storage dumps.
- Programs written in Assembler may be considerably more efficient in storage space and execute-time, a useful consideration if such programs are run frequently.
- Some advanced areas, such as technical support and telecommunications, require an extensive knowledge of Assembler.

Although the material in this text has been used successfully as an introduction to programming, most educational institutes would not teach Assembler as an introductory language. Generally, the concepts of logic and programming style are easier to learn when there is less need for concern with rigorous rules and field sizes. The text does not, however, assume that the reader has had much, if any, programming experience. The approach of the text is to introduce simple processing using card

input and printer output, first with character data only, and then with simple decimal arithmetic, all by Chapter 4. In this way, packed (decimal) data and editing are introduced early, and the user is soon writing quite realistic programs.

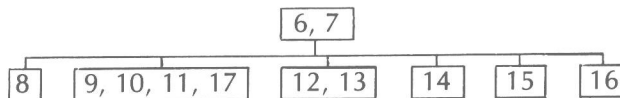
The book should provide both a practical guide for the Assembler student and also may act subsequently as a useful reference. These two objectives are accomplished by:

1. A step-by-step progression of material, from simple processing through to complex. There are many practical examples of complete and partial programs to illustrate concepts as they are introduced.
2. Chapters organized by logical topics, such as character data, packed, binary, input/output. The user can concentrate on mastering one programming area at a time, and most related material is contained in its own chapter.

The complexities of base/displacement addressing and file definition are delayed through use of several simple macros, similar to those used in many colleges. The text drops these macros by Chapters 6 and 7, where the technical material is covered in detail. Appendix E provides a listing of the macros for those who want to catalogue them on their own system.

The two major IBM operating systems are DOS and OS. The text covers the differences between them, giving examples for both.

It is possible to proceed through the text by several routes. Chapters 0 through 3 are fundamental, and the normal steps would be to continue sequentially with decimal arithmetic in Chapters 4 and 5. It is recommended next to cover the important material in Chapter 6 on base/displacement addressing and the instruction format, and then the elements of input/output in Chapter 7. By this point the user should be capable of coding some quite advanced programs. Chapter 8 on Programming Strategy could be covered in part or total, perhaps the sooner the better to get the user into subroutine logic. Following Chapter 8, the chapters need not all be covered sequentially. The following diagram indicates related chapters in boxes that may be taken in any sequence following Chapters 6 and 7:



Chapters 9, 10, and 11 develop related material on processing binary data. Chapter 12 on Magnetic Tape introduces basic material required for an understanding of Disk Storage in Chapter 13. To complete Chapter 14 (Macro Writing) would require some familiarity with the material in Chapter 9. Anyone interested in linking separately assembled programs could attempt Chapter 15 directly after Chapter 7, perhaps referencing Chapter 9 for some basic binary operations. Chapter 16 on Operating Systems is presented for general useful information, although not entirely related to Assembler programming as such.

Among the users of earlier versions of this text, many have worked ahead of the course, experimenting with binary operations, macro writing, and subprogram linkage. Such motivation is certainly commendable and should be encouraged.

The IBM manuals concerned with the material in this text require a bookshelf about five feet wide. Readers should not expect, therefore, that this or any other single book will provide all there is to know about the Assembler language and related topics. Eventually the IBM manuals have to be referenced for current detailed information. The following IBM manuals or their equivalent are especially recommended:

IBM FORM	TITLE
GA22-7000	IBM System/370 Principles of Operation. (370 system organization, machine instructions, input/output.)
GC33-4010	OS/VS-DOS/VS-VM/370 Assembler Language. (Assembler statements and macros.)
GC33-5373	DOS/VS Supervisor and Input/Output Macros.
GC28-6646	OS Supervisor Services and Macro Instructions.
GC26-3746	OS Data Management Services Guide.
GC26-3794	OS Data Management Macro Instructions.

Other useful manuals include those on Job Control, disk file organization, tape labels, disk labels, and the operating system.

*Peter Abel*



# ACKNOWLEDGMENT

The author is grateful for the assistance from all those who contributed typing, reviews, and suggestions, and to IBM for permission to reproduce some of their copyrighted materials. The following materials are printed with permission, and with modifications from publications copyrighted in 1966, 1970, 1972, 1973, and 1974 by International Business Machines Corporation as IBM form numbers GA22-7000, GX20-1850 and GC20-1649: Figures 2-2, 13-2, 16-3, Appendix A, and Appendix C.

# CONTENTS

PREFACE	xi
ACKNOWLEDGMENT	xv

## **PART I      INTRODUCTION TO THE COMPUTER AND THE ASSEMBLER**

<b>0</b>	<b>BASIC COMPUTER CONCEPTS</b>	<b>3</b>
	Introduction, 3	
	The Computer System, 4	
	Input/Output Devices, 6	
	Binary Number System, 8	
	Bits and Bytes, 9	
	S/360 and S/370 Organization, 10	
	Operating Systems and the Supervisor, 13	
	Hexadecimal Representation, 14	
	Problems, 15	
<b>1</b>	<b>THE ASSEMBLER</b>	<b>17</b>
	Languages, 17	
	The Assembler, 18	
	The Assembler Coding Sheet, 21	
	Assembler Instruction Statements, 24	
	The Assembler Card, 26	
	Coding Conventions, 26	
	The Assembled Program Listing, 29	
	Assembler Diagnostic Messages, 30	
	Problems, 32	
<b>2</b>	<b>PROGRAM EXECUTION</b>	<b>34</b>
	Program Execution Statements, 34	
	Condition Code, 38	

Input Data, 40  
 Flowchart Logic, 44  
 Job Control, 47  
 Problems, 47

## **PART II BASIC ASSEMBLER CODING**

- 3 PROGRAMMING FOR CHARACTER DATA 51**
- Declaratives, 51
  - Character Instructions, 58
  - Storage-to-Storage Format, 58
  - Move Characters—MVC, 58
  - Compare Logical Character—CLC, 62
  - Storage Immediate Format, 65
  - Literals, 66
  - Equate Symbol—EQU, 68
  - Sample Program—Read and Print Customer Records, 69
  - Debugging Tips, 73
  - Problems, 73
- 4 DECIMAL DATA AND ARITHMETIC I 76**
- Hexadecimal Constants, 76
  - Move Numeric (MVN) and Move Zones (MVZ), 77
  - Zoned Decimal Data, 78
  - Packed Decimal Data, 80
  - Packed Operations, 82
  - Formatting the Print Area, 90
  - Sample Program—Budget Statement, 91
  - Debugging Tips, 96
  - Problems, 98
- 5 DECIMAL ARITHMETIC II 101**
- Editing—ED, 101
  - Multiply Packed—MP, 106
  - Move With Offset—MVO, 108
  - Divide Packed—DP, 110
  - Sample Program—Inventory Update, 113
  - ORG—Set Location Counter, 119
  - PDUMP—Partial Dump of Main Storage, 121
  - Debugging Tips, 123
  - Problems, 123

<b>6</b>	<b>BASE REGISTERS AND INSTRUCTION FORMAT</b>	<b>126</b>
	The General Purpose Registers, 126	
	Base Register Addressing, 127	
	Instruction Format, 130	
	Control Sections, 135	
	Assigning Base Registers, 136	
	Loading the Base Register, 137	
	OS Initialization, 139	
	Technical Note Re: BCR and BC, 141	
	Debugging Tips, 142	
	Problems, 142	
<b>7</b>	<b>INPUT AND OUTPUT</b>	<b>144</b>
	Input/Output Control System, 145	
	Imperative Macros, 146	
	The DOS DTF File Definition Macro, 150	
	The OS DCB File Definition Macro, 153	
	Locate Mode, 156	
	Device Independence Under DOS, 158	
	Abnormal Termination, 160	
	Debugging Tips, 160	
	Problems, 161	
<b>8</b>	<b>STRATEGY, STYLE, AND STANDARDS</b>	<b>162</b>
	Programming Objectives, 162	
	Subroutines and Linkage, 163	
	Programming Style, 166	
	Program Documentation, 167	
	Debugging Tips, 182	
	Problems, 182	

## **PART III      BINARY OPERATIONS**

<b>9</b>	<b>REGISTERS AND BINARY PROGRAMMING</b>	<b>185</b>
	Binary Data Representation, 186	
	Binary Constants, 187	
	Conversion of Decimal and Binary Data—CVB and CVD, 191	
	Loading Registers—L, LH, LR, LM, LA, 192	
	Store Register Operations—ST, STH, STM, 196	
	Binary Arithmetic—A, S, AH, SH, AR, SR, 198	
	Binary Comparison—C, CH, CR, 199	
	Multiplication—M, MH, MR, 201	

Register Shift Instructions, 203  
 Binary Division—D, DR, 205  
 Conversion of Double-Precision Binary to Decimal Format, 208  
 Sample Partial Program—Finance Charge Rebates, 210  
 Debugging Tips, 210  
 Problems, 212

## 10 EXPLICIT USE OF BASE REGISTERS 213

Operations Explicitly Using Base Registers, 215  
 Tables, 218  
 Direct Table Addressing, 224  
 Sorting Data in Storage, 226  
 Binary Search, 227  
 Sample Program—Calculate Stock Value, 227  
 Debugging Tips, 234  
 Problems, 235

## 11 LOGICAL OPERATIONS AND BIT MANIPULATION 237

Logical Operations, 237  
 Boolean Logic, 241  
 Other Operations, 244  
 Binary Operations—Summary, 255  
 Problems, 255

# PART IV EXTERNAL STORAGE

## 12 MAGNETIC TAPE 259

Uses of Magnetic Tape, 260  
 Magnetic Tape Characteristics, 260  
 DOS Tape Programming Example—Creating a Tape File, 262  
 OS Coding for Magnetic Tape, 265  
 Tape File Organization, 265  
 IOCS For Magnetic Tape, 270  
 Variable Length Records, 272  
 Problems, 276

## 13 DIRECT ACCESS STORAGE 277

DASD Characteristics, 278  
 File Organization, 282  
 DOS Programming Example—Creating a Sequential  
 Disk File, 282  
 OS Program Examples, 284

Disk Labels, 284  
Indexed Sequential File Organization, 287  
Processing DOS Indexed Sequential Files, 290  
Processing OS Indexed Sequential Files, 296  
Problems, 299

## **PART V      ADVANCED TOPICS**

- 14      MACRO WRITING      303**
- Writing Macros, 304  
Conditional Assembly Instructions, 310  
Other Instructions, 313  
Set Symbols, 314  
System Variable Symbols, 319  
Global Set Symbols, 320  
Extended Example—Table Look-Up Macro, 321  
Problems, 322
- 15      SUBPROGRAMS AND OVERLAYS      323**
- CSECT—Control Section, 323  
DSECT—Dummy Section, 324  
Subprogram Linkage, 326  
Linking Two Control Sections, 329  
Passing Parameters, 331  
Linking Phases, 333  
Overlay Considerations, 338  
Problems, 339
- 16      OPERATING SYSTEMS      340**
- Operating Systems, 340  
The Supervisor, 344  
The Program Status Word—PSW, 347  
Interrupts, 348  
Channels, 350  
Physical IOCS, 353  
Problems, 355
- 17      FLOATING-POINT OPERATIONS      357**
- Floating-Point Formats, 357  
Declaratives, 362  
Floating-Point Registers, 363  
Floating-Point Instructions, 364  
Conversion From Packed to Float, 374

Conversion From Float to Packed, 374  
Problems, 376

## APPENDICES

A	HEXADECIMAL AND DECIMAL CONVERSION	377
B	PROGRAM INTERRUPTS BY CODE	379
C	360/370 INSTRUCTION SET	382
D	DOS AND OS JOB CONTROL	384
E	SPECIAL MACROS: INIT, PUTPR, DEFCD, DEFPR, EOJ	388
	INDEX	392

## **PART I**

# **INTRODUCTION TO THE COMPUTER AND THE ASSEMBLER**

